An Evaluation of Caching Policies for Memento TimeMaps

Justin F. Brunelle & Michael L. Nelson
Old Dominion University
Department of Computer Science
Norfolk, Virginia, 23508
{jbrunelle, mln}@cs.odu.edu

July 20, 2013

Abstract

As defined by the Memento Framework, TimeMaps are ma-chine-readable lists of time-specific copies – called "mementos" – of an archived original resource. In theory, as an archive acquires additional mementos over time, a TimeMap should be monotonically increasing. However, there are reasons why the number of mementos in a TimeMap would decrease, for example: archival redaction of some or all of the mementos, archival restructuring, and transient errors on the part of one or more archives. We study TimeMaps for 4,000 original resources over a three month period, note their change patterns, and develop a caching algorithm for TimeMaps suitable for a reverse proxy in front of a Memento aggregator. We show that TimeMap cardinality is constant or monotonically increasing for 80.2% of all TimeMap downloads observed in the observation period. The goal of the caching algorithm is to exploit the ideally monotonically increasing nature of TimeMaps and not cache responses with fewer mementos than the already cached TimeMap. This new caching algorithm uses conditional cache replacement and a Time To Live (TTL) value to ensure the user has access to the most complete TimeMap available. Based on our empirical data, a TTL of 15 days will minimize the number of mementos missed by users, and minimize the load on archives contributing to TimeMaps.

1 Introduction

The Memento Framework provides HTTP extensions for inter-archive communication and integrating the past and current Web [22, 21, 23]. Memento TimeMaps, part of the Memento Framework, provide an aggregate view of mementos of a URI-R existing in distributed archives as a single document called a TimeMap. These TimeMaps, identified by a URI-T, contain a set of URI-Ms that each have a datetime on which they were archived, or a Memento-Datetime [14]. For example, the TimeMap shown in Figure 1 gives a set of mementos aggregated from two repositories, the National Archives of UK and the Internet Archive's Wayback Machine.

It also spans from the first memento at Dec. 12, 2007 to the last memento at Dec. 14, 2011. Because TimeMaps can aggregate lists of URI-Ms from several sources, many factors can influence the cardinality of a TimeMap, including network downtime, availability of the archive, hardware malfunctions, routine maintenance, etc. Human-induced interruptions in memento availability can also occur. For example, the robots.txt protocol is a redaction method at the Internet Archive (IA) that has been well documented[1]. Copyright and content sensitivity can also cause mementos to be taken out of public access [2, 17]. These redaction requests are legitimate removals of mementos at the content owners' request, and not a failure on the part of the archives[2].

A related modification of archives' offerings is URI migration. For example, the migration of a URI such as `http://example.org/archives/http://thesite.com/` to a URI of `http://memento.example.org/archive/http://thesite.com/` would change all URI-Ms for the archive's contributions to the TimeMap of URI-R. Migrations of this nature assign new URIs to existing mementos.

1.1 Caching TimeMaps

Since TimeMaps do not always improve when they change (e.g., due to archive unavailability), caching policies become important – traditional caches could potentially cache a worse TimeMap than should be available. TimeMaps ideally follow a monotonically increasing growth pattern. That is,

