A Survey of HTTP Caching Implementations on the Open Semantic Web

Kjetil Kjernsmo(✉)

Department of Informatics, University of Oslo, Postboks 1080 Blindern,
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here: Research, implementation, and evaluation of caching infrastructure is uninteresting as long as data providers do not publish relevant metadata.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.

F. Gandon et al. (Eds.): ESWC 2015, LNCS 9088, pp. 286–301, 2015.
DOI: 10.1007/978-3-319-18818-8_18

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 `Expires`, whose value is a date and time of when the response is considered stale, and therefore should not be used. The second is `Cache-Control`, which allows detailed control of the cache, including a `max-age` field, which gives the time in seconds for how long the the response may be used from the time of the request. `max-age` takes precedence over `Expires`. In this article, *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, Sect. 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 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.

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.

Table 1. Recorded HTTP headers

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

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

[1] http://stats.lod2.eu/.

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 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::End point, 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

Table 2. `Server` headers for hosts that enabled a freshness lifetime to be computed for *all* requests.

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

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 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 Fig. 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 Fig. 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 Fig. 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.

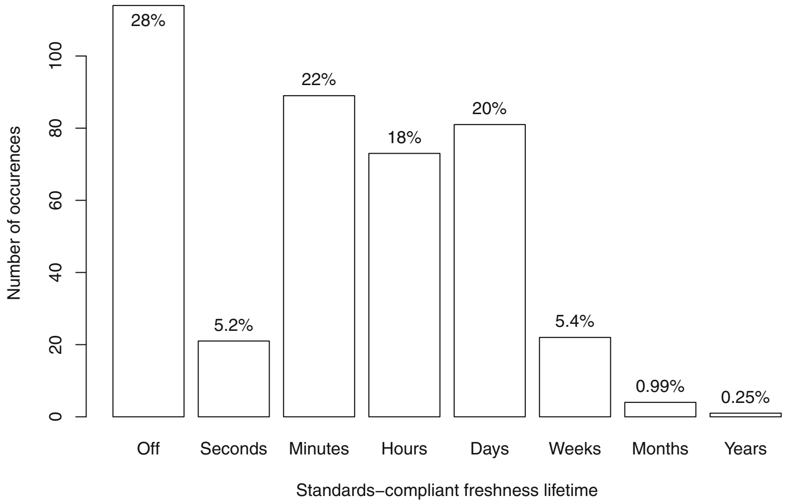

Fig. 1. Barplot counting all standards-compliant freshness lifetimes found, with the percentage of occurrences indicated on the bars. On the horizontal axis, the first bin are the cases where caching is explicitly prohibited. The next bins are for lifetimes, where the values are grouped if they are on 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.

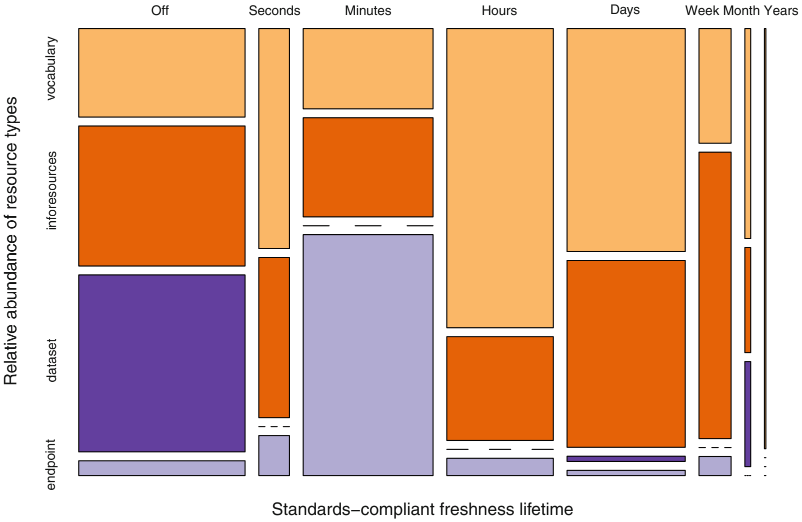

Fig. 2. Mosaic Plot. On the vertical axis, the size of the boxes are determined by the fraction of the types of resources. On the horizontal axis the width of the boxes is proportional to the total counts, using the same bins as in Fig. 1. From bottom to top, light blue boxes denote SPARQL endpoints, dark violet dataset descriptions, orange generic information resources and light orange vocabularies (Color figure online).

