SPARQL on the Open, Decentralised Web

Kjetil Kjernsmo

November 22, 2016

Thesis submitted for the degree of Philosophiæ Doctor
Abstract

This dissertation discusses a broad range of problems concerning the use of the SPARQL query language on the open, public Web. It is motivated from seeing decentralisation of data and infrastructure as an important social goal, and SPARQL as an enabling technology to solve problems using the Semantic Web. The dissertation makes contributions in hypermedia, where RDF is used to create a format that can tell humans and machines alike how to manipulate resources on the Web; philosophy of science, where important foundational problems around how to create valid knowledge and objectivity are discussed; statistical methods that better satisfies the requirements from philosophy of science than current practice; how to improve developer efficiency with novel programming paradigms; and finally how caching infrastructure in the Internet may be used to make query answering across the Web more robust.
Acknowledgements

First and foremost it is my pleasure to thank my caring wife Hege, whose invaluable support and affection has been critical to carry through this project, and then my children Synne, Eivind and Marius, two of whom have been born while working on this, for putting up with a father that sometimes has his mind in the computer. My deceased parents Ragnhild and Dag has had a major impact with their stimulus throughout childhood and their enduring support. Arguably, allowing me to use the 1985 PC with no graphics adapter to write programs was the start of this project.

My advisers Martin Giese and Arild Waaler deserve a big thanks for kick-starting the project, help adjusting its course over the years and thorough and enlightened criticism of manuscripts prior to submission.

I would like to thank my co-authors: In particular Gregory Todd Williams, who have not only co-authored a paper, but written much of the underlying software I use, formed the standards that this work builds on, and been a great discussion partner and friend. John S. Tyssedal deserves a big thanks for agreeing to write a paper where we started out with very little common ground. Discussions with Ruben Verborgh always creates the right amount sparks to be enlightening, and it has been a pleasure to become friends.

It is also a pleasure to thank the entire Semantic Web community for their openness and welcoming attitude. Whether you come as a hacker or want to pursue standardisation, you are likely to meet people like Dan Brickley, Charles Nevile, Andy Seaborne, Steve Harris, Phil Archer or Ivan Herman, all of whom have been important for my understanding of the Semantic Web. The academic community, with their important ISWC and ESWC conferences are equally welcoming, and has culture of sharing insight that many other academic communities would envy. Many thanks are due to Oscar Corcho for first integrating me into the community, Maria-Esther Vidal and Jacco van Ossenbruggen for encouraging me to further pursue the speculations around the role of empirical methods, and general support. Jürgen Umbrich and Axel Polleres deserve a big thanks for welcoming me in Vienna and many discussions.

I have made many friends in the community, for which I am thankful. Of these, the following in no particular order have in one way or the other shaped this work through personal contact beyond published papers: Olaf Görlitz, Norman Gray, Ansgar Scherp, Frank Van Harmelen, Olaf Hartig, Javier David Fernández García, Aidan Hogan, Maribel Acosta, Josef Noll and Philip Stutz.
The Perl community have also been important in developing the ideas and the code of this thesis. In addition to Gregory Todd Williams, I would like to thank Toby Inkster, Patrick Hochstenbach, Chris Prather, Kip Hampton, Robin Berjon, Shawn M. Moore, Michael Nachbaur and Jonas Smedegaard.

My industry colleagues David Norheim and Robert Engels have also been important in shaping my technical understanding and my ideas.

Finally, many thanks to Helen Murray for linguistic assistance.
# Contents

1 Introduction .............................................. 1
   1.1 Semantic Web Technology ............................ 1
       1.1.1 Basic technology ............................. 1
       1.1.2 Interacting With Data ....................... 3
       1.1.3 Hypermedia .................................. 5
   1.2 Research Problems Overview ....................... 6
   1.3 Contributions ........................................ 7
       1.3.1 Hypermedia .................................... 7
       1.3.2 Design of Experiments ....................... 7
       1.3.3 Development problems ........................ 8
       1.3.4 Survey of HTTP Caching ..................... 8
       1.3.5 Philosophy of Science ...................... 8
       1.3.6 Software ...................................... 8

2 A View of the Semantic Web ............................... 9
   2.1 Overview ............................................. 9
   2.2 Semantic Web – A Definition? ..................... 10
   2.3 Objectivity of the Study .......................... 11
   2.4 A Personal History .................................. 12
   2.5 Motivation ............................................ 16
       2.5.1 Cruise Altitude View ......................... 16
       2.5.2 Hypermedia .................................... 18
       2.5.3 Developer Friendliness ...................... 19
       2.5.4 Philosophy of Science and Statistical Methods ........ 19
       2.5.5 Caching in the Internet ..................... 20
### 3 Preliminaries and Related Work

3.1 Preliminaries ........................................... 21
  3.1.1 Caching in the HTTP Standard ................. 21
  3.1.2 Topics in query evaluation ....................... 22

3.2 Related Work .............................................
  3.2.1 Hypermedia ........................................... 23
  3.2.2 Query Answering With Cache ....................... 24

### 4 Problems and Solutions

4.1 Problems ................................................... 29
  4.1.1 Performance and Stability Problems ............... 29
  4.1.2 Problems of Shared Understanding ................. 31
  4.1.3 Problems With Evaluations .......................... 32
  4.1.4 Development Problems ............................... 34

4.2 The Papers in Context ....................................
  4.2.1 Read-write Hypermedia ............................. 36
  4.2.2 Design of Experiments ................................ 37
  4.2.3 Pushing Complexity Down the Stack ................ 38
  4.2.4 Survey of HTTP Caching ............................. 38
  4.2.5 Philosophy of Science ............................... 39

4.3 Unpublished Work on Query Caching ..................... 40

4.4 Future Work ................................................ 41

4.5 Conclusions ................................................. 43

### 5 Prefetching SPARQL Query Cacher

5.1 Introduction ............................................... 45
  5.1.1 Scenario ............................................... 45
  5.1.2 Architecture ......................................... 48
  5.1.3 Understanding the Query Planning ................. 48
  5.1.4 Analysis for prefetching ............................. 52

5.2 Implementation ............................................
  5.2.1 The Attean Framework ............................... 53
  5.2.2 The Caching Proxy .................................... 55
  5.2.3 Cost Model ............................................ 58

5.3 Evaluation .................................................
  5.3.1 Actual Evaluation .................................... 61
  5.3.2 Planned Evaluation ................................... 63
Chapter 1

Introduction

1.1 Semantic Web Technology

The World Wide Web, or just the Web for short, is a well-known global information space invented by Tim Berners-Lee in 1989, that further emerged in the 1990s. It is characterised first and foremost by its universality, anyone can set up a computer, connect it to the Internet and start serving data or documents from it, and further adapt it to their purpose.

The Semantic Web is an extension of the Web that has been under development since 1997\(^1\) to extend the Web with languages for expressing information in a machine processable form [10].

1.1.1 Basic technology

The Semantic Web is built on a number of specifications that has been developed under the auspices of the World Wide Web Consortium. The core technology is known as the Resource Description Framework (RDF). The framework is defined in a suite of specifications, defining concepts and abstract syntax, semantics, and several concrete syntaxes.

Conceptually, an RDF statement is a triple, where the terms are a subject, a predicate and an object. The building blocks for these triples are literals, blank nodes and Internationalized Resource Identifiers (IRI). IRIs builds on the familiar URL: Uniform Resource Locator, informally known as a Web address, e.g. https://www.w3.org/RDF/ is an address for a document about RDF. It is

\(^1\)The first working group draft of the RDF specification is dated 1997-08-01, see http://www.w3.org/TR/WD-rdf-syntax-971002/
On the Semantic Web, the URL is first generalised to a Uniform Resource Identifier, URI, that can not only name a document on the Web, but any resource. A resource can be a physical thing, like a person, or an abstract thing, like the type of weather at some place at some time, numbers, strings, etc. Then, URIs are extended to IRIs to enable the use of any character in them. Literals are strings with a datatype (denoted by an IRI), and optionally a language tag. Finally, blank nodes are unnamed nodes. RDF defines which of these building blocks may be used for what term: The subject may be a IRI or a blank node, the predicate only a IRI, whereas the object may be a literal, blank node or IRI.

The use of IRIs makes it possible to view RDF statements as a directed graph, where the subject IRI is a vertex, the predicate is an edge to an object vertex.

For historical reasons, the first syntax that was standardised for RDF was an syntax based on the tree-structured Extensible Markup Language (XML). The simplest standardised syntax is known as N-Triples, where each statement is written on a single line, where each IRI is written out in full. The basic building blocks of N-Triples has been reused to create a compact and natural text form syntax, known as Turtle. Turtle has gained traction, and is well established as a common syntax. A simple graph expressed in Turtle could be:

**Example 1. Turtle syntax**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix dct: <http://purl.org/dc/terms/>.

<http://www.kjetil.kjernsmo.net/foaf#me> a foaf:Person ;
 foaf:name "Kjetil Kjernsmo" ;
<http://folk.uio.no/kjekje/2013/iswc.pdf> a foaf:Document ;
 dct:title "Introducing Statistical Design of [...]" ;
 dct:creator <http://www.kjetil.kjernsmo.net/foaf#me>,
 <http://[...]/person/john-s-tyssedal> .
```

The example opens with prefix declarations, to allow abbreviating IRIs, e.g.

---

2A working draft of the RDF Semantics specification dated 23 January 2003, see [https://www.w3.org/TR/2003/WD-rdf-mt-20030123/](https://www.w3.org/TR/2003/WD-rdf-mt-20030123/) described the graph structure as a “partially labeled directed pseudograph with unique node labels”.

---
the predicate foaf:name is an abbreviation of the IRI http://xmlns.com/foaf/0.1/name. The IRI http://www.kjetil.kjernsmo.net/foaf#me is a name for this author. The a predicate is standards-defined abbreviation for http://www.w3.org/1999/02/22-rdf-syntax-ns#type, which is used for declaring the subject an instance of a class, in this case foaf:Person. Then, the foaf:made predicate is used to link to an author-published version of one of the publications of this dissertation, and further information is then provided about this publication, including a link to the co-author. As we see, semicolons are used to separate statements that share a subject, commas separate statements with the same subject and predicate, while periods terminate a statement. [...] is just to shorten the lines in the example and is not a part of the syntax.

Note that all the IRIs uses the HTTP scheme. The Hypertext Transfer Protocol (HTTP) is the main application protocol used on the Internet for the Web, and is also the protocol used most for the Semantic Web. It is a request-response protocol in a client-server computing model, but also accommodates for intermediate proxies. HTTP messages consist of a header, typically with metadata, and a body, which is the content itself. The standard defines several headers that can be used to control caching on either the client, the server or an intermediate proxy.

Also, note that all IRIs in this example are dereferenceable, meaning that if an HTTP client is used, a representation of the resource can be retrieved across the Internet. This, along with that it is expressed as RDF, makes the above so-called Linked Data.

HTTP IRIs are not the only possible choice, to the contrary, the scheme and what may be associated with it, such as a protocol, is an orthogonal issue to the data model and syntax, and any IRI scheme can be used.

1.1.2 Interacting With Data

To read and write this graph, several techniques have been developed. For reading, one is to simply parse the string into a suited data structure and use any tool available in the chosen programming language to examine the data structure, or simply write data to e.g. a Turtle string.

A more sophisticated approach is to match parts of the graphs using triple patterns. A triple pattern may contain variables for certain terms, for example:

Example 2. A triple pattern
This triple pattern will match the two triples

**Example 3. Triples matched by triple pattern**

<http://www.kjetil.kjernsmo.net/foaf#me> a foaf:Person .

from the above example (prefixes omitted for brevity). Frameworks in popular programming languages commonly have a method that implements this.

The triple pattern is also a fundamental building block of the SPARQL query language, which is the language that is the main topic of this dissertation. SPARQL is an extensive, standardised query language with both read and write operations. The syntax is near Turtle but as the above triple pattern shows, introduces variables. A set of triple patterns may be used to create a conjunctive query, and along with a filter that can be used to further constrain the query with boolean expressions. Together, they are known as a Basic Graph Pattern (BGP). More advanced parts of the language are built around BGPs.

SPARQL also introduced a mechanism for naming a graph or parts of a graph with a IRI, an approach that was subsequently adopted by RDF. Therefore, RDF is today often implemented not just expressed as a triple, but may be a expressed with a quad.

To execute a query, a SPARQL engine would typically parse the query to create a tree of algebra objects (in the case where an object oriented programming language is used) in the process. Then, a query planner component would traverse the algebra tree, and for each algebra object or branch of algebra objects, the query planner finds various ways to access the data in the underlying store or execute other operations. In many programming frameworks, the only way to access data is through matching a triple pattern. If that is the case and the incoming query consists of just a BGP, the query planner’s task would simply be to match all the triple patterns and join their results, and finally apply any constraining filters.

However, even though the result does not depend on the order of execution of a BGP, the time it takes to find the result often does. For example, consider the following BGP:

**Example 4. A Basic Graph Pattern**

?subject foaf:name ?name ;
  foaf:workplaceHomepage <http://example.com/> ;
  foaf:made ?contribution .
In most cases, if the triple pattern with the foaf:name predicate is evaluated first, it is likely to match many resources, since nearly everything may have a name, and therefore, it does not reduce the number of statements that needs to be searched when matching the next triple pattern by much. Instead, the foaf:workplaceHomepage may be matched first, as we may guess the fact that the object is given will be more restrictive, but that too fails if everyone in the database works for the same organisation and has used this predicate. It is the query planner’s task to choose between equivalent plans to ensure that the most efficient evaluation is chosen. As is clear from this example, it is often insufficient to rely on such heuristics, and therefore, it is common to rely on a statistical digest in the query planner. Still, the task of query planning remains a very complex one, and often, insufficient statistics is available, so that planning is done using heuristics or based on assumptions that do not hold. These are only some of the concerns that affect performance of the overall query engine.

Indeed, detailed knowledge of the data, such as a comprehensive statistical digest may be available close to the database, but is not exposed in ways that higher levels in the technology stack can make use of. In our case, this is especially true for a caching proxy.

More techniques have been developed for writing data. The simplest is to write RDF in one of its serialisations to a file. Basic operations like writing single triples or quads may be done with a framework directly onto a triplestore. Since write operations are specified in HTTP, write operations may use HTTP verbs if they are authorised to do so. Finally, SPARQL has been extended with an update language in version 1.1.

1.1.3 Hypermedia

However, in terms of utility when programming, advanced query languages such as SPARQL may not be needed for many applications, and may represent an overly high barrier to entry.

Hypermedia is the idea that all information needed to drive interaction in an application must be readily available in the request-response dialogue. For human interactions, this has been understood as a requirement for a long time, one has to tell the user how to interact with the application, if not, the user is unlikely to be able to solve the task. Contemporary efforts now extend this to machine-machine interactions.
In this context, hypermedia types is a way to classify what kind of operations a certain media type can help the application perform.

1.2 Research Problems Overview

The present work is located at the confluence of several contemporary efforts in the Semantic Web community. The focus is on query answering with the SPARQL query language, with emphasis on exploiting the World Wide Web, but it touches upon query federation, hypermedia, empirical methods for evaluating performance, standards compliance and even philosophy of science.

The overarching problem this work tries to address is this:

To achieve a Semantic Web where anyone can say anything about anything, and where everyone is enabled to analyse data that have been contributed, there is a difficult balance between centralising infrastructure and decentralising it, and between processing on clients and servers.

A comprehensive treatment of research problems is given in Section 4.1, but to understand the following contributions, a superficial overview is provided:

Many developers find it difficult to interact with RDF, and contributions to alleviate that are summarised in Section 1.3.1.

We have found that in few cases, at least when evaluating SPARQL Engines, the evaluation has had a basis in sound statistics or in philosophy of science and therefore the validity of the conclusions drawn from them should be challenged. We summarise contributions to a new direction to the statistical problem in Section 1.3.2 and a philosophical discussion in Section 1.3.5.

To create a framework that allows the flexible and at the same time efficient manner is difficult. Conventionally, the query engine would break down a query to individual triple pattern matches, which did not allow for much optimisation, or to easily modify the query planning. Our contributions towards these problems are summarised in Section 1.3.3.

As noted, the HTTP standard and the Web architecture allows for caching responses, and in Section 1.3.4 we detail contributions to understand the contemporary usage of these mechanisms. This is particularly interesting in the context
of SPARQL query execution on the Web, as currently, the infrastructure cannot support evaluating arbitrary queries across the Web in a reliable manner. A situation where the client, a caching proxy and a server may share the work when evaluating a query results in a number of research problems that are detailed in Section 4.1.1 and we attempted to contribute an approach to some of these problems in Chapter 5 without reaching any solid conclusions.

1.3 Contributions

In this section, we summarise the contributions of this work, with reference to the detailed problem descriptions in Section 4.1, as well as Section 4.2, where the papers are described and put in context.

1.3.1 Hypermedia

The following contributions were made towards Problem 21 in the paper described in Section 4.2.1:


2. A sketch of a vocabulary for read-write RDF hypermedia.

1.3.2 Design of Experiments

The following contributions were made towards Problem 15 in the paper described in Section 4.2.2:

1. Introduction of a path for critical practice of evaluations, that makes use of contemporary statistical techniques, to establish a practice that can be used to refute assertions on performance.

2. A didactical experiment to help researchers understand the statistics.

3. The novel application of a well-established method in statistics, rarely used in Computer Science, to SPARQL endpoint evaluation.
1.3.3 Development problems
The following contributions were made towards problems 22, 23 and 24 in the paper described in Section 4.2.3:

1. A framework to enable the use of low-level optimisations in databases.
2. Simplification when implementing experimental features in SPARQL.

1.3.4 Survey of HTTP Caching
The following contribution was made towards Problem 10 in the papers described in Section 4.2.4:

1. An understanding of actual usage of caching headers (metadata) on the open Semantic Web.

1.3.5 Philosophy of Science
The following contribution was made towards Problem 18 in the paper described in Section 4.2.5:

1. A provocation to discuss epistemological questions.

1.3.6 Software
In addition to papers, the following contributions were made in terms of Free Software:

1. 5 packages that have been accepted into the recent versions of the Debian GNU/Linux operating system.
2. Further 5 packages that already were accepted in Debian have been enhanced in the course of this work.
3. Contributions to the code base were accepted by another 5 external projects.
Chapter 2

A View of the Semantic Web

2.1 Overview

As noted in Section 1.3, the scope of the study is very broad, ranging from the practical programming details to philosophy of science. With a scope of this breadth, the investigation is shallow, it is hoped that some key insights will provoke further investigation rather than being the end of any conversations.

The introduction provided a very basic view of the technology, this chapter will further discuss some important aspects of the Semantic Web, as it provides guidance for the investigation detailed in the papers.

As such, the following serves to tie the published papers into a whole, provide an overarching rationale and problem statements, overview of the contributions and a discussion of the problematic sides of the study seen from a philosophy of science viewpoint. It is structured as follows: Section 2.2 discusses the possibility of defining the Semantic Web, Section 2.3 discusses the problem of maintaining objectivity, Section 2.4 attempts to satisfy the previous discussion by detailing the author’s personal involvement in the Semantic Web. Section 2.5 has an overview of the motivation behind this work. Section 3.1 provides some further technical details and Section 3.2 puts the present work into the context within the existing literature. Section 4.1 summarises important research problems and details the motivation. Sections 4.2 and 4.3 detail how the contributions address these problems and discusses weaknesses of the papers with the benefit of hindsight. Future work is discussed in Section 4.4. Key insights are finally summarised in Section 4.5.
2.2 Semantic Web – A Definition?

The Semantic Web can be described as a machine-readable Web of Data, defined by a technology stack managed by the World Wide Web Consortium. While individual technologies can be given precise, operational definitions, the above definition is naive and can hardly be used operationally to derive new knowledge. The Semantic Web is a complex, human artefact that must be understood not only in technical terms, but also in social, economic and cultural terms. First, the Web is not only readable, but also writeable. Whether the essential parts of a data integration problem will be solved by e.g. pervasive ontology alignment, a linked open vocabularies approach where many players adopt a fairly consistent set of vocabularies, or by major players that are able to force convergence towards e.g. Schema.org is an example of the social, economic and cultural mechanisms that shape the future of the Semantic Web, see Section 4.1.2 for a more elaborate discussion. Likewise, it may not be the W3C Semantic Web stack at all that achieves success, though unlikely at this point, it may be something in the extension of microformats\(^1\) microdata \(^{32}\), or something else entirely. For a thoughtful criticism of Linked Data with a proposed alternative, see \(^9\), but again, it is likely that social, economic and cultural factors will be as decisive as technical. One may say that, for example microformats cannot help achieve the visions that may be stated for the Semantic Web, but that again emphasises visions, not operational definitions.

One such vision is the article by Berners-Lee, Hendler and Lassila in Scientific American 2001 \(^{13}\), but it does not attempt such an operational definition. Neither \(^{10}\), \(^{12}\) nor \(^{4}\) contain such definitions. However, it is clear that an essential characteristic of the Web, and by extension, the Semantic Web, is its universality and therefore, any actor should be as free as possible to adapt it to their visions and needs. A pursuit of a definition of the term “Semantic Web” would require a philosophical treatment which would be beyond the scope of this thesis, and even with that, may not be a fruitful exercise from a technical perspective.

Therefore, rather than attempt a definition, understanding is better promoted by following the example of \(^{13}\) and declaring a vision.

\(^1\)See \url{http://microformats.org/}
2.3 Objectivity of the Study

A vision has necessarily a strong personal element. This challenges the ideals of objectivity in scientific investigations. A longer discussion of the desirability and attainability of objectivity in science is beyond the scope of this thesis, but objectivity can be challenged from many different angles [52].

The problems are also discussed throughout [17]. It is interesting to consider the example on page 25, where the author describes a method forwarded by Galileo Galilei to measure the diameter of a star. The method was objective in the sense if followed, it would yield the same value today as it did when Galileo employed it. However, it relied on faulty assumptions and is therefore invalid. The same kind of problem may occur in the evaluation of e.g. SPARQL engines. If a benchmark relies on a faulty workload, the application of the benchmark may seem objective to the investigator, but its external validity should obviously be challenged. Moreover, if the benchmark relied on a use case supplied by for example an industrial partner, the validity could be further challenged based on e.g. the restriction on viewpoint that this represents.

The impact of the Web in particular and information technologies in general on society is great, and research projects are often funded because of their relevance to society. With this perspective, this research is hardly value-free. As individual researchers, we may develop a strong relationship to the systems we create, as we have a great intellectual investment in them. This may influence our ability to reject a flawed hypothesis even if we should. Even though we note in the discussion of the contributions in Section 4.2.5 the weaknesses of Thomas Kuhn’s philosophy of science, his insistence of understanding science as a social endeavour stands firm.

These challenges are inescapable, and therefore, it is important that the researcher acknowledges such problems and explicitly details their own personal motivations, visions and possible biases to let the reader decide whether the researcher’s subjective beliefs or other external pressures has influenced the validity of their contribution. It is also important to be humble and admit that even though we as researchers treasure our objectivity, like Galileo, we may not be in the ideal position to judge our own objectivity, as should be clear from the few challenges that research methods are getting in the literature. We should therefore detail our background to better equip the reader to judge.

I shall attempt to do so in the following, but also note that this influences the writing style. There are several conventions of scientific writing that stands in
opposition to the goal of letting the reader understand the possible influence of subjective beliefs. The frequent use of passive voice is one such convention, that in my opinion tries to hide subjectivity rather than alleviate its harmful effects. If the methodology used in the investigation is flawed, it should not be obscured by writing style, the methodology will need to be enhanced. Double blind trials in medicine are an example of why it is important to acknowledge the investigator’s role. That is not to say that the use of passive voice is always inappropriate, when the role of the investigator cannot be misunderstood, it can be used just like any other language construct.

Of less importance is the use of “we”, when it is clear that the work has been done by a single investigator. I hold as a general principle that the role of the investigator should never be obscured, and therefore, such use is inappropriate. The use of pronouns should not differ substantially in scientific writing from other writing, and I will use them accordingly. In particular, I will use “we” when more than one person has been involved, or when the reader is included (e.g. “We note that...”).

Lastly, while the discussion in Section 4.2.5 should make clear that I think that there are epistemological problems underneath the entire field of study, i.e. there is reason to question the validity of any result, there are invariably certain statements that have a particularly weak foundation, especially in the motivation. I shall be careful to use the term “belief” to qualify such statements.

2.4 A Personal History

As noted in the previous section, objectivity may be challenged by a researcher’s intellectual investment in a certain field of study and I admit that I have a very large intellectual investment in the success of the Semantic Web. This section serves two purposes: It should give the reader a good understanding of this intellectual investment, and enable them to assess the impact of any subjective beliefs. Secondly, it serves to clarify my vision, which we noted in Section 2.2 is more important than to attempt a definition. The vision strongly influences the motivation detailed in Section 2.5.

I created my first Web page in December 1994, and strongly appreciated the value of the fact that I could easily contribute to the Web, and therefore, I quickly posted anything that I had of value, for example a collection of my best mountaineering pictures, constrained not by the platform, but by my own time. Ini-
tially, I learnt purely by example, i.e. using the “View Source” menu item when I encountered web pages that I liked. I also had the power to mint identifiers.

With identifiers, and the hypertext language HTML came the ability to link to anyone, and anyone could link to me was highly empowering: Discovering that people I didn’t know linked to my material was a strong, social reward. After some time, I had tens of thousands of visitors every month.

In 1996, I started the Web pages of the Norwegian Skeptics Society using the domain skepsis.no, and therefore got the control over an entire Web server. That was another revelation, as I could actually run code, and from that arose the need to not just learn by “View Source”, but start to read the specifications, notably HTML, CSS and HTTP. Appreciating the design of these specifications took a while, but eventually, the value of orthogonal specifications and separation of concerns became apparent. Also, at the same time, I realised that I could be standing on the shoulders of giants by using Free Software.

When the RDF working drafts were first published, I was first overwhelmed by the amount of new text to read, but at the same time I had a growing realisation that getting information out was not solving the problem The Norwegian Skeptics Society needed to solve: We needed to push information into closed minds. My initial plan to address that was to enable a higher degree of targeting of information, and a Web-wide conversation. To do that, I wanted to create a large thesaurus of topics of interest, and create annotations of the type “this article is a rebuttal of that article”, and an index that browsers could query, so that when a user viewed an article, they would also get a critical context. At the same time, Netscape was open-sourced as Mozilla, and so, I was quite convinced I could contribute the code needed for this to be accepted.

I shared these ideas on some mailing lists in August 1998, and one of the persons who responded was Dan Brickley, who would chair the RDF working group. He quickly convinced me that RDF was not something to be afraid of, quite the contrary, it was exactly what I needed.

However, all the above had happened on my spare time, as the topic of my study was something else entirely: Cosmology. Therefore, the interest lay dormant for a few years as I finished my Cand. Scient.-degree. After that, I got some limited funding to write what today would be called a Semantic Content Management System. The project was far too ambitious and largely a failure, but it got me some valuable experience with Semantic Web technology.

I was convinced that the things that made the Web great to work with had to be present in the Semantic Web as well: Everybody is empowered to publish, find
and quickly make use of Free Software, learn by “View Source” (an idea which becomes even more powerful with semantics, as if you can understand what you are viewing simply by looking at the message, you can begin processing it). Also, anyone can mint identifiers and anybody can reuse those identifiers in their own contexts, and eventually graduate to reading technical materials as needed. Many of these things are virtues of the Semantic Web as it is an extension of the Web, others have not been sufficiently cultivated or even appreciated. Moreover, my experience corresponds with the virtues that Tim Berners-Lee has attributed the Web’s success.

It was not until I joined Opera Software in 2005 to work on the now defunct Opera Community social networking site, that I had a bit more time to work on Semantic Web technology. When I joined, they had rudimentary support for an RDF/XML-based “Friend of a Friend” (FOAF) export, which I improved before the initial release. At the time, it was not anticipated that everything would have a URI, so Dan Brickley was a proponent of the view that most things, including people, would be identified mainly by their properties, e.g. their email address. It was a certain tension between that view and the “give everything a URI”-movement heralded by Tim Berners-Lee at the time, but I decided to place myself in between, by giving everything a URI, but at the same time ensure that enough properties would identify people. I also included outward links, so that people could integrate the database export we provided with their hand-maintained FOAF. I also added support for photo gallery metadata and linking tags as given in a personal SKOS-ontology to Wordnet to enable assigning clearer meaning to tags. This effort was acknowledged in the Linked Data Design Issue that formed the basis for the more recent Linking Open Data project.

However, at this point, I started to realise that it was difficult to develop applications around traversing Linked Data, and that prompted me to consider the possibility to offer the ability to execute arbitrary queries. On 2005-11-30, I published the first public SPARQL endpoint with actual production data, consisting of 2695114 triples, but it quickly grew to approx. 15 million triples.

Unfortunately, management did not permit more time to enhance and document the SPARQL Endpoint further, and it didn’t see much practical use. I was, however, allowed to join several World Wide Web Consortium groups. Firstly, the Web Content Label Incubator Group, which was given the task of discussing


\[3\] See [http://www.onlamp.com/pub/wlg/8609](http://www.onlamp.com/pub/wlg/8609)
a replacement for the then archaic Platform for Internet Content Selection, and resolved to use RDF for this purpose. It transitioned into the POWDER Working Group, which provided the specifications.

Secondly and more importantly, I joined the Semantic Web Education and Outreach (SWEO) Interest Group, where I admitted my frustration with the lack of practical progress with Semantic Web technologies and the relative lack of uptake. This sentiment was echoed by the late Aaron Swartz, who when asked by the group on his opinion, emailed the following statement:

I’m not sure what SWEO is, but my feeling is and pretty much always has been that the Semantic Web people need to start putting together Genuinely Useful stuff that can be done Right Now.  

This was one of my motivations for starting the SWEO Community Projects, where a questionnaire was posted on the Web, and people from the Semantic Web community were challenged to come up with a project that would provide practical benefits in the short term. We received 10 proposals, out of which 3 were selected for backing by the SWEO IG. None of them were successful, but a fourth proposal, submitted by Chris Bizer and Richard Cyganiak built momentum quickly and was therefore also selected for backing, despite some criticism by several IG members, myself included. The proposal was titled “Linking Open Data” and sought to take already abundant open data and model it using guidelines of the Linked Data Design Issue [11]. LODstats [6] provides extensive statistics on the results of this project, and has by the time of this writing seen nearly 10 000 data sets.

At this point, I left Opera to work as a consultant on Semantic Web technologies. One of the projects that were successfully completed were called Sublima [23]. It was driven entirely by RDF and SPARQL, and featured faceted navigation, navigation in a thesaurus, and full text search. Experiences from this project influenced requirements of SPARQL 1.1 [29], and I was an editor of the SPARQL 1.1 Features and Rationale specification [38]. Parts of the property paths feature and aggregate queries were used extensively. Write operations were also implemented with SPARQL using what was then specified only in a member submission to the W3C [56]. This was the basis for the update language in SPARQL 1.1 but differed substantially in surface syntax, partly because our work showed how certain usage patterns could be simplified.

4See https://lists.w3.org/Archives/Public/public-sweo-ig/2006Dec/0138.html
Two features that arose from Sublima requirements were not accepted for SPARQL 1.1: Full text index and a feature known as “Limit Per Resource.” The latter feature arises from the fact that usually, the user is indifferent to the number of solutions to a query, and therefore limiting by the number of solutions is unhelpful. This is especially true since other features of SPARQL, such as optional clauses, are helpful in dealing with heterogeneous data. In the Sublima case, it was interesting to limit by the number of articles that were returned, but the number of solutions depended on the number of authors of an article and the concepts used to classify an article. To achieve the desired effect with SPARQL 1.1, one would have to write a very complex subquery, which is undesirable for such an important feature.

Another project, based on much of the same code base replaced the SKOS ontology of the Sublima project with an OWL ontology, and also used reasoning to aid navigation in multimedia libraries.

However, it became clear that I would not have sufficient time in the industry to pursue what was developing as my main interest, query answering over data on the open Web for generic application development.

## 2.5 Motivation

The following section provides an overview of the motivation behind this work in general and each direction in particular. Detailed problem statements are given in Section 4.1.

### 2.5.1 Cruise Altitude View

There is a class of diverse, yet structurally similar, problems people face nearly every day: When we book a flight, we may be interested in a romantic getaway, yet we are forced to select the airport of destination first. We often end up buying the same groceries because we don’t know what will taste well together. Last time I bought a car, the two most important features was that it had 4 wheel drive and that the trunk was long enough to fit a full-size baby stroller length-wise. To find a property to build a home, I took into account the sun conditions, that it was close to public transport that would transport me to campus within a certain time frame, close to ski tracks, low concentration of uranium in the ground, number of

---

\(^5\)See \url{http://www.w3.org/2009/sparql/wiki/Feature:LimitPerResource}
children for my children to play with, and an affordable price level. And once the house was being built, I had to choose the toilet seat depending on the frame in the wall.

Of fundamental importance is that even though each of these problems are rarely encountered, there are so many such problems that we encounter them frequently. I would like to see a Semantic Web where the plumber delivers his offer in a manner so that the open options can be distinguished from the constraints, where a query can be formulated to find the area I’d like to build my house based on such extensive criteria, where the very rare comparison between a collapsed baby stroller and car trunk can be done, where I will be able to cook better food without being a chef and where I can book my travel based on what I want to experience rather than make me choose a destination based on limited information.

Now, all this information exists on the Web, or at the very least, in some database, and for the most part, the above problems can be solved by surfing many websites. We are, however, reminded of “Connolly’s Lament” (due to Dan Connolly, one of the Semantic Web’s early champions):

The bane of my existence is doing things I know the computer could do for me.

Moreover, the problem could be solved by writing programs to integrate the variety of data sources for each scenario, but this is clearly not sustainable, as each particular integration is of interest to very few, but the cost of the integration increases for every integration task. Fewer people to pay for a more expensive task quickly becomes economically infeasible.

Technically, it might be an easier problem to solve if the data could be centralized, and that solution has been argued for by [15]. Nevertheless, even if all their arguments remain valid from a technical viewpoint, the most important arguments against centralisation are social: If important components of the Semantic Web are under the control of some central authority, it is likely that one cannot publish anything, link to anything, or develop any application without permission. It may create an unhealthy power structure, which may create commercial or intellectual hegemonies. Tim Berners-Lee discussed this in [12] and an interview in Wired Magazine in 2014[7]. This is the key motivation to still design a decentralised Semantic Web, regardless of the technical difficulties that may arise.

---

6See http://www.nature.com/nature/webmatters/xml/xml.html
7See http://www.wired.co.uk/news/archive/2014-02/06/tim-berners-lee-reclaim-the-web
The Crosscloud project is motivated from the same observation:

Today, in a world of cloud-hosted software, every application is a kind of trap. Even if it lets you export your data in a form other systems can read, many of the best apps have social features. You can’t switch because your friends or colleagues are still using the old site — the new site will be a ghost town. This stifes competition, blocks innovation, and leaves users less happy with the systems they are using. While some developers might want to lock-in users, we trust that many value user happiness and would open their systems if it was technologically practical. Our goal is to make it practical.

The naive enthusiast may now forward the claim that Linked Data is the solution and the whole solution, and at times of hubris, I may myself have been guilty of conjecturing that sitting down to write the code based on what we already know would solve the problem.

At times of doubt, the large number of unsolved problems, theoretical as well as practical become apparent. In the following, I will try to enumerate the problems I have foreseen, discuss their place in current research, and what problems I have attempted to tackle in this dissertation.

Both the idea of a decentralised Web and the idea that SPARQL endpoints can be made openly available for any query on the Internet seems to go against the conventional wisdom of the database community. We are, however, motivated to solve this problem by the social requirement of decentralisation and by the use cases mentioned at the start of this section.

It is also important to frame the question correctly: We want to answer queries, we do not require that any remote server accepts the entire burden of evaluating queries. It is becoming quite clear that the current practice of evaluating any arbitrary SPARQL query on a remote server is economically unsustainable, and it is made clear by the instability of SPARQL endpoints on the Web [16].

2.5.2 Hypermedia

The hypermedia idea is attractive as it is the extension of the “View Source” idea that brought me to the Web. In its original form, I believe that this feature is essential for broad developer adoption. With hypermedia, a developer would

---


not need to read extensive documentation, but can start using the data to create applications that interact with the data with a very low barrier to entry. When designing the hypermedia discussed in Section 4.2.1, this was my primary design motivation.

It is also possible to think of hypermedia in terms of machine interaction. A classical SQL database is often the antithesis of hypermedia, as the semantics of columns are usually not clear. To interpret the schema, domain expertise is needed, the documentation may reside in documents not even accessible to machines, for various reasons.

RDF is well suited to include information that clarifies the semantics with the data so it could be argued that most RDF documents would be hypermedia, but what is often overlooked is that for RDF documents to be self-descriptive, they need to explicitly link their relevant schema. Moreover, control information to enable anticipated and serendipitous applications must be made available in the RDF. For example, it is not sufficient for a pizza restaurant to publish data on pizzas and link to the schema, the RDF must also express how to order to enable a software agent to organise a party.

2.5.3 Developer Friendliness

As noted, my hypermedia motivation came from the perspective as a developer, not from the research literature. Nevertheless, hypermedia has a substantial research literature behind it. While software engineering certainly is a large field, the direction of work concerning developer friendliness comes mainly from personal experience, from developing solutions in the industry, but also from working with other developers who are at best indifferent to Semantic Web technologies, and in many cases antagonistic towards it. As noted in Section 2.4, Aaron Swartz echoed the sentiment already a decade ago that the Semantic Web had failed because it did not provide tangible benefits, and had gotten a reputation for being unfriendly to developers. It is crucial to address this problem.

2.5.4 Philosophy of Science and Statistical Methods

While a failure to be developer friendly may be one reason why the Semantic Web has not succeeded on a grand scale, my training in natural science lead me to ask if there may be other reasons that we do not see the remarkable rate of progress
of the natural sciences in this field. These questions are as hard to ask and to live with as to answer.

I became quite confident at an early stage that the practice of benchmarking was an easy target, with its lack of summary statistics and inability to support proper hypothesis testing, standardising to deal with complexity rather than methodological advances, and lacking in ability to formulate the experiment as severe enough.

2.5.5 Caching in the Internet

Based on the this discussion, federated SPARQL could be forwarded as a solution to the problem of query answering over decentralised sources, and that was originally where I intended my main contributions. During the work, it became clear that the stability of individual endpoints would need to addressed first. This will be further discussed in the detailed problem descriptions, as the choice of caching as a point of focus is somewhat arbitrary.

An important motivation was not problems, however, but opportunity: The proliferation of Content Delivery Networks was an opportunity for improvement, the emergence of Triple Pattern Fragments, and traits based planning in Section 4.2.3 all represented important opportunities to seize.

We therefore proceed with the understanding that the Internet infrastructure is accommodating for solving parts of this problem with the presence of caches at several different levels. We would need to create an infrastructure to take advantage of these caches, and encourage data publishers to include metadata to allow caches to be managed in a standard-defined way. The scale at which this can already be done was at first important to understand.

We further note that some clients also have significant processing capacity and could process SPARQL queries themselves. Others, like microcontrollers on the Internet of Things, cannot. This calls for a diverse infrastructure that can accommodate all kinds of clients. Nevertheless, these are among the factors that could make SPARQL query answering on the open Web practically feasible.

The convergence of all the directions of this work was intended to be a contribution to ease the burden of servers by allowing other parties to perform parts of the query evaluation.
Chapter 3

Preliminaries and Related Work

3.1 Preliminaries

This section extends the basic introduction in Section 1.1 where required to understand the following discussion.

3.1.1 Caching in the Hypertext Transfer Protocol Standard

Internet standards come in the shape of Requests for Comments (RFC), that it is hoped will gain traction to become de facto standards. Two such RFCs regulate caching in the Internet when HTTP is used: RFC7234 “Caching” [25] and RFC7232 “Conditional Requests” [26]. In [17], I provided some details on these standards in Section 1.1, that is worth reiterating to understand the problems and contributions of this work.

RFC7234 defines how clients and proxies may reuse a prior response without contacting the origin server at all. The server can do this by supplying a freshness lifetime. It also defines when clients and/or proxies may not reuse a prior response, and what can be done with stale responses. It also contains a Section 4.2.2 titled “Calculating Heuristic Freshness”, which sets some loose constraints for when a prior response may be used even when the server has not supplied a freshness lifetime. These constraints include that heuristics may not be used if there is an explicit freshness lifetime, or the server prohibits caching. The section also proposes a simple heuristic for calculating freshness lifetime.

RFC7232 defines a protocol for asking the origin server if the cached response is still fresh. Two response headers are relevant to achieve this: ETag and Last-Modified. ETag may contain an opaque identifier for a specific version
of a resource representation, for example a query result. A client or proxy may then ask the origin server if the resource is still fresh, by sending a request with a If-None-Match header. If the server is able to validate the resource representation, it will respond with a 304 status code, otherwise it will return an updated representation. It may be of use if servers are able to generate and validate such opaque identifiers cheaply.

The Last-Modified header should contain the time of last modification of the resource representation. A similar protocol is used to validate it.

### 3.1.2 Topics in query evaluation

As shown in [51] that SPARQL has some highly complex parts. In particular, nested OPTIONALs will be difficult to evaluate. It is possible for a user to either inadvertently or with malicious intent (in e.g. an denial of service attack) submit a query that will cause excessive load to the server.

Another problem worthy of special attention is the case where parts of the query has no connecting variable. Let us consider the simple example of when two triple patterns do not share a variable:

**Example 5. BGP with Cartesian join**