[1] `http://archive.org/post/778/exclusions-from-the-wayback-machine`

[2] `http://www2.sims.berkeley.edu/research/conferences/aps/removal-policy.html`

```
<http://http://mementoproxy.cs.odu.edu/aggr/Timemap/link/http://flare.prefuse.org/>;
    rel="self"; type="application/link-format",
<http://mementoproxy.cs.odu.edu/aggr/timegate/http://flare.prefuse.org/>;
    rel="timegate",
<http://flare.prefuse.org/>;rel="original",
<http://api.wayback.archive.org/memento/20071213002102/http://flare.prefuse.org/>;
    rel="first memento"; datetime="Thu, 13 Dec 2007 00:21:02 GMT",
<http://api.wayback.archive.org/memento/20080509125659/http://flare.prefuse.org/>;
    rel="memento"; datetime="Fri, 09 May 2008 12:56:59 GMT",
<http://webarchive.nationalarchives.gov.uk/20080908074106/http://flare.prefuse.org/>;
    rel="memento"; datetime="Mon, 08 Sep 2008 00:00:00 GMT",
   ...
<http://api.wayback.archive.org/memento/20100815085828/http://flare.prefuse.org/>;
    rel="memento"; datetime="Sun, 15 Aug 2010 08:58:28 GMT",
<http://webarchive.nationalarchives.gov.uk/20100909131056/http://flare.prefuse.org/>;
   rel="memento"; datetime="Thu, 09 Sep 2010 00:00:00 GMT",
<http://api.wayback.archive.org/memento/20101107020354/http://flare.prefuse.org/>;
    rel="memento"; datetime="Sun, 07 Nov 2010 02:03:54 GMT",
```

Figure 1: Example Partial TimeMap for URI-R `http://flare.prefuse.org/`.

TimeMaps should never *lose* mementos (except in rare cases of redaction) – mementos should remain listed in a TimeMap after their first appearance. Since TimeMaps are expensive to generate and change slowly, they are good candidates for caching.

1.2 Contributions

This paper studies a set of 4,000 URI-Rs; 1,000 URI-Rs each came from the Open Directory Project, Delicious, Bit.ly, and search engine caches [1]. We observed the TimeMaps of the 4,000 URI-Rs for a 3-month period (from May 1st to July 31st, 2012 for a total of 92 days) and we mapped and evaluated their evolution and change patterns. This paper shows that TimeMap cardinality is monotonically increasing (i.e., the same or increasing) for 80.2% of all TimeMap changes in the observation period.

We also proposes a new caching algorithm utilizing a TTL value for caching Memento TimeMaps. This caching algorithm exploits the knowledge that TimeMaps sometimes do not change "for the better" and therefore should not always be cached. The change patterns of the observed TimeMaps are used to empirically determine the optimal TTL value.

2 Related Work

It is important to understand the behavior of TimeMaps and the associated availability of mementos. For example, the Warrick project [12] uses TimeMaps to recover mementos with the ultimate goal of recovering lost websites. Obviously, understanding how archives advertise their mementos and understanding how the availability changes is important for several aspects of Warrick. Caching TimeMaps is used to balance load on the archives vs. Warrick's need for the most recent mementos.

Due to the popularity of Memento, other services and tools have begun to consume and rely on TimeMaps to provide functionality. For example, the UK Web Archive is providing a visualization tool for TimeMaps [25]. Additionally, Android's Memento browser [18] and an iOS mobile memento browser [20] all rely on TimeMaps to provide time travel for the Web.

Observing and studying change on the Web is not new; several works have observed the changes that Web pages undergo over time, proving their ephemeral nature [6, 7, 4, 8, 16, 15]. In contrast to these studies, TimeMaps can be thought of as Web resources that cannot be evaluated as "fresh" solely on their age or change status.

Disk failures are one reason for changes in memento availability. A case study of failures at the Internet Archive (IA) is provided in Schwarz's 2006 work [19]. The organization of archives is also important; we must understand how an entire server failure affects the availability of mementos. Jaffe's work in 2004 describes the IA's architecture [11].

Caching is an important addition to the Web. It reduces latency, load, and user wait times. A set of caching methods commonly utilized on the Internet – including TTL values – are discussed in Wang's 1999 survey paper [24]. Other work has been done to benchmark the performance boosts achieved when implementing caching of dynamic contents [10]. Other works have studied best efforts for caching dynamically generated content, such as that generated on the server-side [5, 27].

Reverse proxies such as Squid [26] improve the efficiency of Web traffic by caching content. This caching is independent of the content server and requester, allowing both to operate without modification. The work we present can be implemented using a reverse proxy (e.g., Squid allows custom caching rules) or by modifying the Memento aggregator cache, and will reduce the load on the Memento proxies while increasing their reliability for users.