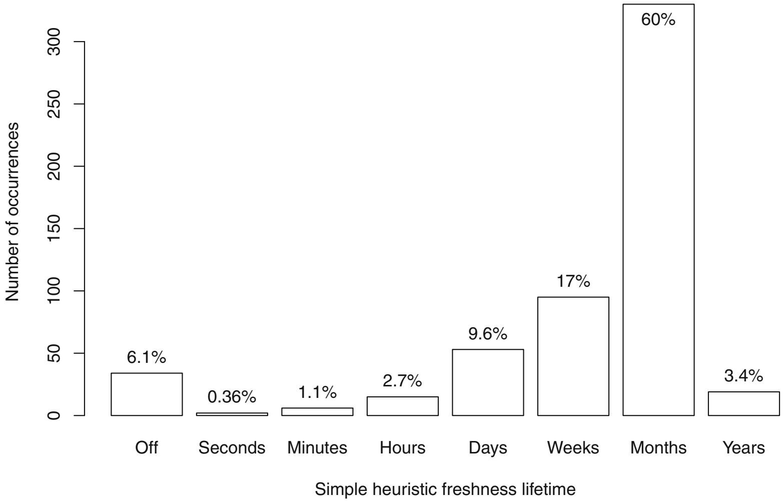

Fig. 3. Barplot counting all simple heuristic freshness lifetimes found. Axes as in Fig. 1.

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 Sect. 4.2.2 in RFC7234 and mentioned in the introduction.

We were able to compute a heuristic lifetime for 554 resources, a larger number than standards-compliant resources. In Fig. 3, we see that the distribution of lifetimes is radically different from the case in Fig. 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:

> [...] We found that 62.2 % of documents didn't change over the six months and found that 51.9 % of domains were considered static.

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

Moreover, by inspecting Fig. 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 $= 0.02$.

[2] http://www.rkbexplorer.com/about/.

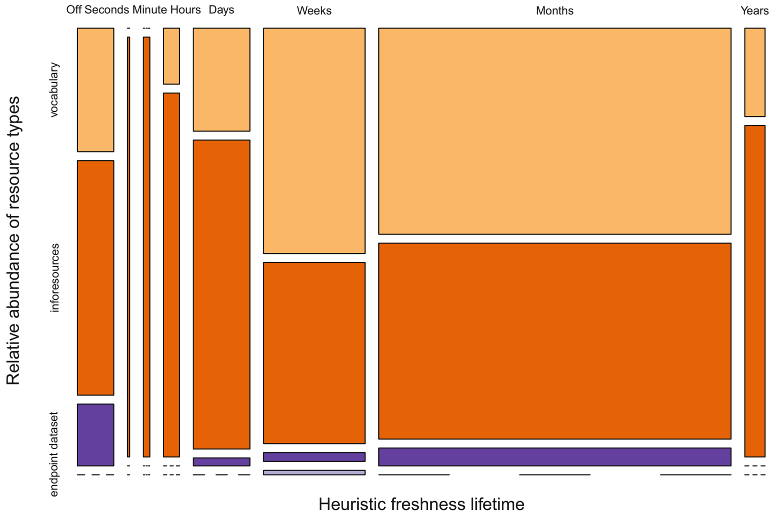

Fig. 4. Mosaic Plot for heuristic freshness lifetime. See caption of Fig. 2 for description.

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.

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.

[3] http://dydra.com/.