```sparql
?person foaf:name ?name .
```

When this is evaluated, it will cause what is known as a “Cartesian join”, in which the result will contain *all* the solutions of the first triple pattern joined with *all* the solutions of the second triple pattern. This will usually be a large result, and it is also resource consuming to compute.

In some cases, the users may have written such a query, in which case the query planner only has the options to execute the query or reject it entirely. In other cases, the query planner may arrive at such queries as part of query rewriting, in which case it must ensure that such plans are not preferred.

### 3.2 Related Work

There is a substantial amount of relevant literature and ongoing research that are well aligned with the present work, either because the authors share parts of the motivation, or this work finds serendipitous reuse of prior work.
The related work on the topic of evaluations based on contemporary statistics, and on the topic of philosophy is scarce, and has been dealt with in the papers. For the other topics dealt with in this thesis, a more extensive treatment is needed to place this work in context:

### 3.2.1 Hypermedia

Hypermedia, in the context of the Web, has been strongly influenced by Fielding’s definition of the REST architectural style, see [27] Chapter 5, in particular, a constraint that he formulated, known as “Hypermedia as the Engine of Application State” (abbreviated HATEOAS). The implications for an application using RDF is that the application should be able to tell from the RDF data only how any interactions should occur. The semantics expressed with RDF may enable this.

An example of this use is the work considered in Section 4.2.1, another example is the new direction proposed by Verborgh et al. [61] who wanted a system that would be able to answer individual triple patterns to enable clients to answer queries, and in doing so, they employed hypermedia, and formulated this as Triple Pattern Fragments (TPF). The key to this hypermedia is that an answer does not only include the data, but also metadata, importantly a cardinality estimate for the triple pattern, and also control information, that tells a client explicitly in every response how further interaction should be done, like in obtaining further data.

Overall, the work on query answering with Triple Pattern Fragments share many of the observations and goals that motivate this study. Their stated goal is to transfer as much as possible of the burden of evaluating a query from the server to the client. This can happen because a simple pattern match for a single triple pattern need not have a SPARQL parser, nor a query planner. Moreover, the server may materialise responses to frequent triple patterns and store them in a file system, which can be managed by a simple Web server. The results may also be paginated, so that each individual response may be kept small.

However, it is not necessarily the case that evaluating the entire query on the client side is the most efficient, depending in particular on the perspective of the different actors. Chapter 5 adopts Triple Pattern Fragments, but makes different assumptions, for example by trying to avoid a large number of HTTP requests.

Also, TPF mandates that every response must contain a cardinality estimate for every triple pattern, an operation that may be quite expensive for the server. Indeed, this information may be so expensive to provide that SPARQL planners may not do so, in order to keep the cost of planning itself down.
In [60], they demonstrated SPARQL evaluation over TPF, and TPF has become an active area of research. The impact of caching is also measured in this work, as the result of a single triple pattern is far easier to cache using HTTP mechanisms than the result of an entire SPARQL query.

Under the auspices of the World Wide Web Consortium a standard for read-write applications using RDF has been developed: The Linked Data Platform [57]. The abstract in the specification describes it as:

Linked Data Platform (LDP) defines a set of rules for HTTP operations on web resources, some based on RDF, to provide an architecture for read-write Linked Data on the web.

While developers who are familiar with HTTP-based applications and Linked Data may feel familiar with the platform, it still requires that they read and understand the specification to be able to use it, and therefore, it is not hypermedia, and I think it misses out on many of the benefits of hypermedia. A discussion at the Developers Workshop at the Extended Semantic Web Conference 2015 found rough consensus around a comment formulated by me:

LDP has to be superseded by hypermedia in some way.

### 3.2.2 Query Answering With Cache

First, we note a claim that SPARQL query caching is not possible: In [33], the authors examine cacheability as one of the desiderata for sustainable data access. They claim, without further justification, that SPARQL isn’t cacheable. With this work and much of the below related work, I suggest they are mistaken. Not only can whole SPARQL queries be cached, but it is possible to break down the query to parts that can be cached.

As such, this mirrors an extensive development known as “Semantic Caching” in the database community. Ahmad et al. [3] surveyed the state of the art in that area. They divided caching approaches into page, tuple and semantic caches, and noted that in the two former cases, if only partial results were available in the cache, the entire result would have to be fetched from the remote database. Indeed, a page or tuple cache may be used to cache the results of a whole SPARQL query in HTTP context, but SPARQL queries are also trivial to partition by e.g. triple

---

1See [https://www.youtube.com/watch?v=F4WN4XEpV1A](https://www.youtube.com/watch?v=F4WN4XEpV1A) for a recording of the discussion session.
patterns or Basic Graph Patterns, which provides further opportunities for caching where only partial results are available.

Two papers that span nearly two decades of research are of particular interest: The HERMES system [2] considered query processing and cost-model based optimisation in a mediator system where the mediator does not have access to source statistics information. Recently, Papailiou et al. [50] made an extensive study of SPARQL caching, addressing some of the same problems as the present work. Even though these do not share our perspective, which is deployment on proxies in the Internet and integration with HTTP-based caching, these contributions provide important context.

HERMES [2] provided motivation for the paper in Section 4.2.3, but is also important for the work in Chapter 5. An important concept in the HERMES system is “invariants”. They are, in our terminology, patterns that do not need to be looked up remotely, but can be integrated in the query answer by the mediator. The expectation in HERMES is that such invariants will be encoded by a domain expert. The HERMES planner bases decision on a cost vector with estimates for time required to find the first answer, time to find all answers and the cardinality of the answer set. Caching summary statistics is another important feature of HERMES, and it also contributes lossy and lossless summaries, for the purpose of saving space and reduce the time it takes to compute a cost estimate. HERMES further has a query rewriter and optimiser, that rewrites the query taking into account the cache and the invariants. While statistics summarisation is relevant and should be studied in future work, we note that with a Linked Data Fragments server, a cardinality estimate is required to be available for every triple pattern, and is also cacheable with the same constraints as the data themselves.

Interestingly, [50] opens by noting that many optimisations used by SQL databases are ineffective in RDF databases, due to that RDF schema, even if they exist, are not as helpful as the obligatory schema in SQL databases. This provides an opportunity for novel research. The solution proposed in [50] is not a cache on a proxy or a mediator like that which is my primary focus. The execution engine in this case, has access to both the primary indices of the RDF store and the cache. A key contribution of the paper is a canonical labelling algorithm that can create a string that will be unique for a Basic Graph Pattern including optional blocks. This is done by transforming a SPARQL query to a directed vertex-coloured graph, where each triple pattern is represented by a vertex, and triple patterns that share a common variable is linked with an edge. For each vertex, a label that consists of three IDs are created. Bound terms are translated to
IDs using a dictionary, while variables are translated to zero. Then, the graph is directed and edges labelled according to the type of join they represent. The vertex and edge labels are then translated to non-negative integers (colours) using a sort key, to form a directed vertex-and-edge-coloured graph that according to the authors is isomorphic with the original SPARQL. Finally, the graph is transformed to a simpler directed vertex-coloured graph. With this graph, the author’s method can produce a canonical label for the query graph.

The query planner first examines all connected subgraphs of a full SPARQL query, and generates canonical labels for each one. To further enhance the query planner, the cache can optionally include indexing. The authors also introduce the concept of profitable query patterns. It is not quite clear from the text, but it seems this implies that the cache can also prefetch results into the cache, if it finds that the result may benefit subsequent queries.

Olaf Hartig has a large number of papers that are adjacent to the present work, including his Ph.D. dissertation [31] where he explores querying Linked Data, i.e. query data that is not contained in an a priori defined collection of RDF data, but where resources are traversed to compute a result. Hartig provides solid formal foundations for such query processing, as well as computational feasibility, soundness and completeness. The most relevant paper to this work is [30]. In that work, he considers the impact of caching on performance and completeness. Completeness is important in the context of query answering when the collection is not defined, and the cache may therefore provide results that were not accessible for some reason, at query time. Hartig finds many cases where it is profitable to cache, but also cases where it is not.

Lampo et al. [42] found similar results. They found that queries with star-shaped groups, that is, triple patterns that share a common variable in the subject position, take best advantage of a cache, and suggest rewriting queries to take better advantage of a cache.

Acosta et al. has published a poster on “SHEPHERD” [1], presumed to be ongoing work, to create a SPARQL processor that takes into account observations done by the SPARQLES survey [16], to decompose a SPARQL query into subqueries that are indicated to be easier for the server to evaluate, and so is well aligned with our goal to ease the load of the server. It is not quite clear from the brief description in [1] whether the SHEPHERD system can cache results locally.

The following studies do not consider how the results of one query can benefit the execution of a subsequent query. Nevertheless, they are important to consider as prior work: Umbrich et al. [59] considered a situation where parts of the graph...
change at different paces. For the slow-changing parts, they prefetch an entire
dataset to a local store to relieve remote endpoints from some processing, and
thereby speed up query responses. Of central importance to this approach is the
notion of coherence of query patterns, defined through the ratio between results
that were found in the live remote endpoint but not in the local cache and the
results from the live remote endpoint. Based on this, the need for correct, up
to date answers can be balanced against the need for good response times. This
approach differs radically from our approach in Chapter 5 in that it replicates an
entire dataset rather than just fragments.

Naturally, the problem of caching in a proxy has much in common with the
federation problem as the cache, as seen from the query execution engine’s per-
spective, could be viewed as yet another possible data source, and the number of
HTTP requests to the remote endpoint can be wisely assumed to be kept small.
Schwarte et al. [55] addressed these problems with FedEx, in which they sought
to group connected subgraphs with their concept of exclusive groups. Montoya
et al. [47] propose a query federation system that minimise the data needed to
be transferred while preserving completeness in the common case where the data
relevant to a query is available on different endpoints. Görlitz and Staab [28]
exploited cardinality estimates sometimes available in VoID [18] descriptions to
improve join order execution plans. It could be advantageous for the system in
Chapter 5 to exploit as well, and straightforward to implement, but since it has not
seen wide uptake and the statistics are mandated by Triple Pattern Fragments, and
therefore available in some cases to the planner, we chose not to.

Martin et al. [45] implemented a reverse caching proxy that controlled the
changes to the dataset rather than rely on HTTP headers to track changes, and so
could relieve the endpoint of the burden of evaluating all queries. Such a proxy
would work poorly in the scenario considered in Chapter 5 as there is no way to
mandate use of the proxy in this scenario.

Williams and Weaver [63] consider the implications of the HTTP protocol
on query caching in a quad store on the indexing methods that were employed.
They propose to introduce a modification timestamp in index structures, and find
that the query engine would be able to identify pieces that have been changed
during execution, and could therefore expose a Last-Modified header and validate
an earlier response in accordance with RFC7232 [26] in a cheaper way than by
evaluating the query. This also represent an alternative to HTTP ETag as in the
discussion of [50].

Dividino et al. [20] proposed strategies to keep caches of Linked Open Data
up to date, in face of changing data. The authors set out to learn the change frequency of resources representations, and based on that schedule updates of them as efficiently as possible under bandwidth constraints. Instead of relying on resource metadata, the authors propose an algorithm that iteratively ensures the quality of the cache by examining the resource representations that have been downloaded and compute certain metrics, such a dynamicity and changerate.

While the approach is interesting and useful, I nevertheless have some points of criticism: They motivate their approach from an earlier finding that only 8% of Last-Modified headers they surveyed reflected the true change frequency. While I have no reason to doubt this finding, the authors boldly go on to claim that “The only alternative [to Last-Modified] is to actually retrieve the data from the sources and check it for changes,”, but this contradicts the results I found in [37]. The authors have not considered all alternatives, including not the alternative presented by the HTTP caching standard RFC7234 discussed in Section 3.1.1 which invalidates that assertion. Moreover, the author’s approach would violate RFC7234 in the case where their algorithm caches results that should not be cached or has an explicit freshness lifetime, as discussed in Section 3.1.1. As a general observation, it is my belief that the Semantic Web community should be more cautious about violating core Web standards.

While the approaches are diverse, these studies pull in the same direction: Motivated by the problems of maintaining SPARQL endpoints, they all seek to share the burden, and so make query answering economically feasible.
Chapter 4

Problems and Solutions

4.1 Problems

We seek to formulate concrete problems that should be solved to realise the vision formulated in Section 2.5. It is important to note that while some of these issues may be problematic for the whole Semantic Web community, their only intended scope is the presented vision. Moreover, many of them are not addressed in this thesis, but discussed for comprehensiveness.

4.1.1 Performance and Stability Problems

As discussed in Section 3.1.2,

**Problem 1.** Some features of SPARQL may cause excessive endpoint load.

We also note from experience that:

**Problem 2.** Public endpoints may have unpredictable usage patterns.

Moreover, an ambition of the Semantic Web is to deal well with heterogeneous data and it has been known for long [19] that power laws are frequently encountered when fitting a distribution to data. As noted in the discussion of Example 4 in Section 1.1.2, a query planner would often rely on statistics, but:

**Problem 3.** The data distribution is often unknown to the query engine.

Only simple summary statistics may be available. The discussion of Example 4 illustrates that even if the distribution of data given for example a predicate is known,
Problem 4. Conditional data distributions are usually unknown to the query engine.

For example, the population of geographical places may be uniformly distributed in a given data set, but the population given that the places are of type city may be a power law. This could have very significant impact on the overall performance, since in the latter case, the query planner may consider a BGP like:

Example 6. Skew distribution

?place ex:population ?population ;
   a ex:City .

as a unit.

Even if this statistics could be found, it is not clear that it should:

Problem 5. Comprehensive statistics may be costly to compute and requires much storage space.

Therefore, it is important to find a balance.

Heterogeneity should also be expected in different types of servers and clients. In particular, microcontrollers and to a lesser extent, mobile phones, are resource constrained devices. Thus:

Problem 6. Clients run on devices with a wide range of processing capability.

Caching plays an important role in the Web architecture [34], because previous responses may be reused to satisfy a current request. However, there are many problems that need to be solved to use it for SPARQL query evaluation:

Problem 7. SPARQL endpoints, which are based on HTTP, will only enable caching syntactically identical queries by default.

The performance issue discussed in Section 3.1.2 has relevance in the context of caching:

Problem 8. Caching of results of individual triple patterns may cause subsequent queries to break into blocks that must be evaluated with Cartesian joins.
To understand this, consider Example 5 and extend the BGP to a query with 3 triple patterns. In addition to the two triple patterns in Example 5, add a triple pattern that connect the two, for example ?person foaf:made ?paper . If the result of this triple pattern is cached, what is left to evaluate are the two triple patterns in Example 5 which is likely a very expensive operation. Another approach to this problem is that of [50], as they insist on caching only connected subgraphs.

It would be useful when caching if one could compare the incoming query to a query with an already cached result and determine that they were equivalent, but unfortunately:

**Problem 9.** Determining that two queries will return the exact same answer is a complex problem.

Metadata is important for query planning, and as explained in Section 3.1.1, the HTTP protocol provides facilities for sharing such information. However:

**Problem 10.** It is not known how much of the Semantic Web exposes information that could be used in HTTP caching.

Even though statistics may be available internally in a query engine, a caching proxy is unlikely to have direct access to that data, and therefore:

**Problem 11.** A caching proxy far removed from the database it caches may have little or no statistics for query optimisation.

### 4.1.2 Problems of Shared Understanding

In [12], Tim Berners-Lee et al take schema heterogeneity for granted and postulate that it would be resolved by using inference, saying:

> When, eventually, thousands of forms are linked together through the field for “family name” or “last name” or “surname,” then anyone analyzing the Web would realize that there is an important common concept here.

He goes on to explain how linking such terms will help make that inference, and how partial understanding is common in everyday life and will also be sufficient in my cases on the Web. The practice of using inference to find a common understanding is now known as ontology matching or ontology alignment, and is
a very active area of research. This problem, as well as related problems, such as entity resolution, are considered orthogonal to the problems in this thesis, so the below problems are issues that arise from the ideal of an open decentralised Web. These problems are therefore not discussed further.

**Problem 12.** *How can suitable inference engines be made available to all users?*

However, we shall note that there are two directions that do not require inference to the same extent:

Schema.org is sponsored by Google, Microsoft, Yahoo and Yandex using a community process to create and maintain a single, coordinated ontology. With their market shares, they have a reasonable chance of getting wide-spread adoption. However, we see also the key problem:

**Problem 13.** *With a centralised ontology approach, how can we ensure that anybody is enabled to say anything?*

The next possibility is that authors converge towards using a single vocabulary per domain, but with no central control. The Linked Open Vocabularies is a useful resource for work in this direction, see [7]. This may occur for economical reasons: If this happens to be the least expensive way to achieve useful data integration, authors may realise that this is a reasonable action to take. While this does not eliminate the need for inference, it may greatly reduce it.

**Problem 14.** *How can the adoption of existing vocabularies be encouraged?*

### 4.1.3 Problems With Evaluations

To understand whether a result from a study on for example, performance is valid, an empirical study must be conducted, as a SPARQL system is generally too complex to be evaluated solely by formal methods. Conventionally, one or more benchmarks have been used for this purpose, and there exists a number of such benchmarks. Unfortunately, due to the complexity of the systems under evaluation, benchmarks have to test a very large number of parameters, which is methodologically beyond their reach. They may also oversimplify the test by attempting to neutralise certain optimisation techniques. Further, they have no structured approach to investigate flaws in the benchmark itself, and have no meaningful summary statistics that can be used to understand the overall performance. While one may standardise benchmarks, doing so makes it impossible to test assertions that are outside of that standard.
Problem 15. Evaluation methodologies are not founded on sound statistics.

It is clear that neither the vision presented in Section 2.5 nor the more advanced vision presented in [13] exists as of today. In both cases, Semantic Web would have to be deployed at a very different scale than what it is today. If the Semantic Web is sufficiently successful to realise these visions, it is also likely that it will be very different from today in terms of data profiles, workloads, etc. Since data profiles and workloads are important to devise a test to evaluate assertions about performance, the fundamental problem arises that:

Problem 16. We currently test against something that does not exist and therefore usage patterns cannot be predicted,

and this differs markedly from the methodology of natural science.

Frank van Harmelen has encouraged a direction that seeks to study laws of the Semantic Web information universe\(^\text{1}\). If this direction is pursued with success, then it may be possible to establish laws that can be used to generate data and workloads that can support realistic experiments even though a future Semantic Web may be several orders of magnitude larger than that of today.

Problem 17. No methodology has been established to find laws of the Semantic Web information universe that can reasonably be expected to have validity far beyond the present.

Even if laws have been found that can make extrapolations across orders of magnitude tenable, evaluations still pose significant epistemological problems. For example, falsificationism, as argued by Karl Popper and adopted in [14] will be unsuitable, for reasons that more recent philosophers have pointed out, see [17] for a comprehensive discussion: If it is insisted upon that hypotheses must be falsifiable, how can a researcher know if their hypothesis should be rejected, or if it is something wrong with the law, or perhaps the experiment itself?

Problem 18. Evaluation methodologies have little foundation in philosophy of science.

\(^{1}\)See e.g. http://videolectures.net/iswc2011_van_harmelen universal/
4.1.4 Development Problems

There are several issues that make Semantic Web development difficult, that are connected with development in current programming languages and current Semantic Web libraries.

This is very important, because even though we have argued that the programmer needs to be eliminated from the data integration task, there are several functions where they need to be involved, including end-user application development, and in the short term, certainly also the data integration task.

Following the discussion in Section 2.4, the availability of tools and examples so that the vast majority of active Web developers can quickly publish and make use of RDF is a key to the success of the Semantic Web.

As pointed out in [9], there are few things that are simpler in contemporary programming than to parse a string containing a tree-formatted data structure into a tree that can be accessed directly. However, RDF makes the assumption that the natural form of data is not a tree, but a graph, and in the general case, it must be addressed as such.

Problem 19. Addressing graph data in a programming language.

The initial learning by example by “View Source” of HTML markup can be generalised to RDF by considering the term hypermedia. The most important aspect of hypermedia is, as argued in [27], that hypermedia can be used to drive the interaction of applications, and that all the information needed to do so is in the messages that are passed between the server and the client, no information beyond that is needed. As such, the importance of “View Source” extends well beyond the original motivation of learning by example. More concretely, hypermedia can be used to describe simple query interfaces, define how read-write operations are to be made, etc.

Problem 20. Since the SPARQL language is described in an external specification, it cannot be hypermedia.

It is likely that the adoption of the open Semantic Web would be accelerated if large numbers of developers felt the technology was more accessible to them. There are a number of broad as well as detailed problems that must be solved to achieve this:

Problem 21. Developers who are not well versed in RDF need a readily understandable format that details how they can interact with an RDF server that offers a read-write interface.
Until recently, a query engine would need to break down a SPARQL query to individual triple pattern matches, see Section 1.1.2 for how frameworks tend to interact with data. Typically, the query engine would then use for example the `listStatements()` method of Jena, `filter()` method of Sesame or `get_statements` of `RDF::Trine`, so that only individual triple patterns could be evaluated against the underlying triplestore. The latter also had a `get_pattern` method that could be implemented if the underlying store had a way to optimise Basic Graph Patterns.

Sesame, on the other hand, allow implementations to get a query representation, which it must then evaluate. Nevertheless, this resulted in many problems:

**Problem 22.** By breaking down the query down to individual triple patterns, the query engine cannot take advantage of optimisations that involve multiple triple patterns or other parts of the query.

For example, an underlying SQL store could not compile one query for a BGP, and could not use a `WHERE` clause to evaluate a SPARQL `FILTER`. To remedy this, one could have a method for evaluating BGPs with (such as the `get_pattern` method of `RDF::Trine`) and without filters, but:

**Problem 23.** If methods such as `get_pattern` are added for every part of the query, it would increase the complexity of the API dramatically.

**Problem 24.** By merely passing the entire query representation to an implementation, the burden of evaluating the entire query is also transferred, which makes it harder to use default implementations for most of the query.

The same kind of problem occurs in many operations, such as parsing and serialisation, where an underlying implementation might have relevant information to enhance, for example, the performance of an operation, but where the API does not allow it to expose it so that upper layers can take advantage of it.

Another type of problem occurs when working with dynamic RDF that may mix terminological and assertional information with object-oriented programming. LITEQ [43] has a modern approach to a static case, where the developer gets substantial support from their Integrated Development Environment when programming, and where static typing is seen as a virtue that helps quality assurance of the code.

However, a more interesting case is where an application (in a broad sense) can adapt to the data it is seeing. For example, say that there is an implementation
of a Boat and a Car. It may not have been anticipated when the application was first developed, but the application sees an AmphibiousVehicle. Since it already knows how to propel a Car and Boat, it should be able to adapt to the situation given the context of whether the vehicle is on the road or in water.

**Problem 25.** *Current programming paradigms deal poorly with dynamic RDF data containing a mix of terminological and assertional information.*

### 4.2 The Papers in Context

This section will put the included papers into context, and enumerates the problem (from Section 4.1) each paper is intended to address. It also enjoys the benefit of hindsight, and will therefore address some of the shortcomings that have surfaced since their publication.

#### 4.2.1 Read-write Hypermedia

This work is a paper entitled “The necessity of hypermedia RDF and an approach to achieve it” that was presented in the First Linked APIs workshop at the Ninth Extended Semantic Web Conference. I was the sole author. The citation is:


This paper is relevant to the topic of the dissertation for two reasons: First, it shows how hypermedia may be used to drive a read-write application, not just read-only, which is the focus of most of the discussion. Second, it provides some of the basis for [61], which is important in the query planner.

The main idea in this paper is to address Problem 21 by looking upon RDF as a natural language sentence with subject, predicate and object, and that sentence should then be sufficiently human-readable to be straightforward to understand when a developer does “View Source”.

Its main weakness is how it deals with authentication: The basic idea is that when a client visits a resource, it will be told what write operations it can perform. However, write operations will rarely be permitted unless the client has authenticated and been authorised to do so, and so, it assumes that credentials will be supplied with the first read request. While the HTTP standard permits clients to
supply credentials with such requests, in practice, it rarely if ever happens, possibly for privacy reasons. Thus, the server will have to challenge the client first, but this should not happen on a resource that the client is permitted to read without authentication. Therefore, the proposed hypermedia needs a property so that the client understands what to do if it wants to do write operations.

### 4.2.2 Design of Experiments

This work resulted in a paper that was nominated for Best Paper on the Evaluation track on the International Semantic Web Conference 2013, entitled “Introducing Statistical Design of Experiments to SPARQL Endpoint Evaluation”, which was co-authored with John S. Tyssedal. The citation is:


Problem [15] motivates this paper. To understand the context of this study, I noted none of the benchmarks that were in use for the evaluation of SPARQL engines could be used for basic statistical hypothesis testing, and clearly, hypothesis testing must be a fundamental requirement. Then, I noted substantial criticism forwarded by [21] and [48].

However, this study merely scratches the surface. To establish a Design of Experiment-based methodology would be a multi-year project in its own right, clearly beyond the scope of this dissertation. Unfortunately, this also implies that this dissertation does not feature an evaluation that lives up to the rigorous standards I am advocating as it is unattainable within the time frame of this work.

Originally, this paper was intended as the start of a methodology to evaluate a federated system that addressed the problems [3, 4] and [5] but as [16] confirmed my suspicions that there were severe stability problems with SPARQL endpoints, my attention shifted towards the more immediate Problem [7].
4.2.3 Pushing Complexity Down the Stack

This work resulted in a workshop paper entitled “Pushing complexity down the stack”, presented in the Developers Workshop at the International Semantic Web Conference, by the lead author Gregory Todd Williams. The citation is:


This work was the outcome of several hackathons that I organised in the Perl community, where problems [22] [23] and [24] were discussed at length after having become a major practical issue. It is not very well known that Gregory Todd Williams’ SPARQL implementation written in Perl is one of the SPARQL 1.1 reference implementations. It is also fully compliant with the specification. Through our work, and the availability of traits-based programming in Perl, see [54], we had extensive discussions on how to leave optimisations to lower levels, e.g. the underlying database engine, when they had the potential to provide a better plan or cost estimate. While it is certainly possible to do this without this new programming paradigm, traits made the methodology much more comprehensible, and so helped the subsequent work substantially.

This work also resulted in the “Attean” framework[2], also mainly authored by Gregory Todd Williams with some smaller contributions from me. It is also available in the Linux distribution Debian (and derivatives, such as Ubuntu) as libattean-perl.

4.2.4 Survey of HTTP Caching

I committed to do this survey after Jürgen Umbrich challenged me to see if the standards were actually followed, and thus address Problem [10] before further work to solve practical caching problems. Rather than just survey the usage for SPARQL Endpoints, I decided to do it for the breadth of the Semantic Web. This resulted in a research track paper on the Extended Semantic Web Conference 2015 entitled “A Survey of HTTP Caching Implementations on the Open Semantic Web” with citation:

https://metacpan.org/release/Attean

38

and a technical report that elaborates on data reduction methods and statistical methods with the citation:


The most important contribution of this paper was an understanding the actual usage of caching headers, but it was also intended to raise the awareness of these important Web standards in the Semantic Web community.

Although the support for caching headers is not yet widespread, it is sufficient to perform experiments on real-world data.

The main shortcoming of this paper was that I didn’t attempt to validate the recorded freshness lifetime against the true change frequency recorded by [35] and discussed by [20]. It would be interesting to see how common it is that the given freshness lifetime, both given as standard-compliant caching and from existing heuristics, is wrong. There are two types of errors: That the freshness lifetime is too short, i.e., we could have cached it longer, and that it is too long, i.e., we would serve stale data.

Also, the investigation would be deeper if it tried to address Problem [17]. A single analysis would not be sufficient to do that, laws would require multiple studies over time.

4.2.5 Philosophy of Science

This direction was motivated by a dissatisfaction with the methodology used in the literature, specifically benchmarking, to test assertions of performance. That led to exploration of statistical methods referenced in Section 4.2.2 and to the organising of the Empirical 2014 Workshop at the Extended Semantic Web Conference (which I was unfortunately prevented from attending myself). Frustrated by the slow progress of the last 16 years, and the difficulty in comparing methods
to the more mature natural sciences, I still felt the need to address Problem 18 and express a philosophical view of science, which I did by submitting an article to the “Negative or inconclusive results in Semantic Web” workshop at the Extended Semantic Web Conference 2015. The essay, see [40] was tangential to the Call for Papers, but the chairs decided to accept it as an interview. With Ruben Verborgh in the role of the interviewer, we reformulated it and it was published in the proceedings and is included in the dissertation. It was performed with Jacco van Ossenbruggen as the interviewer in front of the audience. The essay, at [http://folk.uio.no/kjekje/2015/noise-essay.html](http://folk.uio.no/kjekje/2015/noise-essay.html) is however a more thorough account. The citation of the interview is:

Kjetil Kjernsmo and Ruben Verborgh. How can scientific methods provide guidance for semantic web research and development.
In Workshop on Negative or Inconclusive Results in Semantic Web Proceedings, volume 1435 of CEUR Workshop Proceedings. CEUR-WS.org, 2015.

As a philosophical text, the contributions are hard to detail. It should establish Problem 16 as a fundamental problem. It is further written in a provocative style, to encourage discussions of epistemological questions. It also points out shortcomings in current practice, and it is clear that a proper evaluation of the work outlined in Chapter 5 would be a major undertaking.

In retrospect, the greatest weakness of the essay is that I failed to recognise the impact of Frank van Harmelen’s call for studies that can establish laws of the Semantic Web information universe, as discussed in Section 4.1.3. To my defence, I was only made aware of this keynote by the reviewer.

### 4.3 Unpublished Work on Query Caching

Finally, the intention was to bring convergence in the topics of this thesis around a system to cache results that could be reused in subsequent queries. As noted in Section 2.5.5, this was motivated by opportunity as well as problems, but it is intended to address problems 7 and 8. This work was not finished within the time frame of the dissertation, but is discussed at length in Chapter 5. This work further relates to the problems in Section 4.1 as follows: Triple Pattern Fragments shifts most of the processing from servers to clients, but Problem 6 makes that infeasible in some cases. In addition, problems 3, 4 and 5 are compounded by Problem 11.
but alleviated somewhat by the presence of cardinality estimates given by Triple Pattern Fragments, which we exploit.

4.4 Future work; Discussion of Other Problems

As noted in Section 2.5, it is clear that evaluating arbitrary SPARQL queries are in general unsustainable, and it is likely that the problems in Section 4.1.1 are partly to blame for it. Our work, and most of the related work in Section 3.2 approach this problem from the direction of moving stress from the server, but it is also clear that another way to address Problem 1 is for the SPARQL engine to identify queries it would be too heavy to answer already when parsing or planning, and also to expose that in the service description so that clients can avoid asking them. The HTTP/2 specification [8] has some generic facilities that may be helpful:

```
ENHANCE_YOUR_CALM (0xb): The endpoint detected that its peer is exhibiting a behavior that might be generating excessive load.
```

In Section 5.1.3, it is noted that Problem 6 makes it important for a query planner that integrates LDF query answering to find a balance between client and server capability.

The lack of a formalisation of the notion of a useful cache entry is noted in Section 5.1.4, and while there has been some work on this topic, I am not aware of any attempt to do so in the context of the present topic.

Problem 9 has an interesting possible solution in the application of hashing algorithms that have a reasonable chance of returning the same digest if two queries are equivalent. As noted in Section 3.2.2 [63] can be used for this. It is also interesting to discuss the approach [50] took to canonical labelling in HTTP context:

The ETag header, discussed in Section 3.1.1 can be used for a SPARQL result. Such a canonical label could be used in conjunction with an algorithm that tracks changes to the RDF graph to produce an ETag. For slow-changing Linked Data, which is a common case, it is a possibility that a canonical label in combination with a time-stamp of the entire RDF Graph could yield considerable benefits. However, the presented algorithm does not cover SPARQL in its entirety, for example, it does not handle named graphs, solution modifiers and filters, nor is it clear how it handles property paths (which in many cases should be trivial, since a property path often has an equivalent Basic Graph Pattern). Union queries or negation is also not mentioned, so more work is needed for this method to have
general applicability. Nevertheless, if the entire Group Graph Pattern of a certain query is supported, and the query projects all variables, e.g., it wraps the pattern with `SELECT *` or `CONSTRUCT`, the `ETag` would be trivial to construct using the presented algorithm in a non-trivial number of beneficial cases. With this, cached result verification could be done by a mediator using HTTP requests with the `If-None-Match` header and so make it possible to use parts of the engine of this paper usable in the Internet infrastructure.

Problems 3, 4 and 5 has a promising approach in [49], but a way to expose this information to a client or a proxy is lacking. We saw in [28] that a VoID description can be used with some success, but this needs further elaboration. Histograms have also been proposed for this purpose in [58].

I have not approached any of the problems of Section 4.1.2 mainly because I think it is an open question which of these approaches will be adopted in practice, and whatever the solution may be, their designers should make sure they are orthogonal to the other problems discussed here. Thus, it was natural to consider them out of scope.

Clearly, Problem 17 is a very interesting one, with important implications for evaluation, but also because it shifts the focus of our attention. In this work, the methodology used to study Problem 10 could be used to gain such understanding. As noted in Section 2.4, this study is partly motivated by my subjective experience of why the Web was so successful. Clearly, this should be studied more objectively, but I am not aware of any such study. Such study could also be applied to see if the experiences from the emergence of the Web can be applied to accelerating the Semantic Web, like I subjectively propose.

Problem 20 was along with problems 1, 2 and 7 part of the motivation of [61] and [60], and this builds in part on the insight I contributed when addressing Problem 21. Their solution was to define an ontology to describe an interface that can answer a single triple pattern, but since it is hypermedia, it can be described in the messages of the protocol. However, the solution in [60] does not take into account Problem 6 so query planning that shifts the burden to a cost-model driven balance between server, proxy and client is an interesting direction. In this context, it should be noted that HTTP/2 [8] is intended to make multiple requests less expensive, and it is interesting to study the practical impact of this as it could impact the cost of making many HTTP requests substantially.

I believe that the core of Problem 25 lies in the prevalence of statically typed languages. From an engineering perspective, static typing seems like a good idea, since errors should surface at compile time. However, it seems like it comes at
a high cost: Programmers seem incapable of thinking beyond the constraints it sets. Nevertheless, the theoretical framework that seems to solve the problem was proposed in 1991 in the book “The Art of the Metaobject Protocol” [36]. Perl has a mature implementation of the metaobject protocol, and the topic has been discussed on several hackathons, and some code has been written, but the solution seems elusive.

Finally, the problems 18 and 15 should prompt an elaborate approach to understand and apply the practical implications of contemporary philosophy of science. [46] proposes a framework that defines what constitutes a severe test, and one possible direction is to apply this to query endpoint evaluation.

4.5 Conclusions

The conclusions of the papers can be summarised as follows:

Hypermedia can contribute a “View Source” capability to application development and data exploration, and thinking about an RDF statement with control information as a natural language sentence can make RDF understandable to many developers. Therefore, it is one key to developing applications on the Semantic Web.

When engineering systems to encourage contributions from many different parties, it is important to carefully consider how such systems can be extended and adapted, and how contributions can be made a part of the systems and so enhance the base that researchers and other users can benefit from. Traits ease the task of extending database systems with experimental features and eases the usage of optimisations in underlying systems.

Caching in the Web is a prominent topic, but we found that while the adoption of caching and conditional requests as defined in HTTP is not widespread, the adoption is sufficient to start using them in practical systems. We also found that many resources change at a very slow pace, but that it is not reflected in the freshness lifetime set by the server, but may more commonly be computed by common heuristics.

Evaluation methodologies suffer from a poor foundation in philosophy of science, and research into the epistemology of the field is important. This is due to several interrelated factors: The field of research differs from natural science in that it studies a human artefact that does not yet exist. An example of this problem is that there is no formulated coherent theory of the Semantic Web, so the external
validity of any hypothesis relative to such a theory can always be contested. If one would wish to follow the criterion that science is distinguished by falsifiability, it is currently difficult to tell if the evaluation or the hypothesis is to be rejected.

Moreover, along the above directions, and based on the fact that the author has a large intellectual investment in the success of the Semantic Web, the author admits that the study is lacking in terms of objectivity, but notes that the problems are of a general nature.

Empirical evaluations should be developed from the principles of philosophy of science and statistics, and the practice of benchmarking does not adequately follow these principles. Statistical Design of Experiments provides a path to a critical practice of evaluations, needed to improve the foundations of the field. The experiment we designed provided for both an instructive example of a new methodology and how to use the methodology to evaluate the experiment itself.

Finally, the work extended to a SPARQL query engine that could run on a proxy. It could cache intermediate results of triple pattern matches, either in real time or by prefetching results that might be valuable. In query planning and evaluation, it could use the cache, a remote SPARQL endpoint and a Triple Pattern Fragments server. Unfortunately, this work had to be ended before it could reach a conclusion, but a detailed account of the work is included in Chapter 5.
Chapter 5

Prefetching SPARQL Query Cacher

5.1 Introduction

This chapter describes an effort to create a proxy that can cache the results of not only complete SPARQL queries, but also the results of individual triple patterns. It will asynchronously analyse executed queries, and may prefetch the results of certain triple patterns into the cache.

5.1.1 Scenario

The system is in the convergence of the directions described in this thesis: It should relieve the remote endpoint of some of the burden to evaluate the query and it should add to the robustness of the open Web SPARQL infrastructure. It uses hypermedia to answer individual triple patterns, based on Triple Pattern Fragments (TPF) [61]. The system does not support named graphs (i.e. only triplestores), but other than that, it supports SPARQL 1.1 in its entirety. It was possible to develop this quickly by the efforts put into the query planning in the Attean framework described in Section 4.2.3. Caching based on RFC7234 [25] was shown to be of possible use in the survey I conducted. Finally, it was planned to be evaluated using Design of Experiments.

Unfortunately, the system has as of this writing insufficient performance for an evaluation to be meaningful. Nevertheless, it points out some interesting lessons with varying degrees of certainty. This chapter will detail the system, show its design and features and discuss its strengths and weaknesses.

Figure 5.1 illustrates where caches may be in the Internet infrastructure, as well as HTTP requests and responses in a typical deployment scenario of the
The present system may be deployed at any of these levels (and similar figures could be drawn). However, since the database and the reverse proxy in front of it may have detailed knowledge of the data, e.g. a complete data profile with statistics to optimise join order, these levels have much in common with a conventional system.

The system is based on HTTP, which implies a client-server architecture. In this architecture, a cache may be present at conceptually five different levels. First, the client may have its own cache, which caches only responses made by the client itself. The next level is known as a forward proxy. They aren’t currently very common, but have in the past often been institutional proxies, or proxies employed by Internet Service Providers for caching responses that may be common to many of their users. The next level, generically referred to as a caching proxy, may be in the Internet infrastructure. Currently, the most common form of this type of proxy is known as a Content Delivery Network (CDN). It is very common to have a reverse proxy near the server. They are often known as a “Web Accelerator”. Usually, they will communicate with the server with HTTP, but it is conceivable that it may use a different protocol, and may employ detailed knowledge of the hosted data to optimise cache operations. Finally, the database, in this context a SPARQL Endpoint, may use conventional database caching techniques. The database cache and reverse proxy are typically controlled by the data provider.
My interest is the case where the caching proxy has no further knowledge of the data than what is exposed through the SPARQL Endpoint or hypermedia metadata. While this would be known as a mediator in the database literature and has also been a field of study for 20 years, I focus on a practical motivation stemming mainly from what is currently available. In current practice, very little information is available to the mediator. Also, this study is also practically restricted to HTTP. If the cache is to be shared, then it would typically reside on the forward proxy, or in a CDN.

It is further interesting to note the contrast to the HERMES [2] system’s concept of invariants discussed in Section 3.2.2: a SPARQL query cache should be oblivious to domain knowledge in a CDN deployment and therefore shouldn’t assume any invariants, except in the possible but unlikely case where the remote server declares an infinite freshness lifetime for the result of a certain graph pattern. A forward proxy, on the other hand, may have a deeper understanding of the needs of the client, and could possibly implement invariants. The Attean framework accommodates for this.

Figure 5.1 illustrates the case where HTTP messages are being passed, where the cache is assumed to reside on an intermediate caching proxy in the Internet, i.e. a CDN. The figure may be viewed as having time on the x-axis, with the arrows pointing down being requests and the arrows pointing up being responses. In this example, the client first makes a request (request 1), which is passed through the proxy because it has not yet cached any responses. The response (response 1) is then also transmitted directly from the SPARQL Endpoint at the database back to the client. The entire response may be cached on the proxy in case it can be reused in its entirety. However, the main point is that the request will be analysed by the proxy in parallel to being sent to the server, and another request 2 for a triple pattern fragment that the analyser thinks will enable the proxy to assist the server the best for future requests, is sent. This analysis is discussed in Section 5.1.4. The response 2 is not sent back to the client, but will enter the cache. We then assume that the analyser was successful and that the cache can be used to answer request 3. In that case, the proxy will send rewritten queries for the rest of the data it needs to be able to answer the query to the origin server, indicated by the dashed arrows. This may be zero or more SPARQL queries, TPF requests or a combination thereof, depending on what the planner finds least expensive. The proxy will then evaluate the entire query and send the final result back as response 3. In addition, the analyser may have identified another Triple
Pattern Fragment that it takes as likely to be useful for future requests, and sends a request and caches the response \(^4\). At some point, it may be able to answer the entire query using only cached results, as illustrated by request and response \(^5\).

Finally note that in this text, the term Triple Pattern Fragment (TPF) is understood as a specialisation of Linked Data Fragments (LDF), in which a Triple Pattern Fragment server must be able to answer any triple pattern, whereas for a Linked Data Fragments server, this is not required. However, there is no difference in the current implementation, so in practice, the terms could be used interchangeably.

### 5.1.2 Architecture

The architecture is depicted in Figure 5.2. A SPARQL protocol implementation will accept a client’s query (and could cache it’s result in its entirety, but this is neither implemented nor depicted). Two things happen to a valid query:

- The query (or indeed the algebra) is sent asynchronously to a set of analysers that is tasked with predicting what parts of the incoming query may be useful to answer future queries. The result of the analysis is sent to a prefetcher that will decide how to fetch the needed data, currently it only operates on triple patterns, and therefore, and since it is also assumed that a triple pattern fragments server can answer any triple pattern, it instructs a TPF client to fetch and enter the result into the cache.

- The SPARQL protocol implementation will also send the query to a parser that will create an algebra tree from it. The algebra tree is then passed to a query engine for immediate execution. The query engine has an elaborate query planner that can create plans for accessing a cache, a TPF server and a SPARQL Endpoint. When the planner finishes planning, the evaluator may use a TPF client or a SPARQL protocol client to fetch data that is not in the cache. The results of single triple patterns may be inserted in the cache by the TPF client.

### 5.1.3 Understanding the Query Planning

We saw in Section 1.1.2 that a query would be parsed to a tree of algebra objects and then a query planner would traverse the algebra tree to create a plan tree by
Figure 5.2: Architectural overview. The proposed system is represented inside the box titled “Proxy”. It interacts with a client, a Triple Patterns Fragments Server and a SPARQL Endpoint. The most important components are the query engine, which consists of a query planner and a query evaluator, and the analysers that find subqueries to cache.

finding all the ways the data can be accessed, or operations executed over them. For example, if the input query has a BGP with two triple patterns and there are two types of join operations, it may result in four trees, where each of the join operation plans are paired with both access plans for the triple patterns.

It is next the query planner’s task to determine which of the possible plans it should execute. A common choice to perform this task is to estimate the cost executing various choices.

The resulting plan trees may be represented as strings as follows:
Example 7. Four Attean plan trees

- Hash Join (s) (distinct cost: 27)
- Quad { ?s, <http://ex.org/p>, ?o, <g> } (cost: 15)
- Quad { ?s, <http://ex.org/p>, "1", <g> } (cost: 12)

- Hash Join (s) (distinct cost: 28)
- Quad { ?s, <http://ex.org/p>, "1", <g> } (cost: 12)
- Quad { ?s, <http://ex.org/p>, ?o, <g> } (cost: 15)

- NestedLoop Join (distinct cost: 180)
- Quad { ?s, <http://ex.org/p>, "1", <g> } (cost: 12)
- Quad { ?s, <http://ex.org/p>, ?o, <g> } (cost: 15)

- NestedLoop Join (distinct cost: 180)
- Quad { ?s, <http://ex.org/p>, ?o, <g> } (cost: 15)
- Quad { ?s, <http://ex.org/p>, "1", <g> } (cost: 12)

In the above example, the first plan tree will be returned from the query planner for execution, since it has the lowest estimated cost (27). Since the SPARQL engine operates over quads, we have assigned a dummy graph name when constructing the Model, in this case further shortened to <g>.

The system we propose is based on this paradigm, any choices we make are therefore to be encoded into a cost model. The cost model itself is not considered a part of the contributions, as it is just based on heuristics that arises from the following discussion. The heuristics themselves are best considered an implementation detail and is described in Section 5.2.3.

Recall Problem 8, where a BGP with three triple patterns could break into two components if the triple pattern connecting the two others was cached. It is one of the requirements for the heuristics of the cost model that it is be able to avoid such situations.

As mentioned, the proposed query planner takes may access a remote SPARQL endpoint and a Linked Data Fragments Server as well as the local cache. I have also said that a key motivation behind the study is to relieve the remote endpoint of load. It was argued in [60] that LDF shifts the burden of query evaluation to the client, and so relieves the server of load. It is not clear from the literature that this strikes the right balance, it is my belief that the balance must depend on the concrete server and client capabilities, keeping in mind Problem 6. Future work
to determine the right balance in a given situation would be welcome in this area, but I believe it would be advantageous to build on a cost model.

Following this discussion, consider this example:

**Example 8. A four-triple pattern BGP**