3 Experiment design

We observed TimeMap cardinality (number of mementos present in a TimeMap) daily for 4,000*92=386,400 total download attempts and the consistency of URI-Ms and the datetime values associated with a memento throughout the experiment. Once a day, a set of scripts downloaded the 4,000 TimeMaps so that we can analyze the daily changes of the mementos advertised in the TimeMaps, as well as how the contents of the TimeMaps differ between observations. To ensure the 4,000 TimeMaps could be retrieived in a 24 hour period, the scripts used a timeout for accessing the TimeMaps. A 45 second timeout was used because this is the upper limit that humans are willing to wait when accessing a resource [13]. The experiment also ran 11 parallel scripts to access different portions of the 4,000 URI dataset. With this concurrency and limited wait time, all 4,000 TimeMaps were accessed consistently within a 24 hour period.

3.1 Cache-less Memento Proxies

The first step in the experiment setup was to create and install a new set of Memento proxies separate from the production proxies at Los Alamos National Laboratory and Old Dominion University. The existing proxies are constantly being accessed by MementoFox, users, and other experiments. We installed a set of proxies on a separate, private experiment machine (`128.82.7.240`) to prevent any contamination from the public Memento uses. The proxies were installed just as they exist on the production Memento machines. However, the production proxies cache their responses in an infinitely-sized cache with a TTL=∞ to speed up the response time to the users, as well as to prevent unnecessary load on the archives. The caching software had to be removed from the experiment proxies to ensure fresh results of each observation. These memento proxies queried for, constructed, and served the TimeMaps throughout the experiment run.

3.2 TimeMaps as Web Resources

TimeMaps can change, evolve, and grow like traditional Web resources. However, TimeMaps differ from traditional Web resources in that they do not always change "for the better." Traditional Web resources can be maintained in a cache by monitoring any change to the resource content. TimeMaps can change in a detrimental way – they sometimes lose mementos due to intermittent contributions from archives, URI changes, or archival redactions. Archival redactions will results in a HTTP 404 response when dereferencing a URI-M but redactions are very rare. We are more concerned about transient errors. As such, TimeMaps cannot be cached in the same manner – only TimeMaps that have been changed in a positive way should be updated in a cache. Prior to this work, there was no method to designate a positive or negative change to a TimeMap.

The production Memento proxies utilize caching to limit the load on the archives and increase response time to users. The proxies are research prototypes, and as such, the caching algorithm for Memento was designed around one of two simple solutions. The two simplest caching algorithms are to cache everything or cache nothing. Caching nothing induces unneeded and unfair load on the archives contributing to the TimeMaps and increases the service time for users due to the latency between the Memento Proxies and archives. Therefore, the cache everything approach was taken. The proxies cache the first TimeMap observed in system and holds it until a cleansing of the cache is performed manually. If a *bad* TimeMap is cached, it persists in the cache until the cache is manually cleaned. TimeMaps that are entirely empty (i.e., receive an HTTP 404 response) are especially bad (the TimeMaps show that a URI-R is not archived, when in fact, it may have mementos that were not reported), and are discussed in more depth in Section 4.1.

3.3 Experiment Execution

During the experiment, three outages were observed as noted by the red circles in Figure 2. Annotation 1 in the figure indicates where the locally installed proxies were inoperable from May 16 – May 18 due to a system reset of the department machines. As indicated by 2 in the figure, the Internet Archive proxies were inoperable due to edits to the API occurring at the Internet Archive. As indicated by 3 in the figure, there was a massive power failure at at our university that caused the machines to automatically reboot, killing the experiment run. This failure went undetected for 6 days in June.

We took these time periods of low memento availability into consideration during the calculations of our results. The previous TimeMaps were substituted for the missing TimeMaps, effectively treating them as unchanged instead of non-existent.