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 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 Sect. 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 `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

1. Ager, B., Schneider, F., Kim, J., Feldmann, A.: Revisiting cacheability in times of user generated content. In: INFOCOM IEEE Conference on Computer Communications Workshops, pp. 1–6. IEEE (2010)
2. Baker, T., Vandenbussche, P.-Y., Vatant, B.: Requirements for vocabulary preservation and governance. Libr. Hi Tech **31**(4), 657–668 (2013)

3. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.-Y.: SPARQL web-querying infrastructure: ready for action? In: Alani, H., Kagal, L., Fokoue, A., Groth, P., Biemann, C., Parreira, J.X., Aroyo, L., Noy, N., Welty, C., Janowicz, K. (eds.) ISWC 2013, Part II. LNCS, vol. 8219, pp. 277–293. Springer, Heidelberg (2013)
4. Cyganiak, R., Zhao, J., Hausenblas, M., Alexander, K.: Describing linked datasets with the VoID vocabulary. W3C note, W3C, March 2011. http://www.w3.org/TR/2011/NOTE-void-20110303/
5. Dividino, R.Q., Gröner, G.: Which of the following sparql queries are similar? why? In: CEUR Workshop Proceedings, vol. 1057 (2013)
6. Fielding, R., Nottingham, M., Reschke, J.: Hypertext transfer protocol (HTTP/1.1): caching. RFC 7234 (Proposed Standard), June 2014
7. Fielding, R., Reschke, J.: Hypertext transfer protocol (HTTP/1.1): conditional requests. RFC 7232 (Proposed Standard), June 2014
8. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000)
9. Hogan, A., Gutierrez, C.: Paths towards the sustainable consumption of semantic data on the web. In: AMW. CEUR Workshop Proceedings, vol. 1189 (2014)
10. Ihaka, R., Gentleman, R.: R: A language for data analysis and graphics. J. Comput. Graph. Stat. **5**(3), 299–314 (1996)
11. Jacobs, I., Walsh, N.: Architecture of the world wide web, volume one. W3C recommendation, W3C, December 2004. http://www.w3.org/TR/2004/REC-webarch-20041215/
12. Käfer, T., Abdelrahman, A., Umbrich, J., O'Byrne, P., Hogan, A.: Observing linked data dynamics. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 213–227. Springer, Heidelberg (2013)
13. Käfer, T., Harth, A.: Billion triples challenge data set (2014). http://km.aifb.kit.edu/projects/btc-2014/
14. Kjernsmo, K.: Addendum to a survey of HTTP caching on the Semantic Web. Technical report 444, Department of Informatics, University of Oslo, March 2015
15. Lampo, T., Vidal, M.-E., Danilow, J., Ruckhaus, E.: To cache or not to cache: the effects of warming cache in complex SPARQL queries. In: Meersman, R., Dillon, T., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.-C., Damiani, E., Schmidt, D.C., White, J., Hauswirth, M., Hitzler, P., Mohania, M. (eds.) OTM 2011, Part II. LNCS, vol. 7045, pp. 716–733. Springer, Heidelberg (2011)
16. Martin, M., Unbehauen, J., Auer, S.: Improving the performance of semantic web applications with SPARQL query caching. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part II. LNCS, vol. 6089, pp. 304–318. Springer, Heidelberg (2010)
17. Nottingham, M., Liu, X.: Edge architecture specification. W3C note, W3C, August 2001. http://www.w3.org/TR/2001/NOTE-edge-arch-20010804
18. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 245–260. Springer, Heidelberg (2014)
19. Parreira, J.X., Umbrich, J., Karnstedt, M., Hogan, A.: Hybrid SPARQL queries: fresh vs. fast results. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 608–624. Springer, Heidelberg (2012)

20. Verborgh, R., Sande, M.V., Colpaert, P., Coppens, S., Mannens, E., de Walle, R.V.: Web-scale querying through linked data fragments. In: Proceedings of the Workshop on Linked Data on the Web Co-Located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, 8 April 2014
21. Williams, G.T., Weaver, J.: Enabling fine-grained HTTP caching of SPARQL query results. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 762–777. Springer, Heidelberg (2011)