```sparql
?s <http://ex.org/p> "1" . # A
?s <http://ex.org/q> <http://ex.org/a> . # B
?s <http://ex.org/p> ?o . # C
?o <http://ex.org/b> "2" . # D
```

Say that the first triple pattern has a result in the cache. This will not cause Problem 8 to occur (it would have occurred if the third triple pattern had a cached result, however), so we can safely use the cache. The results of the three remaining triple patterns must be fetched from a remote server. There are essentially two reasonable choices: Either fetch the result from an LDF server as argued by [60] and join them locally or send a single, rewritten SPARQL query to the remote endpoint for the evaluation. When the results are returned from both the cache and the remote endpoint, they must then finally be joined to produce the final result returned to the client. A string representation of the plan tree from the current implementation reflects this and reads (note that accessing the cache is represented by an iterator plan):

**Example 9. An example Attean plan with a remotely executed BGP**

- Hash Join { s } (distinct cost: 74)
- SPARQLBGP (cost: 60)
  - Quad { ?s, <http://ex.org/p>, ?o, <g> }
  - Quad { ?o, <http://ex.org/b>, "2", <g> }
  - Quad { ?s, <http://ex.org/q>, <http://ex.org/a>, <g> }
- Iterator (?s with 2 elements) (cost: 2)

Merging the quad plans into a single SPARQL BGP plan is not straightforward in the presence of a quad pattern that is to be looked up in e.g. a cache. If we assign the letters A, B, C and D to triple patterns in the order expressed in Example 8, the initial join tree produced by parsing may be (((A ∨ B) ∨ C) ∨ D). Now, we are interested in a plan on the form A ∨ BCD, i.e. BCD is reduced to a more fundamental plan type, in this example a SPARQL BGP. This will minimise the number of HTTP requests, in much the same way as the exclusive groups of [55]. To facilitate this, we employ *join rotation*. A simple example of join rotation
would be this: We first generate subplans for both \(((A \bowtie B) \bowtie C) \bowtie D\) and \((A \bowtie B) \bowtie (C \bowtie D)\). Then, \(C\) and \(D\) can be merged to a SPARQL BGP to \((A \bowtie B) \bowtie CD\). Another rotation can then be performed: \(A \bowtie (B \bowtie CD)\) and then coalesced to the final result. The planner will generate alternative plans for both rotated and unrotated plans, and all generated plans are subject to cost estimation, so whether a large BGP plan will be evaluated remotely, or as a join of local cached results and a smaller remote BGP, or a remote Triple Pattern Fragments server, is up to the cost estimation. This will be discussed in detail in Section 5.2.3.

As can be imagined from the fact that a BGP with just two triple patterns result in 4 plan trees, a naive query planner may generate a large number of plans for complex queries. In addition, the proposed system will generate even more plans, since it considers three sources of the same data, the cache, the remote endpoint and an LDF server. Various methods have been developed in the literature to limit the growth of plans. In addition to a naive planner, we also have an Iterative Dynamic Programming planner [41], which prunes plans aggressively based on their estimated cost in each iteration of query plan generation.

5.1.4 Analysis for prefetching

The objective of the analyser in Figure 5.2 is to predict what remote data is most useful to answer future queries. I haven’t attempted to formalise the notion of what a useful cache entry is, nor have I attempted to exploit the concept of profitable query patterns of [50], mostly because this work came at a time when my work was already near the deadline. I opted for a simple approach to consider only individual triple patterns.

With this constraint, the analysers have been implemented with two different assumptions of usefulness: The first analyser, called predicate-count, captures frequently occurring bound predicates. This is due to the observation that predicates are often bound, and so predicates that are common will occur in the future, and therefore, prefetching and caching results of a triple pattern where the predicate is bound to a frequently occurring is likely to be beneficial. This is done by counting the occurrences of predicates, and when the number exceeds a certain threshold, the prefetcher is instructed to fetch the data.

The second analyser, named cost-reduction, works along a different axis: It will find the triple pattern that, if cached, would provide the greatest cost benefit. To do so, it will rerun the query planner on the incoming query while simulating that the results of a certain triple pattern was in the cache. The simulation will run
for all triple patterns that do not already have a cached result. If the cost reduction of the best plan is larger than a certain limit, the prefetcher is invoked.

5.2 Implementation

This section explores the implementation of the general architecture outlined in Section 5.1.2.

The implementation consists of several modules that have been published to the Comprehensive Perl Archive Network as Free Software, and most of which have become a part of the Free Software ecosystem. For a detailed account of the modules that are a part of the system and the authors that have contributed code in this ecosystem, see Appendix 5.A. Some parts are essential for the discussion and to understand the contributions, and therefore detailed in the following:

5.2.1 The Attean Framework

The Attean framework was born from the experiences the PerlRDF community gained with RDF::Trine, the Perl counterpart to the better known Jena framework, and the opportunities presented by the introduction of traits [54], or roles as they are known to the Perl world.

Traits are groups of methods that serve as a primitive unit of code reuse. Traits are not constructed into objects themselves and do not use inheritance for composition, instead a class is composed by a set of traits and possibly the class’ own methods, and then instantiated. A trait can also require a set of methods or attributes, and as such, is similar to an interface, but since its focus is code reuse, it can also provide a default implementation.

The Attean framework packages commonly used classes when using Semantic Web data, like RDF statements and terms, parsers, serializers, triple/quad stores, iterators, but also features that are relevant to query answering, like algebra, query planners and plans. Additionally, it has an underlying extensive API consisting of roles that can be composed to the above classes.

Perl is an interpreted language and is not focused on speed. Instead, the focus is on developer efficiency. However, our motivation for Attean, expressed in [62], was to enable the use of faster implementations by using roles. It also serves to simplify code reuse.
For this work, the usage of roles to simplify query planning is of particular importance. While we do not exploit optimisations closer to the database, we extend the query planner in several directions with several separate add-on modules, with only generic functionality in the core Attean framework.

We have found it convenient to use classical inheritance in combination with traits-based composition, but only when the base class provide just fundamental functionality that is very likely to be common to all implementations. For an example of this, see Section 5.2.2.

Also of importance are the APIs to compose stores. Stores in Attean can be triplestores or quadstores, they can be mutable, bulkupdateable, can enable the use of ETags or modification times for conditional requests (as defined in RFC7232 [25]), and enables further extensions. Each of these primitives is represented by a role that implements default functionality or simply requires such functionality to be present. For instance, a quadstore is required by Attean::API::QuadStore to implement the get_quads method, which can take an RDF term as one or more of the arguments and variables for the rest, and should return an iterator over the quads that were matched. Based on an implementation of the get_quads method, the Attean::API::QuadStore role provides default implementations of methods count_quads, count_quads_estimate, get_graphs and size. These methods may be overridden with more efficient implementations when composing a class. Composing other roles will require further methods to be implemented.

Stores can be rather diverse, so in addition, Attean provides another abstraction layer, called a Model. Models consistently operate over quads, in the case where underlying store is a triplestore, a graph name will be set, or it may wrap several triplestores with a graph name for each. In addition, Models provide higher level methods, for example a subject method to list all subjects, and corresponding for other terms. Thus, upper layers of an application should use the Model abstraction. We have also chosen to implement significant parts of the query planning in the Model.

Based on the algebra tree from a parsed query (see Section 5.1.3 for details), models may generate plans for any algebra object of their choosing by implementing the Attean::API::CostPlanner role. This amounts to implementing plans_for_algebra and cost_for_plan methods. For a given algebra, Attean’s query planner will trust that the model’s plans_for_algebra is better than its own by default, a choice that was justified in [62]. Additionally, Attean has a simpler mechanism for single quad patterns: By default, quad patterns are evaluated by calling the Model’s get_quads, but this may be augmented by adding a wrapper
around an access_plans method used by the query planner to produce further plans.

Based on the algebra tree and plans generated by the models or the default query planner, as well as the access plans, several alternative plans will be generated, and their cost will be estimated. How this happens in detail depends on how the planner is composed. Attean comes with two roles that implement different planners, Attean::API::SimpleCostPlanner will compute all possible plans and then estimate the cost of all plans, and finally return the best 5. Attean::API::IDPJoinPlanner implements an Iterative Dynamic Programming planner [41]. When a final plan tree has been found, the query may be executed.

That new plan types can be integrated into the planner with ease, and that the planner itself can be changed with very little effort is a key contribution of the Attean system, which I have taken advantage of in this work. That this is made easy is to the credit of traits, as will become clear as we study the details of the implementation in Section 5.2.2.

5.2.2 The Caching Proxy

It is interesting to add further implementation details to the overview in Figure 5.2 with a list of the implemented components of the caching proxy:

- Analysers, used to determine whether certain triple patterns should have their results prefetched and cached.
- A prefetcher that performs the needed actions.
- Two query planners, one that can use a local cache and a remote endpoint, and another that extends this to be able to use Triple Pattern Fragments as well, and corresponding models.
- Two roles to generate plans for accessing cache and Triple Pattern Fragments and a corresponding plan class to enter results in the cache.
- A custom User Agent for caching.
- The actual caching proxy that accepts queries, runs the planner and returns the results.
- Scripts to run the whole system.
In terms of technical contribution, most of these are straightforward. The proxy itself, for example, is just a few lines of wrapper code around the At-

teanX::Endpoint module. The User Agent is a class that composes the roles of LWP::UserAgent::CHICaching and adds a hash of the query as the cache key.

The main technical contributions are in the analysis of the query and in the
query planning.

**Analysis and Prefetching**

Analysers and prefetchers run asynchronously with query evaluation. This is
achieved by an addition to the proxy that sends a query to a persistent analyser script that subscribes to a channel on a Redis\(^1\) data structure store. The Redis data structure store is a mostly-memory “NoSQL” database for storing primitive data structures identified by a key. It also features a publish-subscribe system.

The analyser script can be configured to use any number of analysers. Two such analysers have been implemented as according to the ideas in Section 5.1.4. Any further analysers that operate on single triple patterns are trivial to implement, whereas analysers for more advanced query patterns would require new plan classes as the current cache system relies on using the access_plans method, which is applicable only to single triple/quad patterns. Any triple patterns found by the analysers is published by them to another Redis channel for the prefetcher to retrieve.

Another persistent script is subscribed to this latter channel. If the Model supports LDF, a retriever will then download the results of the triple pattern and enter it into the cache, or use a single triple pattern SPARQL query if not. The current implementation assumes that both an LDF server and a SPARQL endpoint can be used to answer any triple pattern. Relaxing this assumption depends on the LDF specifications gaining stronger semantics to link SPARQL endpoints and LDF servers together. With that, the assumption could be abandoned trivially.

The cache itself is assumed to only cache the results of triple patterns where at least one term is bound. I have used the Redis data structure store for the cache as well. This choice is somewhat arbitrary, and has not proved very successful. The cache will store the results as an array if two RDF terms are bound in the triple pattern, or a hash if only one term is bound. The strings put into the cache are serialized N-Triples strings.

---

\(^1\)See [http://redis.io/](http://redis.io/)
In addition to caching the prefetched results, the proxy may also cache the serialized results of any full SPARQL query, and also enter the results of any Linked Data Fragment retrieved when the query planner evaluates a query into the above cache, using a trivial extension to the plan class packaged with the LDF client code described in Appendix 5.A.2.

Cache maintenance, for example purging elements that are not useful or has expired from the cache, is an orthogonal problem to the proposed system. It is currently done straightforwardly in the User Agent described in Appendix 5.A which supports HTTP caching as specified in RFC7234 [25], but further sophistication could be added without modifications to the rest of the system.

**Query Planner**

The query planner is written as completely stand-alone modules, based on the facilities in the Attean framework, but without accessing Attean internals. The system has in fact three query planners:

- `AtteanX::QueryPlanner::Cache`,
- `AtteanX::QueryPlanner::Cache::LDF` and
- `AtteanX::Query::Cache::Analyzer::QueryPlanner`.

Each is an extension of the previous. The first is written to use a remote endpoint in combination with a local cache. The second extends this to also be able to generate and use TPF plans. The final is a query planner for the analyser in Section 5.2.2 that reruns the query planning with all triple patterns in a simulated cache.

These query planners are a good example of the virtues of traits-based programming, as they are typically customised along different axes: access plans, cost estimation, planning of joins within Basic Graph Patterns or left joins, re-writing (like gathering quad patterns to blocks), etc. In the case of our query planner that combines a remote endpoint with a local cache, the module named above `AtteanX::QueryPlanner::Cache`, the basic query planner of the Attean framework, `Attean::QueryPlanner` is extended (i.e. inherited from), but also composes the following roles:

- `Attean::API::NaiveJoinPlanner`,
- `Attean::API::SimpleCostPlanner`,

57
• AtteanX::API::JoinRotatingPlanner,
• AtteanX::Query::AccessPlan::SingleQuadBGP and
• AtteanX::Query::AccessPlan::Cache,
out of which the three first are a part of the Attean framework, the next is described in Appendix 5.A.1 and the last is a part of the query cache system itself.

Attean::QueryPlanner has only minimal functionality, everything else is provided by the roles. Conventional single inheritance is constrained to a tree structure, and would result in a very complex system. Another option is to use the factory method pattern, where a factory object would need to be initialised to return a suitable object for each axis of customisation.

The implementation of the join rotation detailed in Section 5.1.3 is in the role AtteanX::API::JoinRotatingPlanner. In the planner AtteanX::QueryPlanner::Cache adjacent quad patterns are coalesced into Basic Graph Pattern (BGP) plans, and BGP plans are coalesced into larger BGP plans, so that they can be evaluated by the remote SPARQL endpoint as a single unit. The extension AtteanX::QueryPlanner::Cache::LDF adds the option of querying remote LDF servers.

5.2.3 Cost Model

The final query execution is designed to be entirely cost-based, i.e. many alternative plans are generated, but the decision on whether to use a certain plan is entirely up to the cost model. While the framework is flexible and could support further research into cost models, I have considered this to be future work, and only a small heuristic cost model is present in the current system. This is guided by intuition and verified mainly by a number of tests in the code for expected corner cases. Importantly the different parts of the cost model are tuned to balance each other, not actually estimate the cost of any actual operation.

Admittedly, with this approach, using the term “cost model” is somewhat of a misnomer. An important motivation for this work is to relieve the remote endpoint of stress, and so a principled approach could attempt to estimate the cost for the remote endpoint, but rather, the intuition explained in Section 5.1.3 motivates the choices made in the following.

Moreover, since the combination of remotely executed SPARQL with a local cache and TPF is rather complex, the cost heuristics are fairly elaborate. Therefore, an attempt to create a proper cost model would make that a primary focus of
the research, rather than the empirical analysis of a first-generation caching proxy integrating SPARQL and LDF we set out to achieve. That being said, I have chosen to keep the “cost model” terminology since the usage of the metric resembles how it is used in the literature.

Throughout the system, the query planner takes the cost from the models implementing the Attean::API::CostPlanner role. A key challenge is to balance the costs given by different models in the system, and this is done in the module AtteanX::Model::SPARQLCache::LDF. Let us, however, start from the bottom.

The cost of evaluating a Triple Pattern Fragment plan $C_{tpf}$ is designated to be

$$C_{tpf} = 10 + \lfloor 990 \frac{n_{tpf}}{n_{tot}} \rfloor,$$

(5.1)

where $n_{tpf}$ is the estimated number of triples expected to be matched by the triple pattern and $n_{tot}$ is the size of the model. Thus, this cost will be an integer between 10 and 1000. The reason we use a floor function is that the Attean cost model only supports integers. This choice is mainly motivated from the assumption that evaluating TPFs is a relatively cheap operation for the remote endpoint, and so the cost should be contained within an interval, so that other costs can easily be balanced against it.

I have assumed that the cost of accessing the cache is usually lower than the above, that is, it should usually be below the lowest cost of a TPF (yet have no upper bound), and is therefore designated as

$$C_c = 2 + \lfloor \log_{10} n_c \rfloor,$$

(5.2)

where $n_c$ is the number of triples in the cache that matches the given triple pattern.

The cost of evaluating a remote SPARQL query is estimated to be the number of subplans (usually quad patterns) incremented by one and multiplied by 100 if there is a common variable between all the subplans, or 1000 if there is not. This is to prevent the remote endpoint from having to compute Cartesian joins as much as possible, thus addressing Problem 8. A remote SPARQL query with only one triple pattern is assumed to always be more expensive than to evaluate than a Triple Pattern Fragment (recall that we want to relieve the remote endpoint of stress), and so is given the cost of 1001.

Next, we detail the cost of a join where one or more children in the tree is a remote SPARQL plan. It is usually not wanted to execute several remote queries, it is assumed that the number of HTTP requests should be small, and that plans
where the joins have been rotated and coalesced by the implementation in the role AtteanX::API::JoinRotatingPlanner should be less expensive. Therefore, such plans should be penalised, but by how much depends on the type plans. Conventionally, for a nested loop join plan, we first assign

\[ T_{A\triangleleft B} = 10((1 + C_A)C_B), \]

(5.3)

where \( A \) and \( B \) signifies the left and right subplans, and \( C_A \) and \( C_B \) their respective cost estimates (which is originates from the other heuristics in this section). Following the default implementation in Attean, we multiply the cost in Eq. (5.3) by a factor of 10 if there are no shared variables, to mildly penalise Cartesian joins.

If the plan is a hash join, we continue to follow the example of the Attean implementation and set

\[ T_{A\triangleleft hB} = 10((1 + C_A) + C_B). \]

(5.4)

Further, if there are no shared variables, the cost of Eq. (5.4) is multiplied by a factor of 100 rather than 10 so that if it is necessary to evaluate a Cartesian join, it is more likely done using a nested loop join.

Then, to ensure that the join rotation and coalescing can be efficient, we penalise plans that have more of them, so for both these join types, we designate the cost of joins to be

\[ C_{A\triangleleft \ast B} = \lfloor 1.2n_{rs}T_{A\triangleleft \ast B} \rfloor, \]

(5.5)

where the asterisk denotes either \( nl \) for nested loop or \( h \) for hash join, respectively, and \( n_{rs} \) is the number of remote SPARQL Basic Graph Patterns in the tree.

Under the assumption that it is better to join remotely than to possibly retrieve large amounts of data, a similar algorithm is used to penalise join plans that contain many Triple Pattern Fragment plans, by multiplying the cost of the join plan by the number of the Triple Pattern Fragment plans in the tree. Example 8 is an example of a situation where this heuristic ensures a larger BGP. Whether this is a reasonable choice would have been a subject of the planned evaluation.

Finally, if a plan has a Triple Pattern Fragment plan that has a common variable with a remote SPARQL query, that plan is given an added cost to 1000, to allow plans where the triple pattern has been rotated into the BGP win, again to reduce the amount of data needed to be transferred.
5.3 Evaluation

The system was developed in a practically oriented test-driven manner, with emphasis on testing cases that I assumed *a priori* to be difficult. The interested reader may like to refer to the published code, given in Section 5.2. Some of the tests consist of a number of queries, and some of the tests concern the problem I found in a preliminary study, that the cached triples would often break subsequent queries to cause Cartesian joins. Much of the above cost model is designed to prevent this from having a detrimental impact.

While I am quite confident that the code is producing query plans as intended and that analysis and prefetching is able to retrieve and cache results, an elaborate evaluation would be needed to explore further whether the system actually helps remote endpoints and is able to serve clients with adequate performance. Unfortunately, the evaluation showed that the latter is not the case with the current implementation, and the practical constraints of the project then prevented further investigation. Thus, this section consists of an account of the evaluation that was performed and the methodology I planned to employ.

5.3.1 Actual Evaluation

I decided to set up the entire system as a proxy server, and run a realistic query against DBpedia repeatedly. The intention was to let the analyser find all required data and eventually add that to the cache. Once the cache was filled with all the data, the query cache should be able to answer the query on its own, at least within a time period on the order of a magnitude of the original endpoint (which took on the order of 1 second).

The following query was used:

```sql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX dbo: <http://dbpedia.org/ontology/>  
SELECT ?name WHERE {  
  ?s a foaf:Person ;  
  foaf:name ?name ;  
  dbo:wikiPageID 9828878 .  
} ORDER BY ?name
```

The query was chosen because of its wide variety in terms of selectivity of the individual triple patterns. The retriever managed to pull all the data from the LDF version of DBpedia within some hours.
Table 5.1: Top 15 Subroutines Calls as measured by the profiler Devel::NYTProf.

<table>
<thead>
<tr>
<th>Calls</th>
<th>Exclusive Time</th>
<th>Inclusive Time</th>
<th>Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>13309080</td>
<td>244s</td>
<td>549s</td>
<td>IRI::parse_components</td>
</tr>
<tr>
<td>13309080</td>
<td>218s</td>
<td>218s</td>
<td>IRI::CORE:match (opcode)</td>
</tr>
<tr>
<td>8774329</td>
<td>116s</td>
<td>270s</td>
<td>Attean::Result::new</td>
</tr>
<tr>
<td>10573035</td>
<td>111s</td>
<td>1420s</td>
<td>AtteanX::Parser::NTuples::eat_node</td>
</tr>
<tr>
<td>6355072</td>
<td>79.7s</td>
<td>1856s</td>
<td>Attean::CodeIterator::next</td>
</tr>
<tr>
<td>2</td>
<td>77.4s</td>
<td>4540s</td>
<td>Attean::Plan::HashJoin::<strong>ANON</strong>[...</td>
</tr>
<tr>
<td>4217970</td>
<td>70.0s</td>
<td>847s</td>
<td>Attean::Literal::new</td>
</tr>
<tr>
<td>9091102</td>
<td>61.8s</td>
<td>593s</td>
<td>Attean::IRI::new</td>
</tr>
<tr>
<td>2419260</td>
<td>50.3s</td>
<td>208s</td>
<td>Attean::API::Result::join</td>
</tr>
<tr>
<td>4217968</td>
<td>48.7s</td>
<td>1463s</td>
<td>AtteanX::Query::AccessPlan::Cache[...]</td>
</tr>
<tr>
<td>24186545</td>
<td>48.1s</td>
<td>56.6s</td>
<td>Role::Tiny::does_role</td>
</tr>
<tr>
<td>28128408</td>
<td>44.4s</td>
<td>53.8s</td>
<td>Attean::Result::value</td>
</tr>
<tr>
<td>16871877</td>
<td>44.1s</td>
<td>470s</td>
<td>Attean::Literal::<strong>ANON</strong>[(eval 234)[...</td>
</tr>
<tr>
<td>4217969</td>
<td>40.0s</td>
<td>92.0s</td>
<td>Attean::API::Literal::<strong>ANON</strong>[...</td>
</tr>
<tr>
<td>13309111</td>
<td>39.5s</td>
<td>39.5s</td>
<td>IRI::CORE:regcomp (opcode)</td>
</tr>
</tbody>
</table>

The cache was kept in a Redis data structure database on a machine with an Intel Core2 Duo E8500 CPU with 16 GB of RAM. The proxy server with the query engine ran on an Intel Xeon CPU E3-1230 v5 CPU with 32 GB of RAM, residing a LAN with a gigabit Ethernet link between them. Running the query on this hardware took more than 1000 seconds without any performance analysis enabled. This is clearly more than two orders of magnitude longer than acceptable. Moreover, it is then clearly not possible to draw any conclusions as to whether the system would be helpful in achieving its goal of taking load off the remote server, as it could process far fewer queries than the public remote endpoint, and any load saved on the remote endpoint would likely be less than the uncertainty arising from that.

Nevertheless, it is interesting to further understand this failure. In Table 5.1, I have tabulated the result of the subroutine calls that take the most time.

It is clear that the vast majority of time spent is due to my cache implementation. When using the Redis cache, I serialise RDF terms as N-Triples, and when they are retrieved, they have to be parsed and instantiated as objects for the query engine. Remarkably, parsing of Internationalized Resource Identifiers (IRIs) is the single most time-consuming task. While each parse does not take much time, it happens more than 13 million times. The reason why parsing IRIs happens is that...
a certain amount of parsing is required to compare them [22], and the IRI mod-
ule is a rigorous implementation of the standard that does syntax based resolution
if an object is constructed with a base. IRI parsing happens not only when an
Attean::IRI object is constructed, but may also happen when an Attean::Literal
object is constructed, since the data type is a IRI.

While most of the time is spent in parsing and comparison, the hash joins
also require too much time. A discussion of possible improvements will follow in
Section 5.4.2

5.3.2 Planned Evaluation

This section outlines the evaluation I had planned to employ on the query cacher.
The essence is to perform a “query log replay, while recording remote endpoint
load under different circumstances, determined by a experiment designed with
statistical Design of Experiments formalism”.

In more detail, query logs can be obtained from for example USEWOD [44] or
the Linked SPARQL Queries Dataset [53]. An extraction of one of those datasets
could be performed, to create a representative progression of queries. The data
normally residing on a remote endpoint (i.e. “server”) would then be loaded into
a single database system under my control. A proxy system would be set up using
our system. Finally, a client would submit the queries to the server, via the proxy.

On the server, some metric of system load would be recorded. In statistical ex-
perimentation parlance, this is known as the “response variable”. I had not entirely
decided which metric to use, but [24] indicated the load average may be a good
choice. It could be trivially recorded by polling the virtual file /proc/loadavg
on the host system. Possibly a correction for any extra time spent by the proxy
could be applied.

As for the overall design of the experiment, the reader may refer to the paper
discussed in Section 4.2.2 for an introduction to the type of evaluation intended to
be used. As noted, a 2\(^n\) factorial experiment is the simplest experiment to set up,
and I would therefore go to great lengths to ensure that such an experiment could
be used.

It would also be advantageous to design the experiment so that a combination
of factors would correspond to the null hypothesis, i.e. that the proxy was not
in use and the remote endpoint would tackle the whole load. I did not think this
complete through to a concrete experiment, but it might be a challenging problem
to maintain orthogonality, i.e. ensure that all level combinations can occur in the same number of runs.

However, the factors that may be interesting to investigate include both factors that evaluate the analyser and the query planner. These factors might be:

1. Use of the cache and remote endpoints where the levels are on and off.
2. Use of a caching LDF client, levels on and off.
3. Use the predicate count analyser, levels on and off.
4. Use the best cost improvement analyser, levels on and off.
5. The order of the application of the different methods for analysis.
6. Which cost planner to use, with levels being the simple cost planner and the Iterative Dynamic Programming planner, see Section [5.2.1]

While this design would achieve the goal that a combination of levels would correspond to the null hypothesis, certain levels are somewhat problematic. For example, the LDF client in point 2 might be an implementation of [60], but that would be very different from the combination of point 1 and 2, which correspond to the usage of AtteanX::QueryPlanner::Cache::LDF.

If the experiment outlined here could come to fruition, and we take the factor in the first point with level 1 as the cache being enabled, then we can examine the experiment similarly to what we did with the “Implement” factor in [39]. That is, the first step to understand the contributions to the endpoint system load by the different factors, is to create a normal plot and then use the Lenth criterion for significance. For the next step, to test the alternative hypothesis that the endpoint system load has decreased, form an average of the effects for all combinations of the factor in point 1 and the insignificant factors determined in the first step, based on the average over the rest of the factors. Based on this, a common one-sided two-sample t-test can be performed.

5.4 Discussion

5.4.1 Join Rotation and Iterative Dynamic Programming

While the system used a simple brute-force query planner in the above evaluation, using an Iterative Dynamic Programming planner [41] would simply amount to composing Attean::API::IDPJoinPlanner in the planner class.
While this performed better in most cases, the test suite contains a query where it does not:

**Example 10.** Query with results of 2 triple patterns cached and unconnected triple pattern.

```sparql
?s <http://ex.org/p> "1" ;
   <http://ex.org/p> ?o ;
   <http://ex.org/q> _:xyz ;
```

Note that the above Basic Graph Pattern contains a triple pattern that does not share a variable (?a) with any other triple pattern, thus creating a problematic Cartesian join. Additionally, in this test, the results of the two first triple patterns have been cached. *A priori*, I have assumed that the best plan is a plan where the cache is used for these two triple patterns, the last triple pattern is answered by a LDF server and the last two triple patterns are coalesced into a new SPARQL query, and evaluated remotely. This is also coded as a pass criterion for the test. With the simple query planner, this test passes, but not with the IDP planner, as it results in a plan where the two triple patterns that should run is a single SPARQL query are split into two. This resulted in an interesting observation:

As noted in Section 5.1.3, we rely on rotating joins to gather triple pattern plans into a larger structure that can be evaluated together, like a remote SPARQL BGP. On careful inspection, I found that the join rotation would have had to descend into great-grandchildren of the top join plan. However, with the IDP planner, plans are aggressively pruned, and therefore, this constellation is not considered. This led us (with Gregory Todd Williams) to conjecture that this weakness is inherit to IDP planning (and possibly a general problem with any early-pruning planner) when using join rotation. However, we did not attempt to examine this any further.

To remedy this problem, we considered creating a custom query planner for Basic Graph Patterns, but this was not implemented. The outline would have been as follows:

1. Find triple patterns that are connected with a common variable, and group them to “components”.

2. If a triple pattern has a cached result, create a plan that accesses the cache.
3. If the triple pattern in the previous step causes a Cartesian join to be created within a component, create an equivalent additional plan for a SPARQL remote evaluation that does not access the local cache.

4. Any remaining components consisting of a single triple pattern result in LDF plans.

5. Any remaining components are turned into SPARQL remote BGP plans.

This procedure would allow the cost model to balance the cost between doing a Cartesian join and doing a remote SPARQL evaluation of data already in the cache, as plans for both these options would be provided.

5.4.2 Remedies for Performance Issues

The issue could be raised that it was a bad idea to begin with to use a scripting language like Perl to write a query engine with performance requirements. Indeed, that may be the case. However, generally accepted practice is to first find the actual performance issues and then optimise using a bridge to first and foremost the C family of languages as needed. Importantly, with the breadth of the scope, I only had some months to implement the query engine. This is not a software engineering project, but given past industrial experience, it seems unlikely that I could have achieved it within the allowed time frame using older paradigms, such as the factory method pattern.

The profiling points out clearly where large gains could be made. First and foremost, parsing must be avoided, and that could be avoided by ensuring that IRIs where canonicalised before they were written to the cache, so that comparing them later would be a simple string comparison. Secondly, the Redis data structure store could be better exploited to store strings that could be used to construct literals and IRI objects without parsing.

However, since computing the join itself is too slow, it is not necessarily a solution worth pursuing. The main motivation behind caching hashes and arrays in Redis was to save space; since at least one term would be bound in any cached pattern, it shouldn’t be necessary to insert more data into the cache. Since the performance of doing the join is poor, this may not be justified. Instead, it may be a better idea to use a conventional quad store for the cache. In that case, the planner would require a simple modification so that triple patterns are gathered into locally evaluated BGPs and remotely evaluated BGPs as well as LDFs. To
take full advantage of this approach, SPARQL 1.1 VALUES would also need to be supported, so that the local quad store could perform all join operations. With this in mind, I wrote a rudimentary Virtuoso store driver for Attean, but at that point, the project was overdue and needed to be concluded.

It seems likely that the project, with more time, could be brought to the point where the evaluation in Section 5.3.2 could be applied, and new insight could be gained.

Thus, the main contribution from this project is the ease with which it was implemented.

5.A Appendix: Modules and Their Contributors

Installing the system on the top of a Debian system with only essential packages require 146 Debian packages. Running the test suite requires even more. As such, it builds on the code of hundreds of authors, too many to enumerate. I shall constrain myself to list the modules that have code that has been motivated by this study. Modules that have other primary authors than myself have been started prior to this present project, but may have substantial contributions from this project. The Attean framework was started as part of the effort in Section 4.2.3 but even though it has been developed further with the requirements that arose from this project and has contributions from me, the vast majority of the code has been written by Gregory Todd Williams, who would have been a co-author of this work had it been published in the peer-reviewed literature. The main work in this project has gone into the module AtteanX::Query::Cache, where I have authored the vast majority of the code. The relative difference in authorship can be further examined by checking the linked Github repositories, or by using git2prov 2 to create RDF data of the commit history. Table 5.2 shows the modules that have been influenced or written as part of this project.

\footnote{See \url{http://git2prov.org/}}
<table>
<thead>
<tr>
<th>Module</th>
<th>Authors</th>
<th>Github URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AtteanX::Query::Cache</td>
<td>Kjetil Kjernsmo, Gregory Todd Williams</td>
<td>kjetilk/p5-atteanx-query-cache</td>
</tr>
<tr>
<td>AtteanX::Store::SPARQL</td>
<td>Kjetil Kjernsmo</td>
<td>kjetilk/p5-atteanx-store-sparql</td>
</tr>
<tr>
<td>AtteanX::Store::LDF</td>
<td>Kjetil Kjernsmo, Patrick Hochstenbach</td>
<td>phochste/AtteanX-Store-LDF</td>
</tr>
<tr>
<td>LWP::UserAgent::CHICaching</td>
<td>Kjetil Kjernsmo</td>
<td>kjetilk/p5-lwp-useragent-chicaching</td>
</tr>
<tr>
<td>Attean</td>
<td>Gregory Todd Williams, Kjetil Kjernsmo</td>
<td>kasei/attean</td>
</tr>
<tr>
<td>AtteanX::Endpoint</td>
<td>Gregory Todd Williams</td>
<td>kasei/atteanx-endpoint</td>
</tr>
<tr>
<td>RDF::LDF</td>
<td>Patrick Hochstenbach, Gregory Todd Williams, Jakob Voß, Kjetil Kjernsmo</td>
<td>phochste/RDF-LDF</td>
</tr>
</tbody>
</table>
The system also depends on modules I have written that are not part of the project, but that I have enhanced as time has allowed:

- RDF::LinkedData,
- RDF::Generator::Void,
- Test::RDF,
- URI::NamespaceMap and
- RDF::NS::Curated.

To further understand the roles of the different modules, note the following: LWP::UserAgent::CHICaching is a traits-based implementation of the majority of RFC7234. The underlying caching framework used by this module, CHI, will take care of expiry and purging of entries, since this module will supply an explicit freshness lifetime. AtteanX::Endpoint is an implementation of the server side of SPARQL 1.1 Protocol. Another two modules require further details:

5.A.1 SPARQL Protocol Client

The AtteanX::Store::SPARQL module is a partial SPARQL Protocol client that implements Attean APIs. The store implementation itself composes the supplied TripleStore API and implements methods to retrieve the results of a single triple pattern and exact cardinality for a given triple pattern, by using an aggregate query. In addition, it can generate plans for Basic Graph Pattern algebra objects that have more than one triple pattern. If that is the case, it will return an instance of the class AtteanX::Plan::SPARQLBGP, which is also defined by the module. If it is not the case, the module also has an implementation of the access_plans method in a AtteanX::Query::AccessPlan::SingleQuadBGP role, that can be composed by a query planner to provide a single triple pattern AtteanX::Plan::SPARQLBGP object. Finally, it also provides a model implementation that composes the Model and CostPlanner APIs. This contains a cost_for_plan implementation that will provide a cost for AtteanX::Plan::SPARQLBGP objects that are proportional to the number of triple patterns in the plan, but penalises plans that do not have triple patterns that are connected through a variable (i.e. will cause a Cartesian join) with a factor 10. See Section 5.2.3 for details on the cost heuristics.
5.A.2 Linked Data Fragment Client

The Linked Data Fragment client work was started prior to this project by Patrick Hochstenbach, and consists of the client code for Triple Pattern Fragments in RDF::LDF. It also includes the Basic Graph Pattern optimisation from [60]. Included with the client code is an implementation of a triplestore of the legacy RDF::Trine framework that predates Attean. He also started an Attean store implementation AtteanX::Store::LDF as a separate module, but that module was subsequently adopted by me, and I have written most of the functionality. It composes the TripleStore and CostPlanner APIs and implements triple pattern queries and cardinality estimates, both provided by the underlying client code as they are a part of the core functionality of TPF, as well as cost_for_plan.

It also supplies a plan class AtteanX::Plan::LDF::Triple used in the query planning.
Bibliography


Chapter 6

Papers

The following papers are used for this thesis:


The necessity of hypermedia RDF and an approach to achieve it

Kjetil Kjernsmo

Department of Informatics, Postboks 1080 Blindern, 0316 Oslo, Norway
kjekje@ifi.uio.no

Abstract. This paper will give an overview of the practical implications of the HATEOAS constraint of the REST architectural style, and in that light argue why hypermedia RDF is a practical necessity. We will then sketch a vocabulary for hypermedia RDF using Mike Amundsen’s H Factor classification as motivator. Finally, we will briefly argue that SPARQL is important when making non-trivial traversals of Linked Data graphs, and see how a bridge between Linked Data and SPARQL may be created with hypermedia RDF.

1 Introduction

Mike Amundsen defines hypermedia types [3] as

Hypermedia Types are MIME media types that contain native hyperlinking semantics that induce application flow. For example, HTML is a hypermedia type; XML is not.

Furthermore, he defines a classification scheme called H Factor as “a measurement of the level of hypermedia support and sophistication of a media-type.” The REST & WOA Wiki defines “the Hypermedia Scale” [5], where the categorization is based on capabilities to do CRUD (Create, Read, Update, Delete) operations. Considered on their own, RDF serializations are hypermedia types, but only at the LO (outbound links) and CL (control data for links) levels (see [3] for details on this notation), and just an R Type (read) on the Hypermedia Scale. We aim at improving this situation.

Amundsen argues in a blog post [2] that the Semantic Web community should not pursue an API for RDF, but rather make RDF serializations more powerful hypermedia types. He argues that a key factor in the success of the Web is that messages not only contain data but also application control information, and that this is needed for the Web to scale.

Semantic Web services are likely to be an important source of data for future applications, but for foreseeable future they are just one of many. A key promise of the Semantic Web is the ability to integrate many data sources easily. There are two key issues, first read-write operations must be similar to how it is done with other hypermedia types, to make it possible to use generic code to minimize the effort required to support RDF. Importantly, if interacting with Linked Data
requires extensive out-of-band information, it will be harder to use than media types that do not. Secondly, it requires relevant links and that the resulting graph can be traversed by reasonable means.

While links are abundant across the Linked Open Data (LOD) cloud, this paper will argue that key links are missing to make it possible to traverse the resulting graph and to enable read-write operations. Moreover, we will argue that this shortcoming is due to that the community does not adequately take into account the constraint known as “Hypermedia as the Engine of Application State” (abbreviated HATEOAS) from the REST architectural style, see [6] Chapter 5.

In a read-only situation, the HATEOAS constraint requires that the application can navigate from one resource to another using the hypermedia links in the present resource only, they should not require any out-of-band information. Since RDF is built on URIs, many of which can be dereferenced to obtain further links, it will therefore almost always satisfy the HATEOAS constraint in the read-only case.

Contrast the read-write situation: It is very common that specifications that advertise themselves as RESTful has an API that is specified in terms of URI structure as well as HTTP verbs. This is not a violation of the HATEOAS constraint per se, but it would usually be superfluous as the same information must be available in a hypermedia message in order to satisfy the HATEOAS constraint. The constraint does not require this information to be available in all messages, and so, it is not a very strict constraint.

It is debatable whether a protocol can be said to be RESTful if the links don’t enable any useful interactions. Even in the read-only case, we must carefully add links that enable the desired interactions to take place, e.g. linking a SPARQL endpoint. Importantly, if there is to be such a thing as a RESTful read-write Semantic Web protocol, there must be something in the RDF itself that can be used by practical applications to write data. Therefore, whether the HATEOAS constraint is satisfied must be judged on the basis of the practical applications the protocol enables.

2 Required links

We note that the Atom Publishing Protocol [7] has a single service endpoint. From there, you can navigate to what you need. This motivates the first discussion topic of this paper:

Question 1. Is a single service endpoint enough for Linked Data?

We note that the distributed graph structure of the LOD Cloud makes this awkward: For every new data source encountered in a graph traversal, a new service endpoint must be queried. Moreover, it would be awkward if fine-grained access controls for writing are being used to record permissions for all resources in a single service description. It seems likely that a few triples attached to each information resource are better suited in most cases.
3 Defining hypermedia RDF

We noted that for a read-write RESTful protocol, we need to say in the RDF message itself what kind of operations can be made. We also note that only information resources can be manipulated, and we argued that it should be possible to add the required triples to every resource. We should be able to define hypermedia RDF in terms of a minimal vocabulary, but like the H Factor web page, we will find it instructive to use examples. In the following, we use Turtle syntax, with prefixes omitted for brevity. Also, we note that in many cases, we are making statements about the current resource, which given a reasonable base URI can be written as <>. We have not found the following factors to be relevant to RDF: LE (embedded links), CR, CU (control data for read and update requests respectively) and LT (templated queries); and LO and CL are trivially supported. The remaining are:

LN Support for non-idempotent updates (HTTP POST)

<> hm:canBe hm:mergedInto ;
hm:createSimilarAt <../> .

The first triple says that the current resource can be merged with another resource. In a typical HTTP case, this would be achieved by POSTing to the current resource with an RDF payload, and the server would perform a RDF merge of the payload with the current resource. This term is motivated by that POST operations in [10] are specified to result in an RDF Merge of the payload into the named graph. Amongst other possibilities is a union of graphs.

The second triple exists to make it possible to POST an RDF payload to some URI and the server will itself assign a URI to the posted RDF payload. This may for example be a named graph, as defined in [10] or a follow the protocol recently suggested in [9] Section 5.4.

LI Support for idempotent updates (HTTP PUT, DELETE)

<> hm:canBe hm:replaced, hm:deleted .

The first triple says that the current resource may be replaced by PUTting an RDF payload to the resource URI. The second triple says that the resource may be deleted, typically by a HTTP DELETE on the resource URI.

CM Support for indicating the interface method for requests (e.g. HTTP GET, POST, PUT, DELETE methods).

For example (with similar triples for other HTTP methods):

hm:replaced hm:httpMethod "PUT" .

We see that with a simple vocabulary, RDF serializations can become a very powerful hypermedia types. We also note that this vocabulary enables agents to do the same operations as the SPARQL 1.1 Graph Store HTTP Protocol [10], while being fully RESTful. The Protocol specification is currently not RESTful
per the discussion above as it requires extensive out-of-band information, even though it was a key design goal. It should also be possible to use this on the default (nameless) graph by assigning it a name in the service description rather than defining it in an out-of-band specification like is currently being done.

Question 2. Should the SPARQL 1.1 Graph Store HTTP Protocol be replaced by a vocabulary like the above?

3.1 New H Factors

Mike Amundsen solicits feedback on the completeness of his classification scheme. The following are suggested as discussion items:

   The self-describing nature of RDF is an important characteristic of the model and arguably an important characteristic of RDF as hypermedia as agents may be able to infer possible interactions based on vocabulary definitions.

2. Support for supplying control data relevant to subsequent requests.
   It may be useful for agents to know e.g. what formats they can send to a given resource before the request is made. This may require a new control data factor. An example of this use may be e.g.:
   
   <> hm:acceptsFormat <http://www.w3.org/ns/formats/Turtle> .

   RaUL goes beyond HTTP GET queries, which is what the LT factor is about. The interaction RaUL enables are quite different from those discussed in this paper and are not captured by any of the current factors.

Question 3. Can we create a better vocabulary for hypermedia RDF than the sketch above?

4 Bridging LOD and SPARQL

In many cases, it is sufficient to get a small number of resources to collect the data needed for a given usage, but slightly more involved queries (e.g. “what kind of connections exists between Kate Bush, Roy Harper and bands that have sold more than 200 million albums?”) would likely cause thousands of resources to be downloaded and examined. For this, more advanced graph pattern matching, as well as more advanced mechanisms for source selection, is required. [11] provides some techniques that should prove very valuable in this respect and so it becomes important that SPARQL can be used with Linked Data in a RESTful manner.

In [4] Tim Berners-Lee notes “To make the data be effectively linked, someone who only has the URI of something must be able to find their way the SPARQL endpoint.”, but this is generally not possible today without requiring out-of-band information.

The triples needed to do this are already defined in VoID (see e.g. [1]) and are in use:

5 Conclusion

We have shown how RDF serializations can become very powerful hypermedia types by using a simple vocabulary. As a consequence, developers can use Linked Data in a read-write scenario without specialized knowledge about RDF APIs or other out-of-band information. We have argued briefly that some triples should be added to every information resource, but note that most triples are only relevant to write-operations and should only appear in that case. We also noted that SPARQL is important for non-trivial traversal of Linked Data. To sum up, the addition of the following triples would make life easier for developers of applications that use Semantic Web data:

```xml
<> hm:canBe hm:mergedInto, hm:replaced, hm:deleted ;
  hm:createSimilarAt <../> ;
  hm:acceptsFormat <http://www.w3.org/ns/formats/Turtle> ;
```

References

Introducing Statistical Design of Experiments to SPARQL Endpoint Evaluation

Kjetil Kjernsmo\textsuperscript{1} and John S. Tyssedal\textsuperscript{2}

\textsuperscript{1} Department of Informatics, Postboks 1080 Blindern, N-0316 Oslo, Norway 
kjekje@ifi.uio.no

\textsuperscript{2} Department of Mathematical Sciences, Norwegian University of Science and Technology, N-7491 Trondheim, Norway john.tyssedal@math.ntnu.no

Abstract. This paper argues that the common practice of benchmarking is inadequate as a scientific evaluation methodology. It further attempts to introduce the empirical tradition of the physical sciences by using techniques from Statistical Design of Experiments applied to the example of SPARQL endpoint performance evaluation. It does so by studying full as well as fractional factorial experiments designed to evaluate an assertion that some change introduced in a system has improved performance. This paper does not present a finished experimental design, rather its main focus is didactical, to shift the focus of the community away from benchmarking towards higher scientific rigor.

1 Introduction

The practice of benchmarking is widespread, both in industry to select the most appropriate implementation among several choices, and in academia as means to verify or refute an assertion on the performance of a certain practice or system. While we could contribute to the industrial cases, our focus is on the latter. A benchmark typically consists of several microbenchmarks, which are procedures for placing loads on the system to measure its performance as a function of the load. For SPARQL benchmarks, usually the performance is measured as the number of queries the SPARQL implementation is able to answer in a certain time (the throughput) or the time required for it to respond to a query.

However, there are many problems with this approach when used to evaluate assertions about the overall performance of a system or several systems:

- As the complexity increases, the number of parameters that one may wish to test increases dramatically.
- Nothing compels anyone to find microbenchmarks that can refute an assertion, and there is no structured approach to investigate whether a benchmark is flawed.
- There is no meaningful summary of the performance as a whole, so that an assertion about performance cannot be refuted as an optimization may have detrimental side effects that cannot be identified, or cannot be compared to possibly positive effects in a different microbenchmark.
To address some of these problems, one may try to standardize benchmarks, to eliminate some parameters and identify others as believed to be important. Also, social pressure should then prompt developers to use a standard benchmark so that assertions may be refuted.

While this is an improvement which is possibly sufficient for industrial uses, it is still flawed as empirical research, as it cannot capture the full diversity of the problem: For example, if a vendor claims that their implementation is better if used on a highly capable hardware platform, then the standardization of a hardware platform cannot test this assertion. Moreover, to test this assertion, one would need a summary technique that can identify the interaction between implementation and hardware platform and possibly other parameters.

One important aspect of science is the structured approach to falsifying hypotheses, and therefore is important that assertions about performance are formulated as hypotheses that can be falsified.

Therefore, attacking the complexity problem by eliminating parameters we think influence the outcome (e.g. caching) for simplicity is untenable as scientific practice. We must instead tackle this problem by enhancing our capability of testing complex systems and to identify situations where our assumptions are flawed. Such constraints can only be made when we are highly confident that a certain parameter does not have any influence.

In this paper, we employ techniques from the well-established field in statistics known as Design of Experiments (DoE) to point out a direction that promises to address these problems. First, we design a simple 8-parameter experiment, and then we detail an experiment that can meaningfully provide a summary of the statistics.

Then, we demonstrate how the number of runs required in the experiment can be reduced while maintaining a structured approach to the problem, and we discuss the trade-offs involved. We then continue to discuss techniques that show our simplistic experiment to be flawed, yet scientifically better than the practice of benchmarking. Finally, we outline a road-map to establish the use of these techniques in SPARQL endpoint evaluations both as a practical tool and as a scientifically more rigorous methodology.

1.1 Key concepts of Design of Experiments

We expect DoE to be new to most readers, but some familiarity with hypothesis testing is assumed.

In the experiments DoE is concerned with, a response variable is measured under a various combinations of parameters. These parameters are known as factors. For each factor, a range of possible values is fixed. These values are known as levels for this factor. Levels are not constrained to be continuous variables, they can be discrete, or even two different instances of a class. Experiments are run by choosing a combination of levels for the factors.

For example, one may measure the execution time of a SPARQL endpoint by measuring it when it contains 1 or 2 million triples. Then, the execution time is the response variable, the factor is the number of triples and it has two levels, 1
or 2 million triples. We have chosen in this paper to only deal with the relatively straightforward aspects of the DoE formalism, and therefore constrained the experiment strictly to two levels.

In an experiment, there are several factors, and one may run the experiment by measuring the response variable for every combination of levels. In a two-level experiment, this will result in $2^n$ runs, where $n$ is the number of factors. This is called a full factorial experiment. This is done in Section 3.2.

In many fields of science, experimental economy is extremely important, and so, DoE offers extensive methodology to run a certain fraction of the runs at the price of explanatory power. Such experiments are called fractional factorial experiments and are covered in Section 3.3.

We describe the influence of the choice of levels on the response in terms of effects. For a factor $A$ with two levels, we let $a_1$ and $a_2$ be the average response of all $2^n - 1$ measurements with $A$ at level 1, and 2, respectively. The main effect of $A$ is then defined as $a_2 - a_1$.

Similarly, interaction effects for two factors $AB$ are defined by comparing averages for equal versus non-equal levels of $A$ and $B$. Details of this theory, and how to compute the effects in practice, using linear regression, are found in [14].

The next step is to understand which factors are important or significant. One approach is to plot the sorted effects against the normal distribution. To understand why, note that if there is nothing of interest, the measurements have a certain normal distribution purely due to noise. When plotted against a normal distribution, the plot would be a straight line. If there are any deviations from the straight line, it implies that they may be significant. Such plots can be found in Figs. 1 to 4. The range in the $y$-axis is dictated by the number of runs.

2 Related work

A literature survey has not revealed any direct prior art, neither in the Semantic Web field nor more generally in database research. However, the general approach has been well established, not only in statistics. A relatively well cited textbook is [8] but we have not found the parts discussed in the present paper to be widely adopted. Recently, a comprehensive text on experimental methods has been published in [1] demonstrating the broad applicability of the methodology we employ. We have chosen SPARQL Endpoint evaluation as an example in this study since SPARQL Federation is our main interest, and to turn to statistical standard texts [14] for this study.

Some problematic sides of benchmarking have been noted in several papers, notably [2] and [11]. We also acknowledge that great progress has been made to improve benchmarks, in particular, we are indebted to [12].

We believe that automatic creation of benchmark queries, as pioneered by [5] is a critical ingredient for the application of DoE to be successful as a methodology.
Finally, we acknowledge the efforts of the Linked Data Benchmark Council and the SEALS project. However, they appear to focus on cases where the workload is assumed to be well characterized, rather than scientific evaluations.

3 Experiments

Common experiments include the comparison of several different implementations. However, we focus on a slightly different problem: We would like to compare the same implementation before and after some change has been made, typically with the intention to enhance the performance of the system. There are two reasons for this: One is that there are currently many SPARQL implementations available, also Free Software ones that can be modified by anyone, and the other is to constrain the scope of the study to comparing just two different things.

3.1 Setup

As the focus of this paper is didactical, the actual measurements are much simpler than those that have been used in benchmarking. We have used the data-set of DBPedia SPARQL Benchmark [12], but only consider the smallest of their data-sets (that we found to be more than 15 MTriples). Moreover, we have taken subsets of that data-set by using the last 1 or 2 MTriples from their file.

We have chosen 8 factors each having 2 levels, “TripleC” the number of triples in the data-set, 1 or 2 MTriples, “Machine”, which is the software and hardware platform, one larger with slower disks

3 Running GNU/Linux Debian Squeeze, has 16 GB RAM, an Intel Core2 Duo E8500 CPU and two Parallel-ATA disks in a RAID-1 configuration

4 Running Debian Wheezy, has 8 GB RAM, an Intel Core i7-3520M CPU and a single SSD on SATA-III.

The following factors test the absence (at level 1) or presence (at level 2) of the following clauses:

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
</tr>
</thead>
</table>

The following factors test the absence (at level 1) or presence (at level 2) of the following clauses:

3 Running GNU/Linux Debian Squeeze, has 16 GB RAM, an Intel Core2 Duo E8500 CPU and two Parallel-ATA disks in a RAID-1 configuration

4 Running Debian Wheezy, has 8 GB RAM, an Intel Core i7-3520M CPU and a single SSD on SATA-III.
These fragments have been carefully designed for illustrative utility, as well as suitable selectivity. They are not themselves important, they only serve to illustrate certain examples of possible factors. When the experiment is run, they are combined to yield a query, which is then sent to a SPARQL endpoint. The SPARQL endpoint is itself set up using 4store (see [6]) version 1.1.5.

Finally, “Implement” is the implementation undergoing evaluation. Level 1 is running the new implementation, whereas Level 2 is the old implementation. The null hypothesis $H_0$ is that the old implementation is as good as the new, and the alternative hypothesis $H_1$ is that the new implementation has overall improved the performance of the system.

A real optimization is beyond the scope of this paper. Instead we have performed a simulation that enables us to understand the effect of the changes. In a real-world case, it is often the case that an optimization has negative side-effects, and we would need to simulate both the optimization and the side-effects. To do that, we degraded the performance of the 4store SPARQL implementation by inserting `sleep` statements. Specifically we inserted the C statement `sleep(2)` on line 920 in `src/frontend/query.c` on level 1, to simulate the optimization. This has the effect of delaying execution for 2 seconds for every block for all kinds of joins. On level 2, we inserted `usleep(2000)` on line 987 in `src/frontend/filter.c` to simulate the negative side-effect. This delays execution for 2 milliseconds every time the `langMatches` SPARQL function is called.

The experimentation is implemented in R [13], which is a free software environment for statistical computing and graphics. Necessary tools for DoE has been implemented by the R community in packages called DoE.base [3] and FrF2 [4]. The experiments are run on a third computer in a 1 gigabit Ethernet LAN with the two experiment hosts. On the experiment hosts, four 4store instances runs sequentially and independently on different ports, where processes that are not actively answering a query are idle. Practically, the “Machine” factor specifies a hostname, whereas “TripleC” and “Implement” specifies a port number.

As response variable, we have chosen to use the time from the namespace server lookup finishes to the data has been transferred. This choice is somewhat arbitrary, many other response variables could be chosen, and indeed, future work should explore multi-variable responses, but for simplicity, we think this
reflects the total amount of work done by the SPARQL endpoint well. For the measurements, we have chosen to use the RCurl package [9], which is a thin wrapper around the curl library. Curl has well developed facilities for timing requests and responses, and so we rely on its measurements.

Finally, note that there are two common issues in benchmarking we do not consider: we do not allow a warm-up run, nor do we take into account that the server may cache all or parts of the result set. The reasons for this choice will be discussed later.

All the code to reproduce this study as well as detailed instructions have been published on Github: https://github.com/kjetilk/doe-sparql.

3.2 Full factorial experiment

In the full factorial experiment, all combinations of the 8 factors are executed, in total 256 runs. This is called a $2^8$ factorial experiment, and while it quickly becomes infeasible for many factors, we shall nevertheless find the following small example instructive. Note that each combination is executed only once (i.e. it is unreplicated), however, so even a full factorial experiment compares well to a typical benchmark in which each microbenchmark must be executed several times to obtain enough data to compute a mean with reasonable certainty.

The above experimental setup is executed across a design matrix generated with the DoE.base package, which returns a data frame consisting of the factors with a column containing the corresponding curl measurements. With that, we do as described in the introduction and fit a linear model with all factors. Then, we generate a normal plot, see Figure 1. Any significant departure from a straight line can be interpreted as a significant effect (see Section 4.8 of [14] for a detailed explanation). In our case, it is most important to note that any effect that is negative means the effect enhances the performance of the SPARQL endpoint, the runtime decreases. If “Implement” and its interactions are negative conditional on that the factors involved with its interactions are on their high level, then it supports the alternative hypothesis, i.e. we have successfully improved the performance.

To proceed beyond a visual judgment of significance, we may use Lenth’s method [10], which is a simple method for estimating significance in an experiment with no replicates. In Table 1, the most important effects are listed. In Figure 1, Lenth’s method is used to label the significant effects. We see that we have many very significant effects (unfortunately so many that some labels are illegible). We also note that “Implement” itself is significantly negative and that the “Implement:Lang” interaction is significantly positive, as expected from our setup, where “Lang” was the detrimental side-effect we simulated.

To investigate whether the detrimental side-effects cancel the positive effect of our simulated optimization, we have to formulate a hypothesis test. By inspecting the normal plot in Figure 1 and Table 1, we see that six factors are highly significant by either being a significant main effect or participate in a highly significant interaction. We say that the two factors “Range” and “Machine” are inactive since they do not contribute significantly to the observed variation in
In this context, we call “Implement” a control factor since we can only control “Implement”, i.e. we can only change the implementation. Of information, or change the query, since it gives a different answer. In practice, we can only control “Implement”, i.e. we can only change the implementation. In this context, we call “Implement” a control factor and the other five active factors environmental.

We would like an overall test to see whether the new implementation is better than the old, and the presence of inactive factors makes it possible to average the performance of the new and the old implementations into two distinct vectors. We do this by creating an average over all the 32 level-combinations of the five active environmental factors. This leaves us with 4 values for each of the two levels of “Implement” and a two-sample t-test can be performed. Effectively, we treat our experiment as a $2^6$ experiment replicated 4 times.

Fig. 1. A normal plot of the Full Factorial Experiment. The labelled points are considered significant by using the Lenth criterion at a level $\alpha = 0.05$. 

... and a two-sample t-test can be performed. Effectively, we treat our experiment as a $2^6$ experiment replicated 4 times.
Table 1. The magnitude of effects for some important main effects and interactions.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement</td>
<td>-21.27</td>
</tr>
<tr>
<td>Implement:Optional</td>
<td>-11.49</td>
</tr>
<tr>
<td>Implement:Union</td>
<td>-6.21</td>
</tr>
<tr>
<td>Implement:TripleC:Lang1</td>
<td>4.12</td>
</tr>
<tr>
<td>TripleC:Lang</td>
<td>4.20</td>
</tr>
<tr>
<td>Implement:TripleC</td>
<td>4.32</td>
</tr>
<tr>
<td>Implement:BGPComp</td>
<td>4.89</td>
</tr>
<tr>
<td>Lang:Union</td>
<td>5.16</td>
</tr>
<tr>
<td>Implement:BGPComp:Lang</td>
<td>5.16</td>
</tr>
<tr>
<td>BGPComp:Lang</td>
<td>5.25</td>
</tr>
<tr>
<td>Implement:Lang:Union</td>
<td>5.75</td>
</tr>
<tr>
<td>TripleC</td>
<td>7.06</td>
</tr>
<tr>
<td>Implement:Lang</td>
<td>7.73</td>
</tr>
<tr>
<td>Lang</td>
<td>8.62</td>
</tr>
<tr>
<td>Optional</td>
<td>11.30</td>
</tr>
<tr>
<td>Union</td>
<td>17.18</td>
</tr>
</tbody>
</table>

We perform the test as a one-sided hypothesis test where $H_0$ is that the mean of the vectors are equal and $H_1$ is that mean sof the vectors representing the new implementation is lower. In this example, we find that the available data supports the assertion that the new implementation is better with a high probability, $p = 1.16 \cdot 10^{-07}$.

While it is interesting to test the overall hypothesis that the new implementation is an improvement, it is also interesting to know if there are cases where it fails. Again, we treat the experiment as a $2^6$ experiment replicated 4 times. Now, for each of the 32 resulting combinations of the environmental factors for each level of “Implement”, we extract 4 values. We may now run a t-test for each of the replications, as above. The results are tabulated in Table 2. We see that there are significant improvements in most cases; one may want to further investigate the cases where they do not.

For future larger experiments, one must investigate whether it is necessary to adjust the $p$-value due to the larger numbers of hypotheses tested.

The main objective of this paper is not to establish that the optimization is significant, it is merely a simulation. It is to establish a critical practice of evaluations. We must therefore understand why some effects are significant. That “Implement” is significant is hardly a surprise, that is what we worked for. That the main effects “Union”, “Optional” and “Lang” are strong is also due to that these are fairly demanding things to evaluate. Since the delay we inserted affects all kinds of joins, it is also intuitive that interactions between “Implement” and those that require joins are strong. The strong “BGPComp:Lang” and “Lang:Union” interactions might be due to that many more triples need to be searched for a language tag in one of the levels of the interacting factors. However, the positive “Implement:TripleC” interaction evades such explanations; in fact, it hints that our optimization may not work as well for larger databases. By carefully inspecting Table 2, we see that whenever “TripleC” is at the low level, the new implementation is always best. Only at the higher level, the old implementation may be better. This is a clear indication that the separation between levels for the sizes of the data-set is too small and should be investigated further. Such reasoning should be applied to all significant effects.
Table 2. $p$-values for different parts of the experiment

<table>
<thead>
<tr>
<th>“TripleC”</th>
<th>“BGPComp”</th>
<th>“Lang”</th>
<th>“Union”</th>
<th>“Optional”</th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.012</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>$3.1 \cdot 10^{-09}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>$6.1 \cdot 10^{-11}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>$3.3 \cdot 10^{-06}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>$2.7 \cdot 10^{-09}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$2 \cdot 10^{-06}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>$3.6 \cdot 10^{-10}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.014</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>$1.2 \cdot 10^{-10}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>$2.8 \cdot 10^{-14}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>$4.1 \cdot 10^{-15}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>$2.1 \cdot 10^{-05}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>$2.7 \cdot 10^{-07}$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.0072</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>$1.6 \cdot 10^{-05}$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.28</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>$3 \cdot 10^{-07}$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>$3.3 \cdot 10^{-07}$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$1.7 \cdot 10^{-08}$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.0023</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>$6.5 \cdot 10^{-07}$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.00032</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>$1.3 \cdot 10^{-06}$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.013</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>$1 \cdot 10^{-11}$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>$3.8 \cdot 10^{-11}$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>$4.1 \cdot 10^{-15}$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>$2.3 \cdot 10^{-05}$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.99</td>
</tr>
</tbody>
</table>

3.3 Fractional factorial experiment

As mentioned, the full factorial experiment goes as $2^n$ and becomes prohibitively expensive when $n$ is a large number of factors. As the evaluation of SPARQL endpoints is an inherently complex problem, a large number of factors are needed, and so full experiments cannot scale. However, we may reduce the size of the experiment significantly by sacrificing some explanatory power, in the form of fractional factorial experiments.

We lose explanatory power due to aliasing; for example, the interaction labelled “TripleC:BGPComp” may be aliased with the “Machine:Range” interaction. That is to say, a detected increase in run time from larger Basic Graph
Patterns for large databases, can also be explained by a less powerful machine that is worse at evaluating FILTER clauses with ranges. They are indistinguishable. This may or may not cause a problem. In many cases, we may not be interested in these effects, we may only be interested in “Implement” and its interactions. Moreover, it is possible (even easy using the FrF2 package in R) to declare which effects must not be aliased, and determine the size of the experiment based on that. Such a main effect or two-factor interaction is called clear if none of its aliases are other main effects or two-factor interactions. Another possibility to shrink the size of the experiment is if we a priori can say that some effects are negligible. We have not found such assumptions to be tenable in our case.

We have made two fractional factorial designs: One with 32 runs and one with 64 runs. Again, the number of runs is in powers of 2, but much smaller. In both cases, we have specified that all the factors must be clear and also all two-factor interactions where “Implement” is involved must be clear. The resulting experiments is returned to us as design matrices, but they can also be described sufficiently for reproducibility in terms of design generators. These also declare aliasing relations for some of the main effects. The design generators are

- “Range” = “TripleC” “Machine” “BGPCmp”
- “Union” = “TripleC” “Machine” “Lang”
- “Optional” = “TripleC” “BGPCmp” “Lang” “Implement”

and

- “Union” = “Implement” “TripleC” “Machine” “BGPCmp”
- “Optional” = “Implement” “TripleC” “Lang” “Range”

for the 32 and 64 run experiments respectively.

![Normal plot](image)

**Fig. 2.** A normal plot of the Fractional Factorial Experiment with 32 runs. The labelled points are considered significant by using the Lenth criterion at a level $\alpha = 0.05$.

The resulting normal plot is in Figure 2. We see that only two main effects are significant on a $\alpha = 0.05$ level according to the Lenth criterion, “Implement” is again deemed a significant improvement, and “Union” is deemed significantly
the hardest operation for the query engine. We see that some other points also
deveiate from the straight line, but with as few runs as this, the total variance
is great, so no other conclusions can be drawn. We may use the t-test we used
in the previous section, as we have only two active factors and so we may treat
it as a $2^2$ experiment with 8 replications. The resulting $p = 0.00098$ is still very
low, albeit larger than in the previous experiment. This also explains why there
are so few significant effects, and had to be expected with such a small number
of runs and thus higher variance.

For any other purpose than a rough idea of some key influences on the over-
all performance of the endpoint for the given levels, this little experiment is
insufficient.

We turn to the 64-run experiment and its normal plot in Figure 3. We see that
then we get more significant effects, and that they correspond well to those we
saw in the full factorial experiment. They also all have an intuitive explanation
as above. However, we do not see the effects that we found worrisome in the full
factorial design. The “Implement:TripleC” interaction emerges only for a level
$\alpha = 0.15$. To ensure that we discover such cases will be a key challenge in further
studies.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{A normal plot of the Fractional Factorial Experiment with 64 runs. The labelled
points are considered significant by using the Lenth criterion. Triangles correspond to
a level $\alpha = 0.15$ and crosses at $\alpha = 0.05$. The use of different symbols has been added
to FrF2 by the authors.}
\end{figure}
Since the 64-run experiment has 3 inactive factors, “Machine”, “TripleC” and “Range”, we can treat it as a $2^5$ experiment replicated twice. Using the same procedure as above, we find that again, the hypothesis test indicates that the new implementation is an improvement at $p = 5.9 \cdot 10^{-6}$. Again, the comparatively higher $p$-value is due to the lower number of runs. It is also possible to perform the per-environmental factors test.

### 3.4 Fractional factorial experiment with more data

The importance of finding flaws in the levels is highlighted by the normal plot in Figure 4. The smaller experiment made it feasible to run with a larger data-set, this experiment uses the original 15 MTriples data-set of [12]. The picture is now as hinted in the previous discussion, for larger “TripleC”, our simulated performance is actually worse, as evident from the positive “Implement”. We also see that the interactions “Implement:Lang” and “Implement:TripleC” are highly significant. It is clear what happened: For larger “TripleC”, the delay we introduced in the langMatches function has a devastating effect, since it delays the execution for every triple that the query engine applies the filter to, and thereby completely dominates the execution time.

Moreover, we note that the “Machine” factor has become highly significant in the other end along with its “Machine:TripleC” interaction. One should take careful notice that the “Machine” factor encompasses many different things, that should be different factors; CPU, disks, available RAM, installed software, tuning parameters and so on. It is advantageous to have such broad factors in the design in early experiments, because they may point out unexpected effects that should be further scrutinized. The “Machine” factor is a good example of usage that also helps us manage complexity: If possible, one may group many factors into one if in doubt of their relevance and break it up in subsequent experiments if deeper understanding is needed. To some extent, “BGPComp” is also a broad factor, since it encompasses many structures that are normally seen in a Basic Graph Pattern. In this case, the cause of the performance problem was highly audible: It was the old P-ATA disks of the larger machine, but in general, such clues are not available to the experimenter, therefore more factors are needed.

We are forced to conclude that the three preceding experiments are severely flawed, as they failed to take into account the effects from larger data sizes on slow hardware.

Note that we do not take advantage of the usual performance of 4store in this simulation, to the contrary, we chose 4store because of its relative simplicity to modify it to suit our needs for illustrative purposes, which consists of degrading it in several ways. The small data-sets used here do not imply that the approach will not work for large data-sets; properly configured, actual optimization will give very different runtimes, and the small number of runs will make it possible to evaluate a very large number of factors.
Fig. 4. A normal plot of the Fractional Factorial Experiment with 64 runs where the level 2 of “TripleC” is 15 MTriples. The labelled points are considered significant by using the Lenth criterion at a level $\alpha = 0.05$.

4 Discussion

It is a fundamental property of two-level experiments that they can only model linear effects, however, they are usually quite sufficient for identifying significant effects even though the underlying response may be non-linear. As we saw in Section 3.2, the choice of levels may be critical, and if non-linearity is expected, more than two levels must be considered, or at the very least, attention must be paid to the choice of levels.

We have argued that fixing certain parameters is untenable, but we have allowed ourselves to disregard caching and warm-up runs. We are aware that caching may be important in 4store, as a preliminary experiment gave a response time of 53.9 s of a query with “BGPComp” at level 2 but other factors at level 1. When an OPTIONAL clause was added, the response time dropped to 1.25 s. As this is not a more restrictive query, the most likely explanation is that the
result of the evaluation of the Basic Graph Pattern was cached, so only the OPTIONAL left join was necessary to add in the second query.

We did this to illustrate randomization as this is what makes it permissible, i.e. the order of execution is random, so the benefit of caching is likely to apply randomly to different runs. For example, had the run order been reversed in the previous example, it could have been the query without OPTIONAL that would have benefited. Unless the effect of caching is cumulative throughout the experiment (which is possible), the randomization will have the effect of reversing the runs randomly. The end result is that the effect of so-called “lurking variables” such as caching, warm-up effects, etc, contribute to the total unexplained variance of the experiment, but should not skew the results to invalidate the estimated effects.

As the unexplained variance of the experiment may cloud important effects, notably effects that indicate flaws, it should be kept to a minimum. Thus, lurking variables should be turned into factors whenever possible, as they degrade the quality of the experiment, but in many cases, we can live with them if their contribution to the variance is small.

Finally, we note that the most problematic cases are those that are covered neither by the broad factors, nor the lurking variables, or any of the specific factors. We cannot in the general case be sure that they are accounted for, so reviewers must remain vigilant that some factors are simply ignored. What finally invalidates the present experiment is the complete absence of certain obvious language features, such as solution modifiers.

5 Future Work

The present experiment is very primitive in its choice of factors, and we believe that the strategy employed to construct suitable experiments will be the main determinant for the long-term feasibility of this direction. One strategy will be to start with broad factors such as “Machine” and refine them with experience. Also, the data must be parameterized with many more factors than just the number of triples; data heterogeneity [2] is one example, but also queries over highly skewed data is important. A more difficult issue is how to address the problem of testing the whole SPARQL language [7]. We believe that the parameterization work initiated by [5] with their SPLODGE system is a key ingredient. They parameterized SPARQL queries in such a way they could be auto-generated. One suggestion is to parameterize queries based on the grammar, so that one factor becomes the number of e.g. UNIONs, and the levels are chosen. One problem one will soon run into is that parts of the language is recursive, e.g. a GroupGraphPattern can consist of several GroupGraphPatterns. However, we believe that pragmatic limits can be set, it is for example not of practical interest to nest OPTIONALs to any great depth even if it is allowed by the language.

We saw in Section 3.3 that with a smaller experiment, fewer interactions are significant, and so provide us with fewer clues to assess the soundness of the experiment. This should be a focus of further research.
In this paper, we have only employed the simplest parts of the DoE theory. What has been presented here is part of a much more general formalism known as Orthogonal Arrays. The use of orthogonal arrays allow for different numbers of levels, for much greater flexibility in total run size, and for non-regular designs that can provide a good fit for the complex problem of SPARQL endpoint evaluations.

Finally, as each run is cheap, and the experiment can usually run without human intervention, we believe it is interesting as a case for advancing the state of the art of DoE.

6 Conclusions

As evaluating SPARQL endpoints is inherently difficult, a simplistic experiment such as the one we designed should not hold up to scrutiny. We set out to demonstrate how an experiment could be set up using DoE, and show how it can be analyzed. In sections 3.2 and 3.3, we first saw how the analysis correctly pointed out the most important effects, under assumptions dictated by the factors and levels that were given. We saw how the formalism provided a comprehensive view of the experiment. Then, we saw that the formalism pointed out weaknesses that could invalidate the experiment, and in Section 3.4 we saw that the experiment did indeed not hold up to scrutiny.

We saw that while two-level experiments can perform well in determining significant effects, they do not necessarily work well to find a model, as exemplified by failure to identify the possibly non-linear effect of the P-ATA disks in the first experiments. This issue can be addressed by using an orthogonal array design.

Also, experiments as small as the 32-run are not useful in estimating any detailed characteristics and are thus of little use for science, but could be useful in engineering: If the experiment has been validated by a larger experiment, it could serve well in a Continuous Integration system.

We have seen that we can perform a proper hypothesis test based on summary statistics, and by using our expertise, we showed how to reveal that the experiment was flawed, and we have cautioned that failure to control unexplained variance may compromise our ability to do so. We have also pointed out how this direction can provide more rigorous evaluation practices.

To this end, the following questions must be asked:

1. Are there factors that cover all realistic features?
2. If not, are they adequately covered by randomization?
3. If so, would the variance resulting from randomization obscure factors that could provide clues that the levels are wrongly set?
4. By carefully examining interactions with “Implement”, are there any that are unaccounted for, and that could point out wrongly set levels?

Even if the complexity is great, with properly tuned endpoints, it is feasible to do millions of runs, and with orthogonal arrays, this formalism can be extended to many different evaluation problems.
Acknowledgments Kjetil Kjernsmo would like to thank his main supervisor Martin Giese for kind assistance. He would also like to thank Steve Harris for his generous support in degrading his excellent work on 4store.

References

Pushing Complexity Down the Stack

Gregory Todd Williams¹ and Kjetil Kjernsømo²

¹ greg@evilfunhouse.com
² University of Oslo, Department of Informatics, Postboks 1080 Blindern, N-0316
Oslo, Norway kjetil@kjernsømo.net

Abstract. We believe that there is great potential for performance improvements in making high-level query features visible to low-level data sources. In our work in the PerlRDF and SPARQLKit systems, we have tried to apply this thinking while maintaining flexibility and extensibility. We discuss two major approaches we have taken – growing low-level APIs to handle more complex query operations, and allowing low-level frameworks to participate in query planning – and briefly examine some benefits and challenges to their use.

1 Introduction

Commonly, APIs to triple stores have made it necessary to break down more complex (e.g. SPARQL) queries into individual triple patterns, which are then evaluated against the triple store. This causes many problems: It is hard to use any optimizations on the triple store level, to use any statistics available for index scan efficiency or join order optimization, hard to implement experimental extensions to query languages, and hard to exploit special features of triple stores. Some systems provide an alternative to using these APIs by allowing the triple store to evaluate the entire query. Between these two extremes, we propose to make high-level query features visible to low-level data sources, allowing planning- and evaluation-time customization of the query answering process. We discuss approaches we have implemented in the SPARQLKit¹ project and the new PerlRDF² Attean³ framework, and the challenges we see in using them in the future.

2 Increasing API Capabilities

In the PerlRDF system, we define data stores (either triple or quad stores) at the most fundamental level as objects implementing a method to match triple/quad patterns (along with some other utility methods). However, many of the stores we support are capable of more complex data matching operations, and are very often able to perform these operations more efficiently than the generic system

¹ https://github.com/kasei/SPARQLKit
² http://www.perlrdf.org/, also packaged and available in Debian and Ubuntu
³ https://github.com/kasei/attean
routines. For this reason, our API has been extended over time to support more and more complex matching operations. These operations include:

1. Match triple/quad pattern
   The fundamental matching operation supported by all data store implementations.

2. Match BGP pattern
   This allows stores to implement the matching of multiple triple/quad patterns at once, returning a multiset of solutions (mapping variable names to RDF terms).

3. Match BGP pattern with simple filter
   This operation extends BGP matching with support for filtering intermediate results, leveraging existing data indexing and/or knowledge of the store-internal data layout and representation.

4. Match full SPARQL query
   This allows stores to produce results to entire queries at once, capturing the full range of operations including pattern matching, graph operations, and solution modifiers.

Data store classes may indicate their conformance to one or more of these operations, allowing the query planner to choose the best available. This has allowed store implementations flexibility in choosing a balance between complexity and performance. For example, our relational database store implements APIs 1 and 2 (allowing triple pattern joins to be handled entirely within the database), while the SPARQL Protocol store providing access to a remote SPARQL endpoint implements APIs 1–4, simply forwarding full query string across the network.

This sort of pushing down of query operators can be seen in a limited form as far back as System R[1] where filtering expression “search arguments” could be attached to data access operations, allowing only data satisfying the expression to be returned. In Semantic Web systems, the OpenRDF Sesame[2] SAIL API provides a general mechanism for handling this sort of optimization with the SailConnection evaluate method. SAIL implementations that override this method are passed a query representation and must return corresponding query results. In this way, implementations may optimize queries in any way they see fit, but they will also be responsible for the evaluating the entire query, which is a situation we have set out to improve upon.

3 Delegating Query Planning

In implementing the SPARQLKit project, and in designing the next generation PerlRDF API, we take a more flexible approach by simply delegating planning decisions to the underlying data stores. To do this, we rely on the trait systems[3] (also known as roles or protocols) of the Perl/Moose and Objective-C languages to define a QueryPlanner trait as requiring the single method:

\[
\text{plan} : \text{Algebra} \mapsto \text{Option}[\text{Set}[\text{Plan}]]
\]
Any data source conforming to this trait may participate in the query planning process, as described in Algorithm 1. (Here we show the planning process for a graph store composed of discrete triple store implementations, but the same approach can be used for quad stores.)

Algorithm 1: Delegating Query Planner, dqPlanner

\begin{verbatim}
Input: graphStore, a collection mapping graph names to triple store objects
Input: graph, the active graph
Input: algebra, a SPARQL algebra expression
Output: plans, a set of query plans for executing algebra

1 if graphStore[graph] conforms to the QueryPlanner trait then
  2 p ← store[graph].plan(algebra) ;
  3 if p = Some(plans) then
  4 return prunePlans(plans)
  5
6 return BuiltInPlanner(store, graph, algebra)
\end{verbatim}

It is worth noting that due to the composability of traits, triple stores conforming to the QueryPlanner trait do not need to rely on inheritance to provide a default behavior. In traditional object-oriented systems (such as the Sesame SAIL API discussed in section 2), optimizing triple store planners are often modeled as inheriting from a system-provided base class which provide the default planning routines. In a trait system, the query planner is able to test each triple store for trait conformance and conditionally call the store’s planner, defaulting to the system planning routines. Freeing the planning classes from this unnecessary inheritance allows more flexibility in design and implementation both triple stores and optimizing planners, and shows one of the major benefits to implementors of leveraging a trait system.

A triple store that conforms to the QueryPlanner trait may authoritatively return custom query plans for any part of the query algebra it wishes. Alternatively, the triple store may decline the request for planning by returning None. For example, the photo library triple store\(^4\) optimizes matching triple patterns of the form \{ \texttt{?s a foaf:Image} \} by directly returning the set of known images (as opposed to using the more general triple pattern matching mechanism).

However, the structure of the query algebra may not be in a form the backing store can directly use. Separately, the photo library store can optimize matching of both a geographic metadata BGP such as

\{ \texttt{?image dcterm:spatial [ geo:lat ?lat ; geo:long ?long ]} \} and a depiction BGP such as \{ \texttt{?image foaf:depicts ?person} \}. The store cannot directly optimize a BGP query combining all of these triple patterns:

\(^4\) https://github.com/kasei/GTWApertureTripleStore
To produce an optimized plan for this BGP the store must synthesize one by joining an optimized sub-plan with a system-generated plan for the remaining triple pattern(s). In this case, the store might choose to first produce an optimized geographic sub-plan, and request the system generate a plan for the remaining depiction triple pattern. The system planner will immediately try to determine if the store can also optimize the depiction triple pattern (which it can in this case). A representation of the final resulting plan would be:

```sparql
Join(
    PhotoStore_GeographicQuery(?image, ?lat, ?long),
    PhotoStore_DepictionQuery(?image, ?person)
)
```

Furthermore, our approach would simplify extensions to SPARQL greatly. For example, all optimizations done in e.g. the stSPARQL extension [4] could be implemented and encapsulated in a trait, and the rest of the query evaluation could be left to the default implementation, which would not have to be modified.

Information integration systems such as Garlic[5, 6] and HERMES[7] have previously explored this sort of pushing down complexity of query planning (and cost estimation) to heterogeneous data providers. We draw heavily on this work, and find that it pairs naturally and to great effect with a trait system. The generality of the RDF data model, extensibility of SPARQL, and flexibility of traits allow a wide range of data sources and query features to be captured by a trait-based planning system. As a result, our systems can be widely applicable without requiring data source schema modeling and other bookkeeping tasks often required by information integration systems.

## 4 Challenges

We see several challenges to using the approaches described above in a well-designed SPARQL system. The accretion of more and more complex API methods (as described in section 2) over nearly a decade is clearly not sustainable. While there are historical architectural reasons why we did not implement a fully general system like Sesame SAIL’s `evaluate`, we believe such generality is the correct approach. However, we see value in performing this sort of store-specific optimization at planning time (as with our delegating planner) rather than during query execution. This provides flexibility to the planning system, allowing further planning, rewriting, and optimization to occur after the custom plan is produced.

The delegating planner method is still somewhat brittle with respect to the exact structure of the query algebra. Specific queries may not be as easy to decompose in a planning store as the synthesized join example in section 3. For
example, even for a store that can produce optimized plans for certain filters and triple patterns, planning may not succeed if unrelated operations appear between them:

\[
\text{Filter}(\?o > 10, \text{Extend}(\?z ← \?o + 1, \text{BGP}(\?s \?p \?o)))
\]

If the triple store can only optimize BGPs with an optional enclosing filter, this query would not be optimized even though it is semantically equivalent to one in which the filter and BGP operations are adjacent (and therefore available for optimization):

\[
\text{Extend}(\?z ← \?o + 1, \text{Filter}(\?o > 10, \text{BGP}(\?s \?p \?o)))
\]

Without relying on the query planner to exhaustively test equivalent query plans, a more flexible system is needed to allow recognizing query structures that are well-suited for store-specific optimization.

Finally, custom query plans produced by stores should ideally work with the query planner’s cost model to allow the system to compare the relative costs of otherwise opaque custom plans. We don’t currently have a good system for allowing this, and instead trust that store-produced plans will always beat system-produced ones (and that all equivalent store-produced plans are equally efficient). In the long run, we believe having an extensible cost model (likely by requiring that query plan implementations conform to an \texttt{Auditable} trait, allowing cost information to be accessed) is important for a system that allows custom query plans.

\textbf{Acknowledgements} We thank Toby Inkster, Chris Prather, and Shawn M. Moore for their assistance in applying trait-based programming techniques to the design of the next generation PerlRDF system (and indirectly the SPARQLKit project).

\textbf{References}

5. Roth, M.T., Schwarz, P.M.: Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In: VLDB. (1997) 25–29


A survey of HTTP caching implementations on the open Semantic Web

Kjetil Kjernsmo

Department of Informatics, Postboks 1080 Blindern, N-0316 Oslo, Norway
kjetil@kjernsmo.net

Abstract  Scalability of the data access architecture in the Semantic Web is dependent on the establishment of caching mechanisms to take the load off of servers. Unfortunately, there is a chicken and egg problem here: Research, implementation, and evaluation of caching infrastructure is uninteresting as long as data providers do not publish relevant metadata. And publishing metadata is useless as long as there is no infrastructure that uses it.

We show by means of a survey of live RDF data sources that caching metadata is prevalent enough already to be used in some cases. On the other hand, they are not commonly used even on relatively static data, and when they are given, they are very conservatively set. We point out future directions and give recommendations for the enhanced use of caching in the Semantic Web.

1 Introduction

Caching has been given a prominent place in the foundational documents of the World Wide Web. Out of the 6 documents that make up the HTTP 1.1 standard, RFC7234 [6] is entirely devoted to the topic. RFC7232 [7] defines conditional requests, and is also important when constructing caches. As RFC7234 notes:

The goal of caching in HTTP/1.1 is to significantly improve performance by reusing a prior response message to satisfy a current request.

Furthermore, caching is discussed throughout the Architecture of the World Wide Web [11], and the definition of the Representational State Transfer (REST) architectural style [8] is partly motivated from the requirement to implement efficient caching. We also note that caching in the Internet infrastructure, through so-called Content Delivery Networks, is both a large business area and could provide great value to the Semantic Web.

If used correctly, caching mechanisms will reduce the need to make HTTP requests, reduce lookups to the backend systems, reduce the need to make repetitive computations, enable sharing of responses in Internet infrastructure, improve uptime and reduce latency since requests may be answered closer to the client.

In spite of this, we have not seen it in widespread use in the Semantic Web, and therefore we decided to conduct a survey to investigate the actual compliance to RFC7234 and RFC7232. The objectives of this paper are:
1. Understand the actual usage rather than rely on anecdotal conceptions.
2. Encourage the implementation of these mechanisms in Semantic Web infrastructure.
3. Point out future research directions.

The contributions of this paper are to meet these objectives by means of a survey that shows that while the uptake has been moderate, practical benefits may be realized already. Based on this survey as well as practical experience, we point out future research directions as well as some recommendations for deployed implementations.

We note that caching is not only useful for long-living resources, even though that may be the most important use. If a resource is frequently requested, it may make sense to cache it even though it may be fresh for only a very short period.

Caching may be deployed at several different levels: An HTTP cache may be in a reverse proxy close to the server, in which case it may have much in common with a conventional database cache. It may also be anywhere between a server and a client, in which case it may be shared, i.e. it may cache responses from a number of servers to many clients. Another example is an institutional forward proxy, which are close to several users. Finally, the User Agent may implement a private cache for its user at the client side.

1.1 HTTP Caching standards

As mentioned, the two documents from the HTTP 1.1 standards suite that are relevant for this study are RFC7234, named “Caching”, and RFC7232, named “Conditional Requests”. The main difference is that the caching standard defines when a response may be reused without any contact to origin server, whereas the conditional requests define how to validate a response by contacting the origin server. The two can be combined: Clients and proxies may use the latter to revalidate a response that has been cached based on the former.

RFC7234 defines two important headers. The first of which is \texttt{Expires}, whose value is a date and time of when the response is considered stale, and therefore should not be used. The second is \texttt{Cache-Control}, which allows detailed control of the cache, including a \texttt{max-age} field, which gives the time in seconds for how long the response may be used from the time of the request. \texttt{max-age} takes precedence over \texttt{Expires}. In this article, \textit{freshness lifetime} is understood as the number of seconds that the response may be used without contacting the origin server. Ideally, the calculation of the freshness lifetime should be based of the above, we therefore shall refer to this as “standards-compliant caching”. It can also be based on heuristics, Section 4.2.2 in RFC7234 provides some loose constraints for such practice as well as a suggestion for a useful heuristic. This heuristic is based on a fraction of the time lapsed between the current time and the modification time given in the \texttt{Last-Modified} header. This approach still requires the Web server to be cooperative to be successful. Commonly, Web servers can track this, for example if RDF is served from a file system the file modification time is used.
Table 1. Recorded HTTP headers

<table>
<thead>
<tr>
<th>Header</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>RFC7234</td>
<td>When obtaining response from a cache, the number of seconds since validation</td>
</tr>
<tr>
<td>Cache-Control</td>
<td>RFC7234</td>
<td>Header used for a variety of directives</td>
</tr>
<tr>
<td>Expires</td>
<td>RFC7234</td>
<td>Gives the date/time after which the response is considered stale.</td>
</tr>
<tr>
<td>Pragma</td>
<td>RFC7234</td>
<td>Archaic HTTP 1.0 header</td>
</tr>
<tr>
<td>Warning</td>
<td>RFC7234</td>
<td>For additional information about possible incorrectness</td>
</tr>
<tr>
<td>Content-Type</td>
<td>RFC7231</td>
<td>To select the correct parser</td>
</tr>
<tr>
<td>If-None-Match</td>
<td>RFC7232</td>
<td>Request header to check if ETag has changed</td>
</tr>
<tr>
<td>If-Modified-Since</td>
<td>RFC7232</td>
<td>Request header to check if Last-Modified has changed</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>RFC7232</td>
<td>When the resource was last modified</td>
</tr>
<tr>
<td>ETag</td>
<td>RFC7232</td>
<td>An opaque validator to check if the resource has changed</td>
</tr>
<tr>
<td>X-Cache</td>
<td>RFC7231</td>
<td>Inserted by some caches to indicate cache status</td>
</tr>
<tr>
<td>Date</td>
<td>RFC7231</td>
<td>The time of the message. Used in conditional requests and heuristics</td>
</tr>
<tr>
<td>Surrogate-Capability</td>
<td>Edge [17]</td>
<td>Draft to allow fine-grained control for proxies.</td>
</tr>
<tr>
<td>Client-Aborted</td>
<td>libwww</td>
<td>Header inserted by User Agent to indicate that it aborted the download</td>
</tr>
<tr>
<td>Client-Warning</td>
<td>libwww</td>
<td>Header inserted by User Agent to give details about problems with the download</td>
</tr>
</tbody>
</table>

RFC7232, on the other hand, defines a protocol for asking the server if the cached response is still fresh using conditional requests. This doesn’t burden the content provider with the task of estimating the freshness lifetime beforehand. However, the server is then required to be able to answer if the resource has changed less expensively than it would be to serve the entire response. Either of two headers must be set by the server to achieve this: ETag, which sets an opaque identifier for the response, or Last-Modified which gives the time and date of the last modification of the resource. Clients that have obtained these values may use them to validate an earlier response by using If-None-Match and/or If-Modified-Since respectively in a subsequent request. If the server finds the response has not changed based on this, it will respond with a 304 status code and no body, otherwise it will return the full response. The other headers we recorded are listed in Table 1.

RFC7234 provides detailed control of caching, and caching may also be prohibited by the server, either by setting a non-positive freshness lifetime or explicitly using a no-store control field.

In this paper, we study to what extent SPARQL endpoints, vocabulary and data publishers support these standards. Data and code to reproduce this work are available at http://folk.uio.no/kjekje/#cache-survey.
2 Related work

We are not aware of any surveys of this type. Although the database literature is rich with query cache literature, it is mostly relevant to what would happen within the server or between the server and a reverse proxy, which is opaque to the Internet, and therefore not of our concern. For the same reason, caching that happens within the SPARQL engine is not relevant.

The Dynamic Linked Data Observatory (DyLDO) [12] performed, and continues to do so, monitoring of parts of the Linked Open Data Cloud to determine dynamicity characteristics of Linked Data. Caching is one of their motivations, but they have not published statistics on HTTP headers.

Linked Data Fragments is claimed in [20] to take advantage of caching and contrasts this with the unavailability of SPARQL query caches. They assert that this is an architectural problem. In [9], the authors examine cacheable as one of the desiderata for sustainable data access. They claim, without further justification, that SPARQL isn’t cacheable.

In [16] the authors implemented a reverse proxy that controlled the changes to the dataset, and therefore could make sure the proxy had all the information needed to determine freshness. We are interested in the situation where the changes cannot be controlled.

In [19], the term caching was used in a different sense than we use it. They rather prefetched an entire dataset to a local store and based on heuristics tried to determine which parts of the query should be evaluated remotely and locally. [15] explored when caching had a positive effect on complex SPARQL queries.

In the broader Web literature, [1] analysed the value of caching based anonymized traces of actual Web usage at a major Internet Service Provider. They found that while caching often yields little benefit when content is user-generated, there is still some potential.

While these studies have little overlap with the present paper, they underline the importance of understanding the current deployment and future potential. In some of the related work, it is shown that caching does not necessarily give tangible benefits. Yet, we shall assume that sharing the metadata required for caching outside of the server is desirable, and that it is possible in most cases. We shall see that it most likely will be beneficial in cases that do not benefit from caching today.

3 Methodology

We want to find information resources on the Web, and examine HTTP headers that may allow caching. To do this, we perform GET requests on SPARQL endpoints, vocabularies, dataset descriptions and other resources and record headers recommended by current standards, as well as obsoleted and non-standard headers. Additionally, we examine the triples in the returned information resources to see if there is information that may be used to calculate heuristic freshness.
We made several approaches to ensure that we visited a large and representative section of the open Semantic Web. We took SPARQL Endpoints from the SPARQLES survey [3], vocabularies from Linked Open Vocabularies (LOV) [2] and prefix.cc, and we augmented these data with spidered data from the Billion Triple Challenge (BTC) 2014 [13] dataset. Of these, BTC2014 is by far the largest, the others are small, curated and targeted datasets. However, the size is besides the point, we were only interested in examining as many hosts as possible, and they are still few.

We used SPARQLES survey list of SPARQL endpoints as of 2014-11-17, and filtered out those deemed unresponsive. This resulted in a list of 312 endpoints.

To examine as many different implementations and hosts as possible, we noted that the Billion Triple Challenge 2014 [13] dataset consisted of a 4 GTriple corpus of spidered Web data. This was seeded from datahub.io (aka CKAN), as well as other sources. To compile a list of candidates for further examination, we performed a series of data reduction steps, manually inspecting the result between each step. The details of this process are given in a companion technical report [14].

The end result of this process is a list of 3117 unique hosts, for each several resources would be visited, some several times, as they may host SPARQL endpoints, vocabularies, or other information resources, by a spider also detailed in [14], resulting in 7745 requests, done on 2015-01-02.

This results in an NQuads file per host, which is then loaded into a Virtuoso-based SPARQL endpoint for analysis by using the statistics system R [10] in the following section.

### 3.1 Challenges to validity

Key challenges to the validity of the survey are biases that may be introduced by the coverage and then the data reduction. The breadth of the Semantic Web is derived mainly from the BTC2014 crawl. While LODstats\(^1\) has presently seen an order of magnitude more triples, the number of error-free datasets were at the time of this writing 4442. We work under the assumption that cache headers are set mostly on a per-host basis, and if this assumption is valid, sampling a URL per host is sufficient. LODstats do not report per-host statistics, but often one host will host several datasets. Another recent crawl was reported by [18]. It is not clear how many hosts were crawled, but the number of triples is much smaller than that of BTC2014. It is therefore a fair assumption that BTC2014 fairly well represents the breadth of the Semantic Web, momentarily at least.

As for the coverage of vocabularies, we have verified that all resolveable vocabularies in LODstats that are found in more than 10 datasets are visited, and it is not far inferior in number to LODstats. The number of SPARQL endpoints found in SPARQLES is larger than LODstats, and we also looked for further endpoints both in the BTC2014 and our own crawl, finding only 18. If endpoints

\(^1\) [http://stats.lod2.eu/](http://stats.lod2.eu/)
went underdiscovered, then there is a discovery problem that is beyond this survey to rectify.

The data reduction that was subsequently done was mainly done to eliminate errors. We have not investigated biases that may be introduced by discarding momentarily dysfunctional parts of the Semantic Web, but we investigated whether the freshness lifetimes reported in the case of certain errors were distributed differently from those that returned a valid response, see the companion technical report [14]. We found that they were, but we have assumed that this is due to that errors are configured to be cached differently, which we know from experience is common practice. The following analysis is based on valid responses.

4 Analysis

The analysis is focused on finding descriptive statistics to understand how different servers support caching, for how long resources hosted with those that do support caching may be considered fresh, if it is possible to easily compute a heuristic freshness lifetime, and revalidate the response on expiry. Apart from quoting the numbers we aggregated, we do this by presenting summarized data distribution visualizations, to allow for an intuitive understanding of the data. Where appropriate, we also do statistical hypothesis tests, using so-called contingency tables, see [14] for details.

4.1 Different server implementations

First, we investigated whether certain server implementations provided better support for caching than others. To do this, we formulated SPARQL queries to examine the Server headers of successful responses. We used optional clauses matching the standards-compliant computed freshness lifetime (which is the ideal) as well as whether the response had other indications of caching-related metadata that may assist caching such as modification time, certain predicates, etc.

For each unique Server header, we found the ones where all responses had a freshness lifetime or other usable metadata. For the former, this amounted to 22 servers, which are listed in Table 2. 70 servers always responded with usable metadata. Inspecting the values we find the well-known Virtuoso and Callimachus servers, as well as the Perl modules RDF::LinkedData and RDF::Endpoint, which are partly developed by and run on a server operated by this author. Apart from those, we see that the they reveal very little about the RDF-specific parts of the underlying server implementation, e.g. Apache is a very common generic Web server, the others are also generic. A quick inspection of all Server headers confirmed that few reveal any further detail.

For a more systematic approach, we wish to test the hypothesis that some servers are better configured to support caching than others. Using the methodology given in the companion technical report [14], we find in both the cases of standards-compliant freshness lifetime and for the other usable metadata, the
Table 2. Server headers for hosts that enabled a freshness lifetime to be computed for all requests.

<table>
<thead>
<tr>
<th>Server headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DFE/largefile</td>
</tr>
<tr>
<td>2 git_frontend</td>
</tr>
<tr>
<td>3 nginx/1.3.9</td>
</tr>
<tr>
<td>4 thin 1.6.0 codename Greek Yogurt</td>
</tr>
<tr>
<td>5 Oracle-Application-Server-10g/10.1.3.4.0 Oracle-HTTP-Server [...]</td>
</tr>
<tr>
<td>6 Oracle-Application-Server-10g/10.1.3.4.0 Oracle-HTTP-Server [...]</td>
</tr>
<tr>
<td>7 TwistedWeb/8.2.0</td>
</tr>
<tr>
<td>8 RDF::Endpoint/0.07</td>
</tr>
<tr>
<td>9 Jetty(6.1.26)</td>
</tr>
<tr>
<td>10 nginx/1.6.1</td>
</tr>
<tr>
<td>11 Jigsaw/2.3.0-beta3</td>
</tr>
<tr>
<td>12 Apache/2.2.9 (Win32) PHP/5.2.6</td>
</tr>
<tr>
<td>13 Apache/2.4.10 (Unix) mod_fcgid/2.3.9</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15 GFE/2.0</td>
</tr>
<tr>
<td>16 RDF::LinkedData/0.70</td>
</tr>
<tr>
<td>17 Apache/2.2.17 (Unix) mod_wsgi/3.3 Python/2.6.6</td>
</tr>
<tr>
<td>18 Virtuoso/07.10.3211 (Linux) i686-generic-linux-glibc212-64 VDB</td>
</tr>
<tr>
<td>19 Apache/2.2.24 (Unix) mod_ssl/2.2.24 OpenSSL/0.9.8y</td>
</tr>
<tr>
<td>20 Apache/2.2.22 (Fedora)</td>
</tr>
<tr>
<td>21 INSEE</td>
</tr>
<tr>
<td>22 GitHub.com</td>
</tr>
</tbody>
</table>

test reports $p$-value = 0.0001. We can conclude that it is highly likely that some servers are better at exposing cache headers than others. Unfortunately, since most Server headers only contain generic values, little can be learnt about these implementations. We note, however, that DBPedia exposes standards-compliant freshness lifetime of 604800 seconds (i.e. 1 week) for both LOD and SPARQL endpoints. DBPedia has historically been updated only a few times a year, but this was probably chosen to avoid making a commitment far into the future. It may provide considerable benefits.

4.2 Other caching headers

We also looked for other headers in Table 1. We found Pragma (archaic HTTP 1.0 header) in 287 responses, but except for two hosts, where they were superfluous, they were only used to prohibit caching. Surrogate-Capability were not observed.

4.3 Distribution of freshness lifetime

We obtained a successful response from a total 2965 information resources, either with SPARQL results or RDF data. A successful response is rather strictly
defined, not only must there be a successful HTTP response after redirects are resolved, the response must also return a valid RDF media type (unless it is a SPARQL result) and the response must parse into an RDF model. We have given priority to survey many hosts since configuration usually doesn’t differ much across a host, especially since it also captures different types of resources. It is therefore acceptable that the number of resources is relatively small.

Since we are interested in the properties of valid response, including examining some of the RDF contained in them, and that web servers may be configured to instruct clients and proxies to cache errors differently, we will study the statistical properties of valid responses.

**Standards-compliant caching headers** Of the 2965 resources, 405 returned valid headers, but 114 did so to prohibit caching of the response, and 3 contained conflicting headers, i.e. set a freshness lifetime, but also prohibited caching. In most cases, *Cache-Control* and *Expires* both occurred, but the former is more common than the latter in the cases where only one of them occur. Additionally, 269 resources had a *Cache-Control* header to control other aspects of caching than lifetime, i.e. to say that only private caches may use the response, that the cache must be revalidated, or to prohibit caching. Note that the freshness lifetime is 0 whenever caching is prohibited.

In Figure 1, there is a barplot where the freshness lifetime is grouped in bins. We see that in these categories, the most common is to prohibit caching. Nevertheless, many also declare a standards compliant freshness lifetime in minutes to days.

In Figure 2, we have broken this up by the type of resource that was accessed, i.e. SPARQL endpoints, vocabularies, dataset descriptions or unclassified information resources. Firstly, we note that it seems like the distribution of freshness lifetime is quite different for the different types, an observation that is also supported by a similar hypothesis test as above, with a $p$-value $= 0.00001$ (note, however, it is more contrived than above, since the bins are like in Figure 2, which is chosen for intuitive interpretation rather than statistical rigor). Secondly, we note that it is often prohibited to cache dataset descriptions. This is odd, since statistics about datasets is usually costly to compute and should be cached. The VoID specification [4] also notes that the statistics are considered estimates.

We also note that prohibition of SPARQL results caching is rare. Amongst the servers that expose caching headers it is common that the result may be cached for some minutes, and closer inspection of the dataset reveals that many of these are due to that RBKExplorer\(^2\) sets an freshness lifetime of 300 s for many endpoints.

**Simple heuristic freshness estimates** We next consider the simple heuristic freshness lifetime as suggested in Section 4.2.2 in RFC7234 and mentioned in the introduction.

\(^2\) [http://www.rkbexplorer.com/about/](http://www.rkbexplorer.com/about/)
Figure 1. Barplot counting all standards-compliant freshness lifetimes found, with the percentage of occurrences indicated on the bars. On the horizontal axis, the first bin are the cases where caching is explicitly prohibited. The next bins are for lifetimes, where the values are grouped if they are on the order of seconds, minutes, hours, etc, i.e. the second bin counts the lifetimes in the interval \([1,59]\) seconds, etc. On the vertical axis, the number of times a certain freshness lifetime was found.

We were able to compute a heuristic lifetime for 554 resources, a larger number than standards-compliant resources. In Figure 3, we see that the distribution of lifetimes is radically different from the case in Figure 1. In this case, we may cache many resources for months. Only a handful of resources changed in the last minutes. Since this is based on actual times since last modifications, this suggests that many resources should have had explicit cache headers with very long lifetimes. This is supported by DyLDO [12], which concludes that:

\[\ldots\text{We found that 62.2}\% \text{ of documents didn’t change over the six months and found that 51.9}\% \text{ of domains were considered static.}\]

This agrees well with that 60\% of the simple heuristic lifetimes are in the month range.

Moreover, by inspecting Figure 4, we note that the difference between different types of resources is much smaller. This is confirmed by a hypothesis test that yields \(p\)-value = 0.02.

We find that only one SPARQL endpoint yields a heuristic lifetime, on closer inspection, we find this to be hosted by Dydra\(^3\). We speculate that this is due to that few underlying DBMS systems help track modification times in a way that can be used on a SPARQL result basis.

\(^3\)\url{http://dydra.com/}
Figure 2. Mosaic Plot. On the vertical axis, the size of the boxes are determined by the fraction of the types of resources. On the horizontal axis the width of the boxes is proportional to the total counts, using the same bins as in Figure 1. From bottom to top, light blue boxes denote SPARQL endpoints, dark violet dataset descriptions, orange generic information resources and light orange vocabularies.

Figure 3. Barplot counting all simple heuristic freshness lifetimes found. Axes as in Figure 1.
Heuristic freshness from Dublin Core properties  We noted that the Dublin Core Metadata terms vocabulary has a number of predicates that may become useful in determining heuristic freshness in the future, so we recorded any statements containing the predicates `dct:date`, `dct:accrualPeriodicity`, `dct:created`, `dct:issued`, `dct:modified` or `dct:valid`.

First, we compared dates given in `dct:modified` to dates given in `Last-Modified` when both are available for a resource. They were often not the same, but it appeared that dates in the former are further back in time than the latter. We speculated that this may be due to that the web-server tracks semantically insignificant changes through the file system, while authors of RDF only update timestamp when significant changes are made, or it may be that authors forget to update their timestamps.

`dct:modified` occurred in 2687 triples, but 2487 of these does not have the Request-URI of the information resource as their subject, i.e. it gives the last modification time of some subgraph of the returned RDF. Nevertheless, given its prevalence, it is highly likely that the presence of `dct:modified` will be useful in determining a heuristic freshness lifetime, as the latest date may be used.

`dct:valid` occurred 21 times, and could be used in lieu of an `Expires` header, but none of these occurrences had a date in the future.

`dct:accrualPeriodicity` occurred only twice, and in neither case contained machine readable data.

`dct:date`, `dct:created` and `dct:issued` were present in 36, 389 and 1475 triples respectively. They correspond roughly to the `Date` header, which is present.

Figure 4. Mosaic Plot for heuristic freshness lifetime. See caption of Figure 2 for description.
in all requests, and they are therefore not important, but given their prevalence they could be useful in further analysis.

4.4 Cache validation

So far, we have considered the case where the client or proxy does not send a HTTP request to the server when a resource that is present in the cache is requested. This is desirable if the client can be sufficiently confident that the response is fresh, either by having a standards-compliant freshness lifetime or a heuristic to determine it. At the end of this period, or if the previous response have no lifetime, due to lack of information, or to that the server has stated so in the Cache-Control header, responses must be revalidated. In this case, RFC7232 defines the behaviour, with ETag and Last-Modified as the relevant response headers.