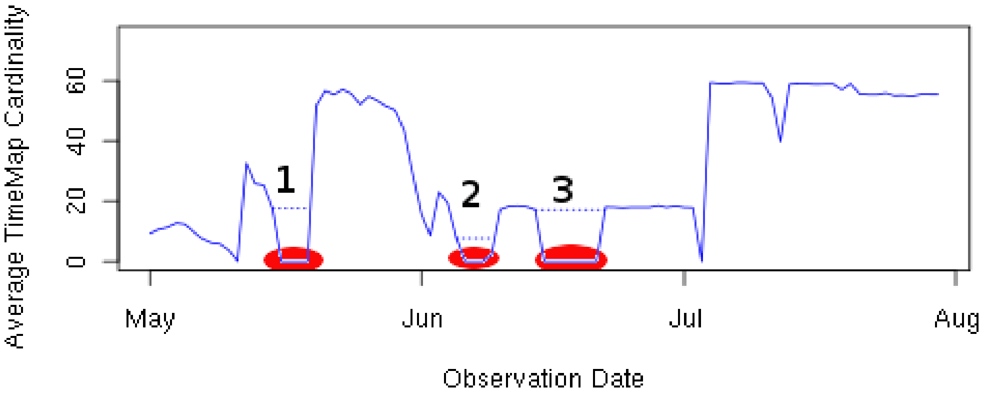


Figure 2: Average TimeMap Cardinality during experiment execution (2012).

4 Experiment Results

It was immediately clear that TimeMaps were not monotonically increasing. That is, TimeMaps sometimes get smaller – or "worse" instead of staying the same or growing.

4.1 TimeMap Change Types

Utilizing the collected TimeMaps over the course of the observation period, we categorized the changes to the TimeMaps. We used the classifications in Table 1 to determine how changes affected the TimeMaps. The table uses a to denote the number of archives that contribute to a TimeMap at time t and *a'* to denote the number of archives that contribute to a TimeMap at time *t+1*. Similarly, the table uses m to denote the number of mementos in the TimeMap at time t and *m'* to denote the number of mementos in the TimeMap at time *t+1*.

TimeMaps can lose and gain mementos and contributing archives. This experiment utilizes the change patterns of TimeMaps to predetermine the change rates for the observed URI-Rs. The number observed changes are provided in Figure 3. Most TimeMaps changed 3 or 4 times over the course of the experiment.

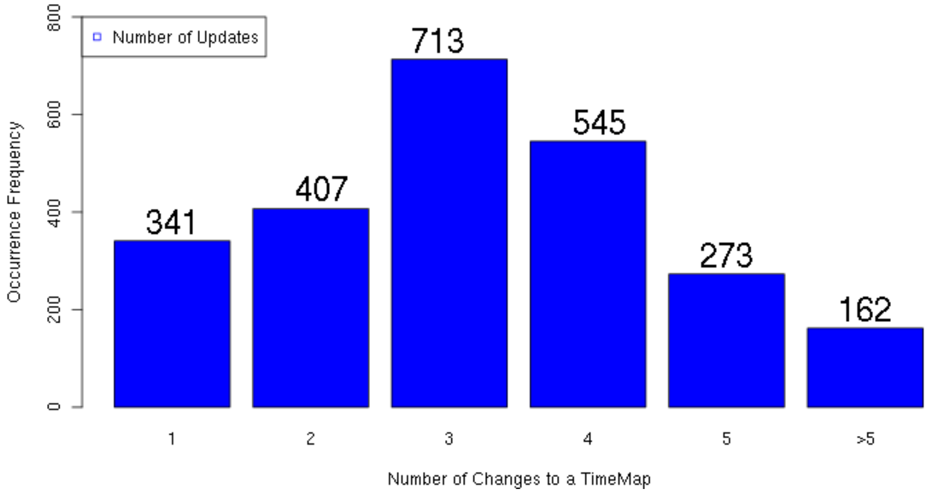


Figure 3: Number of changes to TimeMaps.

As shown in Figure 3, most TimeMaps underwent 3 changes throughout the observation period. Observing these change rates provides insight into the most appropriate TTL values for the TimeMaps. It also disregards TimeMaps that were always empty during the collection part of the experiment. On average, the TimeMaps observed in this experiment changed every 37.6 days, with the most frequently occurring change rate observed of every 30 days, which roughly equates to 3 changes throughout the observation period.

Mementos can be added by contributing archives, or mementos can be redacted by contributing archives. Contributing archives can also disappear from a TimeMap due to service interruption or other unavailability. New archives can begin contributing to a TimeMap if a new URI-R is added to the collection. Cases 6 and 7 are the most detrimental to a TimeMap – the overall cardinality of the TimeMap is reduced by the loss of mementos. Case 1 is the most observed case, and represents the TimeMap remaining consistent between two times. Cases 2, 3, 4, and 5 all result in additional mementos being added and are therefore improvements upon the TimeMap at the previous time. Case 4 is unique in that an archive is lost but there are still new mementos in the TimeMap; we lean toward updating the cache in Case 4. Given that Case 6 is the most frequently observed change to TimeMaps, we show that if a TimeMap changes, it is most often for the worse. The occurrence of cases throughout the experiment is provided in Table 1. These classifications of TimeMaps establish the notion of *better* and therefore how to handle cache replacement.

4.2 Definition of TimeMap Changes

The cardinality of a TimeMap at observation time t is shown as the red lines in Figures 5 - 8. The blue lines are the cumulative size of the set of all unique URI-Ms up to time t: [0, t].

A memento is defined in Equation 1 as a version of a URI-R at time k.

$$\begin{aligned} m_{R,k} = & \quad (URI-R, k), \\ & k \in \{\text{times in RFC 1123 format [9]}\} \end{aligned} \tag{1}$$

The URI that identifies a memento is defined in Equation 2.

$$URI - M_{R,k} = \text{URI of } m_{R,k} \tag{2}$$

TimeMaps are composed of mementos of a URI-R, as defined in Equation 3.

$$TM_{R,t} = \bigcup_{k} URI - M_{R,k} \tag{3}$$

The cardinality of a TimeMap $TM_{R,t}$ is defined in Equation 4. Note that this refers to the number of unique URI-Ms that appeared in a URI-R's TimeMap as observed at time t.

$$|TM_{R,t}| \tag{4}$$

Thus, monotonically increasing TimeMaps would satisfy the condition in Equation 5.

$$|TM_{R,t_1}| \leq |TM_{R,t_2}|, \text{where } t_1 < t_2 \tag{5}$$

4.3 Strict versus Loose Policy

When comparing URI-Ms in a TimeMap to determine how TimeMaps grow over time, lexigraphically comparing URI-Ms is the intuitive method. We will refer to this method of matching as the *Strict Policy.*

The best possible set of URI-Ms is the cumulative set of all URI-Ms that have been observed in the TimeMaps until a time t. This represents the best possible TimeMap we would see if TimeMaps were monotonically increasing. This is represented by Equation 6.

$$M_{strict} = \bigcup_{t} TM_{R,t} \tag{6}$$

URIs are not expected to change [3], but, due to archives being restructured or changes in the URI-M structure, new URIs identifying existing mementos sometimes occur. The *Loose Policy* only matches the tuple of Memento-Datetime and archive ($\mathit{archive}(URI - M_{R,k})$, k) of a URI-M and the URI-R for which it is a memento. The archive is determined by the hostname and occasionally path of the URI-M. The *Loose Policy* is immune to URI changes and should recognize mementos identified by different URI-Ms across time, testing the theory that when mementos disappear from TimeMaps, they actually just change URI-Ms, and not Memento-Datetimes. Conditions under which this occurs include architecture modifications by the archives, URI scheme changes, or even errors with the URI-M listed.

TimeMap cardinality under the *Loose Policy* is measured in accordance with Equation 7.

$$M_{strict} = \begin{array}{c} \bigcup_{t} TM_{R,t} \\ \forall \text{unique values of } \mathit{archive}(URI - M_{R,k}) \text{ and } k \end{array} \tag{7}$$

For example, the TimeMap in Figure 4, the URI-Ms would *Loosely* match the other URI-Ms with the same Memento-Datetimes. Since they have the same archive (`web.archive.org`) and Memento-Datetime (`Mon, 01 Nov 2010 06:02:04 GMT`). All would be considered lexigraphically different and not match according to the *Strict Policy* because of the same Memento-Datetime but different URI-R (`http://aarp.org:80/Health/`).