The BTC had recorded these headers in their data, where 1733 had the ETag header and 690 had Last-Modified with a great overlap. For the resources where either or both were available, we made our initial request conditional, and 911 responses were verified as still fresh.

In total, 1260 successful initial responses contained an ETag, 606 for vocabularies, 117 for datasets, just 12 for endpoints and 525 for unclassified information resources.

To see if the server actually supported conditional requests, and not just merely set the response headers, we made another 1822 requests to the resources that had these headers. Then, we checked if the response code was 200, and the conditional headers had not changed since our initial requests. In 85 cases, conditional requests were not supported according to the standard, no cases for endpoints, 3 for datasets and 23 for vocabularies, 59 for generic information resources.

5 Conclusions and outlook

We found moderate uptake for HTTP caching and conditional requests in the Semantic Web. We found, in agreement with DyLDO [12], that many resources change at a very slow pace, but also that this is not reflected in the standards-compliant freshness lifetimes advertised by servers.

We found that errors are commonplace, but that they do not usually pertain to the caching headers. We found a small number of self-contradictory Cache-Control headers, and some servers that set conditional request response headers, but could not support conditional requests.

It is possible in a substantial number of cases to compute a heuristic freshness lifetime, either from the Last-Modified header, or from Dublin Core properties.

For SPARQL endpoints, we found that conditional requests are seldomly supported, but standards-compliant freshness lifetimes have been seen, and since it is supported by DBPedia, benefits from caching may be realised already.

In spite of this, most of the Semantic Web is unhelpful even though it is changing slowly.
5.1 Future work

In this work, we have only explored the cases where the server is cooperative, in the sense that message data or metadata provides at least a hint of a resource’s cacheability. We also noted that the majority of the Semantic Web is unhelpful. Therefore, an interesting direction is to learn the change frequency of resources to use in a heuristic freshness lifetime estimation. However, such work should operate within the loose constraints of Section 4.2.2 in RFC7234, and should find its niche when the other techniques described in this paper are unavailable. Once this is done, the correctness of caching headers should be assessed, possibly using contingency tables similar to those in the companion technical report [14].

Investigate whether curated collections such as LOV or SPARQLES contain resources that have different characteristics. Since the different sources we surveyed overlap, this cannot be done with the simple hypothesis test in this paper, but requires more sophisticated statistics.

We found that some implementations are likely better than others, but further understanding of these differences are impeded by the fact that most Server headers contained very little information. Future work could seek more sophisticated fingerprinting and understanding of the nature of these differences. With this, it may also be possible to investigate whether expiry times have been set consciously.

Further investigate the suitability of the \texttt{dct:modified} property for estimating heuristic freshness lifetime.

Estimating the freshness lifetime is a challenging problem for data owners. It must necessarily include the human users involved in the publishing cycle since they are making a commitment about future changes. Designing user support systems as well as interfaces that fit the publisher’s workflow is an important problem.

We believe that [20] and [9] prematurely reject caching in connection to SPARQL. They are correct that currently, query results can only be cached on a per-query basis. Moreover, semantically insignificant changes to a query, such as whitespace or the order of triple patterns in a basic graph pattern, currently causes problems for caches. The latter problem can probably be fixed by developing digest algorithms. Such algorithms exist, but are focused on cryptographical strength, and much simpler algorithms could be used for this problem.

Furthermore, by using similarity measures, like those discussed in [5], shared proxies, e.g. an institutional cache or a Content Delivery Network, can examine queries for frequent subpatterns and proactively prefetch data into a cache to help answer queries on the proxy. A cost model that takes into account several data access methods, those described by [9] as well as the novel work in [20] may be a key ingredient in enabling efficient SPARQL evaluation on the Web.

Even though it is clear that other parts of the Web benefit greatly from caching, and that the potential for HTTP metadata to better reflect actual update practices in the Semantic Web is great, estimating the actual impact of doing so should be a topic for further research.
5.2 Recommendations

It is mainly the objective of this paper to be descriptive, but from the results of this study and on the results of DyLDO, we note that it is highly likely that most cache prohibitions are misguided, and server administrators are advised to turn them off unless they are sure they are required.

Additionally, based in part on the survey, but also on other practical experience, we suggest the following:

In many cases, setting reasonable cache headers is straightforward, and should be done by data owners. Framework authors should make it easy to set the expected lifetime, heeding the metadata association good practice recommendation of [11]. If the author is unable to influence HTTP headers, they should set a `dct:valid` time into the future and make use of `dct:modified`.

To allow generation and validation of `Last-Modified` and `ETag` headers, DBMS authors should make sure it is much cheaper to retrieve the modification time of any subgraph, than to retrieve the subgraph itself. This would be a great improvement for RFC7232-based caching, when revalidation is required. It would also help simple heuristics based caching. Research in that direction has been published in [21].

A change periodicity predicate should be standardized in VoID [4].

All Web cache implementations we have studied have cached responses in the form of a key that identifies a serialized object. For short-term impact, future work should accept this as an architectural constraint.

Acknowledgements The author would like to thank Jürgen Umbrich and Martin Giese for careful review and critical comments, and Gregory Todd Williams for promptly solving issues in the underlying libraries used in this study, Axel Polleres for encouraging comments and Jonas Smedegaard for proofreading. We also thank the anonymous reviewers for extensive reviews. Finally, many thanks to Helen Murray for linguistic assistance.

References

Addendum to a survey of HTTP caching on the Semantic Web

Kjetil Kjernsmo
20th March 2015

1 Introduction

This Technical Report is an addendum to a paper to be published in the proceedings of the Extended Semantic Web Conference 2015 titled “A survey of HTTP caching implementations on the open Semantic Web”.

That paper was motivated from the perception that we had that caching was much used on the open Semantic Web. This is surprising, since caching has been given a very prominent role in the architecture of the World Wide Web, and caching based on HTTP headers as defined in RFC7234 [4] and RFC7232 [5] is prevalent elsewhere in the Web.

To understand the actual usage, we conducted an extensive survey of actual resources on the Semantic Web. We obtained data from different sources, with the breadth of the Semantic Web mainly covered by the Billion Triple Challenge (BTC) 2014 dataset [7].

In this addendum, we document in detail several aspects that there were not sufficient space for in the conference paper. Since we made the assumption that the relevant headers are mostly the same for each host, Section 2 documents how the large BTC2014 data set is reduced to visit a large number hosts, but a small number of resources. Then, Section 3 discusses the system we developed to visit these resources and extract the information that we were interested. Section 4 documents our main statistical method and Section 5 has an analysis of the distribution of valid versus invalid responses.

2 Implementation of data reduction

The BTC2014 dataset provided data in the form of NQuad files. Due to the presence of invalid RDF, we iterated through the NQuad files on a line-by-line basis. First, we matched each line against a regular expression were lines matching ontology|endpoint|sparql|vocabular passed the filter. Then, the Perl framework RDF::Trine\(^1\) was used to parse the line. Lines that failed to parse were discarded. We have not investigated whether this could introduce biases. Statements were then accepted into a new NQuad file if they had a predicate that matched the sd:endpoint or matched a case-insensitive regular expression sparql if the subject and objects were both resources, or the predicates void:vocabulary, rdfa:vocabulary, or api:vocabulary, as well as having a

\(^1\)https://metacpan.org/release/GWILLIAMS/RDF-Trine-1.011
resource as object. Finally, statements with the the classes `cogs:Endpoint`, `owl:Ontology` and `voaf:Vocabulary` in the object position were also accepted\(^2\). More classes and properties were considered, but not used in the data reduction if they did not occur in the original data.

In the next step, we filtered out statements with URIs that were invalid or irrelevant, e.g. URIs that didn’t have a scheme or where the scheme weren’t HTTP(S), or they were referring to private IP addresses.

We then sought to classify resources into the categories “endpoint” for SPARQL endpoints, “vocabulary” for vocabularies, “dataset” for datasets that may contain further descriptions of several resources, or simply “inforesources” for those that did not fit in the above classes. To do so, we classified based on certain predicates and classes. Additionally, URIs derived from prefix.cc were classified as “vocabulary” (even if we found several that were not) and those from SPARQL:ES as “endpoint”.

Since we blatantly violated URI opacity with our regular expression matching in the first step, we needed to further filter candidates for SPARQL endpoints. This step therefore included filtering as well as classification.

We found in the data a large number of ontologies that consist of many information resources with just a few triples in each. Since they appear to be produced by the same software, usually Semantic Mediawiki, we assume that they are configured with a single setup, and thus we merely sampled these resources.

We continued to also sample the HTTP headers gathered in the BTC2014 dataset. First, we traversed the files with a simple UNIX `grep` to find the resources that had reported one of the RDF serializations as content type. We then traversed this list, first discarded the resources that did not have a valid IRI (this amounted to just 3273 resources). For the resources we found, as well as for the resources that was of `rdf:type owl:Ontology` above, we kept one resource per hostname, with the exception of the popular blogging platforms Livejournal and SAPO, where each blog has their own host and they expose FOAF data. For those, we only kept one hostname, since they are likely configured similarly.

### 3 Implementation of spider

We then developed a fairly elaborate parallel spider to examine the resources found on hosts that the previous steps deemed interesting using the Perl frameworks RDF::Trine and libwww. The spider operated with a timeout of 20 seconds and a maximum message size it would accept of 1 MB.

The parallel spider would then launch a process per host, but each request to one host would be delayed by 10 seconds. For each host, the spider would go through the list of URLs found by previous steps for that host. Since the BTC2014 recorded the `Expires`, `Last-Modified` and `ETag` headers where they existed, we first examined whether any of the resources were still fresh, but none were. Wherever the last two headers existed, we added the corresponding `If-Modified-Since` and `If-None-Match` headers for a conditional initial request.

For endpoints, we made the following SPARQL query:

\(^2\)Linked Open Vocabularies [1] may be used to resolve these prefixed names
SELECT DISTINCT ?Concept WHERE {[] a ?Concept} LIMIT 2

which should be quite light, yet likely yield results.

Then, the first request would be made, and a selection of the resulting HTTP headers recorded in an per-host NQuads file. For this purpose, we developed and released a module RDF::Generator::HTTP\(^3\) to CPAN. We recorded whether the conditional request showed that the BTC2014 data were still fresh, and if it was, we retrieved the current data, as we had not coupled the headers to the body in our original retrieval. Based on the resulting headers, we let libwww\(^4\) calculate both standards-compliant and simple heuristic freshness lifetime.

If the initial response had RFC7232 headers, we made another request to see if the server included the headers but does not support conditional requests. The heuristic we employed is that if the headers remain the same, but the result was returned, rather than just a response code 304 (which indicates that the previous result can be reused), the server does most probably not support it.

For endpoints, we examined the response message, to see if there are any results to our query, and recorded that if there are. In addition to the endpoints registered in the SPARQLES survey [3], our process found 18 endpoints that responded with results. For all others, we parsed the response, and recorded any errors if the parser concluded the content were invalid.

For resource types other than “vocabulary”, we look for SPARQL endpoints in the response, using the predicates sd:endpoint and void:sparqlEndpoint. We then do the same query as above and record the relevant headers. Unfortunately, we found early that this only turns up misconfigured endpoints that point to localhost, and was removed from the spider for the final analysis.

Finally, if the Linked Open Vocabularies [1] SPARQL endpoint used a URI for the vocabulary that was different from the namespace URI (after a normalization step), another request would be made to record the selected HTTP headers from that as well.

4 Statistical method

The hypothesis tests in this paper were implemented using contingency tables (see e.g. [2]). This formalism is suited to see if the distribution of Server headers are different for those implementations that offer caching headers from those that don’t. Intuitively, we expect these distributions to be similar, “long-tail” distributions, i.e. a handful of servers are used by a large number of projects, and then it falls off rapidly, and so, some servers are used only by very few. Likewise, it is to be expected that only a few projects have given caching enough attention, but that they account for the majority of the support. The question is if the presence of caching headers is merely a matter of proportion, or if there are some that have given it more attention, but still is in relatively little use.

Using the statistics system R [6], we use a statistical test, namely Pearson’s \(\chi^2\) test with simulated \(p\)-value (based on 10000 replicates). The simulation is done using a Monte Carlo method to compensate for the fact that many servers

\(^3\)https://metacpan.org/release/KJETILK/RDF-Generator-HTTP-0.003

\(^4\)https://metacpan.org/release/GAAS/libwww-perl-6.04
will not expose caching headers at all, an issue that would otherwise violate the underlying assumptions of the test.

Since the same test can be used for binned data in many different situations, we used it more informally in other parts of the paper as well.

5 Valid versus invalid responses

Since web servers may be configured to instruct clients and proxies to cache errors differently from successfully served requests, it is interesting to investigate whether we can simply use all cache headers in the analysis, or if we should remove unsuccessful responses to avoid introducing bias to the analysis of valid responses.

For a visual inspection, we may use a quantile-quantile plot to plot the freshness lifetimes of successful responses versus requests that failed for some reason. If the distribution of both variables are the same, these points will lie on a straight line. In Figure 1, we see successful responses vs. the most common parse error, and it is clear that these are so different that no further formal analysis is necessary. This is one of the reasons why we filtered unsuccessful requests before we analyzed the freshness lifetimes in the paper.

![Figure 1: A quantile-quantile plot of freshness lifetimes for standard compliant headers with and without errors.](image)
References


Interview: How can scientific methods provide guidance for Semantic Web Research and Development?

Kjetil Kjernsmo¹ and Ruben Verborgh¹

¹ Department of Informatics, Norway
² Ghent University – iMinds, Belgium
kjetil@kjernsmo.net, ruben.verborgh@ugent.be

The organisers have asked Kjetil Kjernsmo to prepare a interview around his submission about an epistemological discussion around Semantic Web research. Together with one of the reviewers, Ruben Verborgh, he examined his contribution in this new format with success. The questions of the audience were answered during the succeeding discussion section.

Interview transcript

So, Kjetil, what’s your confession? Like most people at this conference, I claim that my work advances the Semantic Web, but I can’t prove that. In fact, I don’t know any way to reach any conclusive result with respect to a larger Semantic Web vision, so in a way, all results are inconclusive.

We hope to advance the Semantic Web by doing research within the boundaries of the known. Isn’t that all we can do? It seems hard to break out of the boundaries of the known, but no, I don’t think it is good enough, and that we need to challenge this view, and that we should look to philosophy of science and natural science for inspiration. My fundamental assumption, which I hope you will agree with, is that science has had a remarkable rate of progress, and that we should adopt the practices that achieves this.

Now, we just end up testing one little piece at a time, based on what is already there, but the fact is that if we get to the visions that are presented as the Semantic Web, then it will be completely different in terms of workloads, data profiles, etc. And therefore, the central problem is that to really argue that we progress towards the Semantic Web, we have to test against a reality that doesn’t exist yet. That makes the problem different from other sciences.

It seems that your background is not computer science then? Yes, I’m not a computer scientist. For my master’s degree, I studied theoretical astrophysics. I only happened to bump into Dan Brickley for a side project in 1998, who went in to Chair the RDF working group the week after, and after I finished my master’s, I’ve worked with Semantic Web in the computer industry. It is funny, I didn’t work much as a physicist, but this training still makes me feel like a natural scientist. I’ve also read a lot of history and philosophy of science. I think that we might find a key to speed up progress if we start discussing epistemology.
What does "epistemology" mean? Naively, epistemology is the study of what it really means to say that you know something. So, how do you really know that a triple store performs better than another? How do you know that a certain ontology engineering approach will forward the Semantic Web and not hinder it?

I understand your frustration with the lack of progress and the lack of methodological clarity, but I’d like to understand why you’re concerned with this. Ok, I’ll introduce and example from modern astronomical history. I studied quasars, which are active central parts of very remote galaxies. Rather spectacular and complex stuff, there was initially a lot of confusion, but many possible directions were quickly terminated. Like, it was quickly clear that they were remote objects, just not how insanely remote they were. After a couple of years, people were finding the spectra, determining redshift, etc. Very cleverly, but with a clear methodological direction, after a few years, the picture of this very remote, complex thing became quite clear, and has largely stood the test of time. Now, the big difference is that with quasars, we were discovering a world that existed, but that we didn’t know about, but we quickly figured out how to understand. With the Semantic Web, there’s a world that we’re creating, but we don’t know how to understand how to get there. So, if you accept the premise that science shows a remarkable rate of progress, then I would like to discuss and understand what it is about science that does this.

But isn’t the Semantic Web progressing equally fast? First of all, what do we mean by the Semantic Web?

A machine-readable Web of data... I think it has to be much broader defined, for one thing, it is also machine writeable. In the keynote at ISWC 2011, Frank van Harmelen discussed the the heterogeneity problem and argued that the solutions are not so much technical, but social, economical and cultural, and a definition of the Semantic Web must reflect that. I’ve been around for 17 years, and I don’t think it is progressing very fast. If you look at job postings, you’ll find that the number of jobs containing Semantic Web is varying but low, and orders of magnitude lower than e.g. MongoDB. Also, Google Trends shows a steady decline. And we still spam W3C mailing lists with CfPs, even though if we had the Semantic Web, it should be easy to create a system to match a paper to a venue. I liken the Semantic Web to a flying car: People had visions about flying cars and how they would impact society. Guess what: flying cars exist, they have been built. But there are 5 of them. Or maybe 10. But clearly, their mere existence does not give them the impact they had in those visions.

There has been some papers on research methodology, for example, Avi and Natasha have their "Is This Really Science? The Semantic Webber’s Guide to Evaluating Research Contributions" [1], doesn’t that provide enough guidance? It is an excellent read, but it largely adopts a falsificationist view of science,

as it was argued by Karl Popper. He said that one should seek to formulate bold conjectures, so that if we’re wrong, then a test should easily show that. So, considering the epistemology, the things that were knowable are the things that you can formulate a falsifiable hypothesis for. The funny thing is that falsificationism worked really well for me when I studied quasars. There are two large theories that I worked within, Theory of General Relativity and Quasar Theory. It was easy, even for a master’s student, to come up with conjectures that were falsifiable, not only for my own work, but observations done within the framework I developed could potentially brings parts of the large Quasar Theory down with it. That’s of course a long shot, but it is really cool to know that when you’re a student.

Why can’t we just apply falsification to the Semantic Web? That’s what researchers do—and successfully, it seems? For the reasons that falsification has been criticized for by more recent philosophers. I would recommend "What is this thing called Science" by A.F. Chalmers [2]. The TL;DR version is "science is pretty darn hard, especially when doing research". By that, I mean that you can always construct a philosophy of science around conjectures like "all swans are white", but as researchers our conjectures are never that simple. For example, you can never know if it is really your hypothesis that is wrong, or the evaluation. So, it might both happen that your hypothesis is really true, but you rejected it due to a faulty evaluation methodology, or your hypothesis was false, but you failed to reject it, because the evaluation was not good enough.

Do you have an example of this? There was a good talk at ISWC last year, where the author analyzed several SPARQL benchmarks, and showed how each of them were unsuited for important real-world considerations. However, the problem is that if you applied the same methodology to his own study, that too would have fallen. So, to gain any knowledge, you always have to question your methodology. Moreover, it could be that there are auxiliary hypotheses that you simply accepted, it could be that your hypothesis had not been rejected if the triple store data structure had been a trie and not a B-tree.

So falsification does not work in all cases? Some philosophers have trolled the scientific community and said stuff like "falsificationism would destroy all of science". For example Tycho Brahe correctly deduced that Copernican theory predicted an observable property, which he, despite his instruments being orders of magnitude better than his predecessors, could not see. So if you really adopted falsificationism, you’d probably reject prematurely in many cases. Besides, what if falsificationism was applied to the Semantic Web visions, wouldn’t we have rejected the whole thing by now, stopped our conferences and gone on to do something else? So, no, it is not just that falsificationism doesn’t work well, it is that we wouldn’t really have gotten started, and isn’t really what scientists, historically, at least, have been doing.
At ISWC2013, you presented a paper that attacked benchmarking [3], and proposed to use statistical design of experiments instead. Is that part of that picture? Yeah, I wanted to illustrate a methodology where you could actually test the evaluation to see if the methodology stands up to scrutiny.

But the flaw here seems to be that you would test the methodology, but you need another test of the test methodology, to evaluate that methodology, and so a test of that methodology again, ad infinitum... Yeah, that’s the kind of philosophical problem I’m struggling with. There’s a recent direction in the philosophy of science called the "New Experimentalism" that focuses on experimentation, instrumentation, laboratory practices, that goes further in this direction, where it is not sufficient to test a hypothesis, you need to prove that the test is severe, and they go on to formalize what severe means. I have not yet appreciated the broad relevance of this philosophy, but the people working on this are still alive, and it seems promising.

As you talk about hypotheses and theory, can you formulate a theory of the semantic web, like you have the quasar theory? I think it is perhaps one of the most important exercises we can do, but I’m not confident enough to attempt that, it must necessarily be a community effort.

Ok, but can you explain what is meant by "hypothesis" and "theory", then? Maria-Esther forwarded Oxford definitions, but I don’t think you’d find consensus about those if you asked philosophers, it is remarkable. Superficially, I think hypotheses are targeted conjectures, easy come, easy go things, things that you try to make testable in everyday research. A theory is a collection of hypotheses that have been tested.

A collection, can you attempt a more stringent definition? Well, I can try a definition that I personally find good, but that doesn’t mean it is well founded in philosophy of science. It would be "A theory is a coherent set of hypotheses...".

Coherence, what would that be in this context? Coherence places a burden on the researcher when considering the big picture: If my work finds that I get a performance benefit in spite of a longer connection time, it will be incoherent with a hypothesis that connection time must always be kept to a minimum. There are some major schools of thought called coherentism, which requires that all hypotheses are not only consistent, but also meet the requirements of e.g. Occam’s razor. To me, pure coherentism appears to have some merits when dealing with large, theoretical frameworks, such as string theory, or indeed much of the work on logic that this community does. A good thing to say about it would be that it might have many points of attack, and a successful attack could bring the whole theory down.
What do you mean by "attack", is that a to test the theory against reality? Yeah, so my full definition would be something like "A theory is a coherent set of hypotheses that are held as true after having been subjected to vigorous testing by a scientific community".

So this brings up social aspects of science. This is something the physicist and philosopher Thomas Kuhn emphasized, right? Yes, but I'm not a big fan of Kuhn. He introduced some interesting concepts, like "normal science", "scientific revolutions", "paradigms", and the emphasis on science as a social process. So, yes, with this definition makes it important that the community is capable of vigorous testing. I think he also makes an important point in that he thinks philosophy of science must be based on history of science, and in that way, he differs from Popper. The problem is, he got significant parts of his history wrong, and for that reason, I reject most of his philosophy.

You opened by saying that every result is inconclusive. I suppose that can be said for anything, also in physics. For example, Newtonian mechanics is inconclusive in retrospect, since it has been replaced by relativity. So, what is different in our field? Yeah, that's a fair point to argue, but our case is very different, because our hypotheses are tested against a reality that doesn't exist yet. Once the Semantic Web exists, it will be very different from what we have today, like if everyone had a flying car, we'd have a completely different set of traffic regulations. So, you can take the exception, and say that this is how Computer Science operates, and there's nothing you can do about it. If you do that, how can then this field achieve the rate of progress that natural sciences has? I'm arguing that this isn't good enough, and that's why I initiate this discussion. In all fairness, I think that if you claim that you are forwarding the Semantic Web (and I suppose you do if you're at this conference), you have the burden of proving that you do. But that can only be done if we devise a way to test our hypotheses against a more clearly defined Semantic Web Theory.

So, to round off, what do you suggest we do? First, we need to acknowledge there's a problem, that just testing pieces of the big picture isn't good enough. Secondly, we have to start formulating theories. Then, we have to increase the prestige of empirical methods. For example, can we formulate our tests as "severe" with the "New Experimentalism"? And, finally I think we should spend more time to sit down and hack and bring the code to the people.

References