Comparisons of cumulative mementos under the *Loose* and *Strict Policies* show that mementos receive new $URI - M_{r,k}$ but are still the same memento

Case #	$\mid a \mid$	$\mid m \mid$	Description	Cache Action	Occurrence
1	a = a′	m = m′	Same TimeMap.	Do not update	77.4%
2	a <a′	m <m′	New archives and new mementos.	Update	2.4%
3	a = a′	m <m′	New mementos, same archives.	Update	0.4%
4	a >a′	m = m	Lost an archive, but an archive gained mementos.	Update	0.01%
5	a >a′	m <m′	Lost an archive, but gained more mementos.	Update	0.01%
6	a >a′	m >m′	Lost an archive and lost mementos.	Do not update	19.7%
7	a <a′	m >m′	Gained archives, but lost mementos.	Do not update	0.1%

Table 1: Classifications of TimeMap Changes

$m_{r,k}$. The *Loose Policy* graphs do not show an increase when this activity takes place, while the *Strict Policy* mistakes this new $URI - M_{r_k}$ as a new memento.

The two graphs of the yardsellr.com TimeMap, (Figures 5(b) and 5(a)) and whitehouse.gov TimeMap, (Figures 6(a) and 6(b)) both demonstrate well-behaved TimeMaps in which the *Loose* and *Strict Policy* graphs match one another. This shows that there is no difference between the TimeMap cardinalities according to the *Loose* and *Strict Policies.* However, these two TimeMaps change in different ways during the observation period. The yardsellr.com TimeMap incurs a spike of mementos in mid-May due to an influx of mementos from the Internet Archive. These mementos do not reappear in the TimeMap for the duration of the observation period, which results in a gap between the cumulative and observed mementos (observed as the gap between the blue and red lines, respectively). Alternatively, the whitehouse.gov TimeMap frequently has observed cardinality equal to the cumulative cardinality, thus showing that the maximum number of mementos frequently appears in the TimeMap. The consistent whitehouse.gov TimeMap demonstrates expected behavior from a TimeMap. There are some slight dips in TimeMap cardinality caused by an archive being temporarily unavailable. These dips are not permanent, as seen in the yardsellr.com TimeMap.

An additional scenario can occur when a memento, or subset of mementos, disappears from the TimeMap altogether. This is observed by the dip representing the loss of URI-Ms from the Google Code API TimeMap on May 22nd in Figures 8(a) and 8(b). This is noteworthy because there is a complete swapping of mementos that occurs. While the observed cardinality decreases, there is an increase in the cumulative number of mementos (the red line goes down while the blue line goes up). This is due to the swapping of an entire set of URI-Ms.

```
<http://web.archive.org/web/20101101060204/http://aarp.org:80/Health/>;
rel="memento";datetime="Mon, 01 Nov 2010 06:02:04 GMT",
<http://web.archive.org/web/20101101060204/http://www.aarp.org:80/Health/>;
rel="memento";datetime="Mon, 01 Nov 2010 06:02:04 GMT",
<http://web.archive.org/web/20101101060204/http://www.aarp.org:80/health/>;
rel="memento";datetime="Mon, 01 Nov 2010 06:02:04 GMT",
```

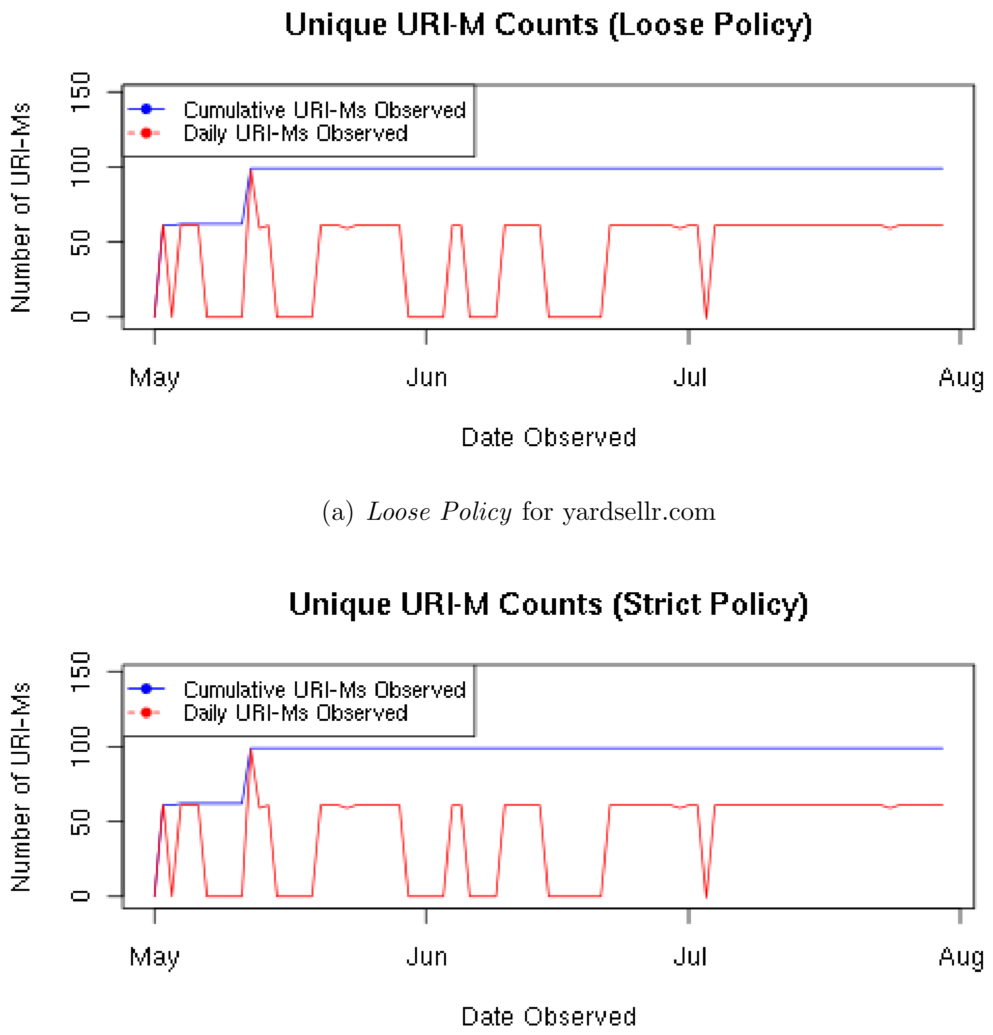
Figure 4: A subset of an example TimeMap for URI-R `http://aarp.org/Health/`.

Google Translate has a TimeMap cardinality $|TM| = 5{,}800$. These mementos are populated from several different archives, but the majority of URI-Ms come from the Internet Archive and from Archeif Web. The TimeMap size over time is provided in Figures 7(a) and 7(b) using the *Strict* and *Loose Policies*, respectively. These graphs differ greatly in their representation of the TimeMap sizes.

When using the *Strict Policy*, the TimeMap appears to be nearly consistently growing in cardinality (the blue line, and number of total unique URI-Ms observed over the observation period), while showing only slight (nearly static) growth in the cardinality of the TimeMaps observed on a given day. This is due to URI-Ms with the same Memento-Datetime appearing in a TimeMap for a short span of time and never returning.

The *Loose Policy* provides completely different results. As mentioned previously, the *Loose Policy* uses only the combination of URI-M's Memento-Datetime and the hosting archive as an identifier. The size of the daily observed URI-Ms rises above the total number of observed Memento-Datetimes. This is observed to be an error in the Archeif Web proxy – the proxy was providing `00:00:00` as the time portion of the Memento-Datetime for all mementos returned, causing a discrepancy between the *Strict* and *Loose Policies*.

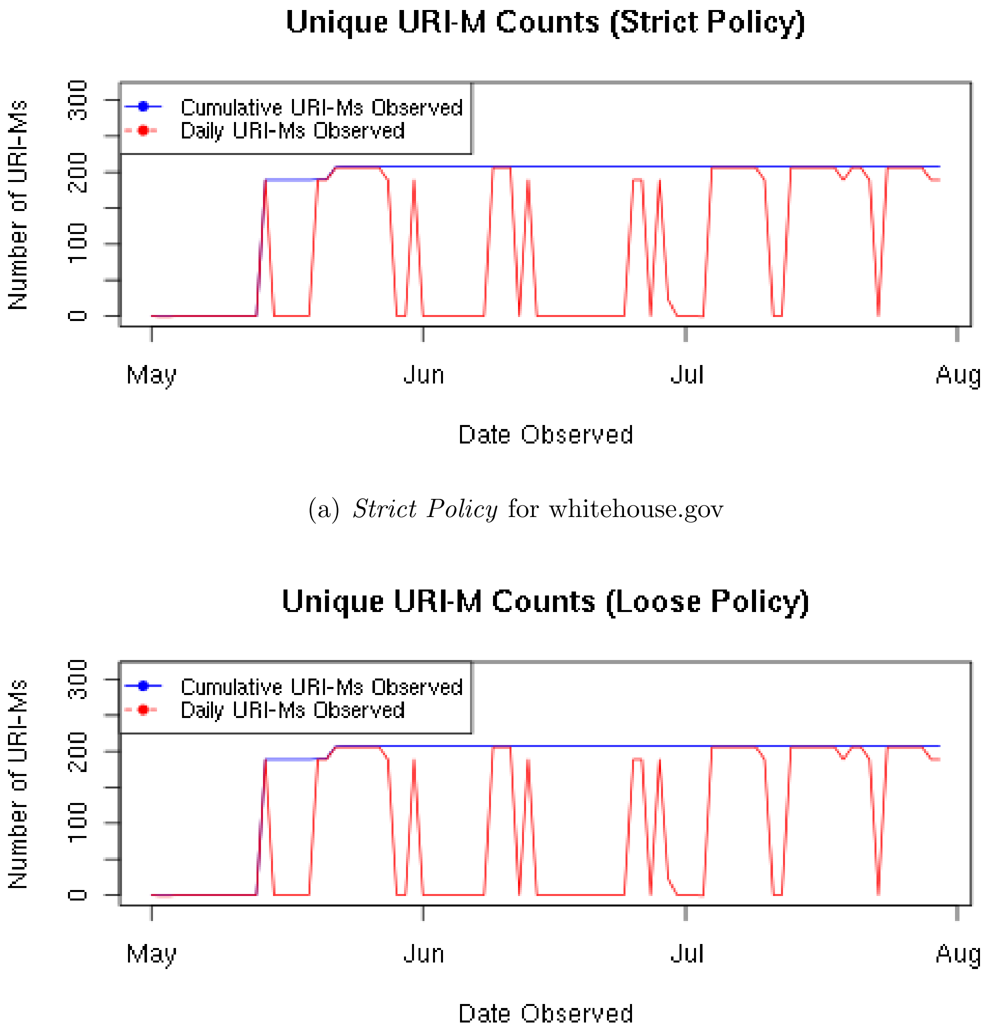
Notice that some Memento-Datetime values are replicated, even though the URI-M is not. This is due to the simultaneous capture of resources by the archives. This produces a discrepancy between the *Strict* and *Loose Policies*.



(a) *Loose Policy* for yardsellr.com

(b) *Strict Policy* for yardsellr.com

Figure 5: The TimeMap of `http://yardsellr.com` shows identical graphs for *Strict* and *Loose Policies*, and demonstrates that mementos blink on and off in the TimeMap, but do not change the overall cumulative set of all observed mementos.



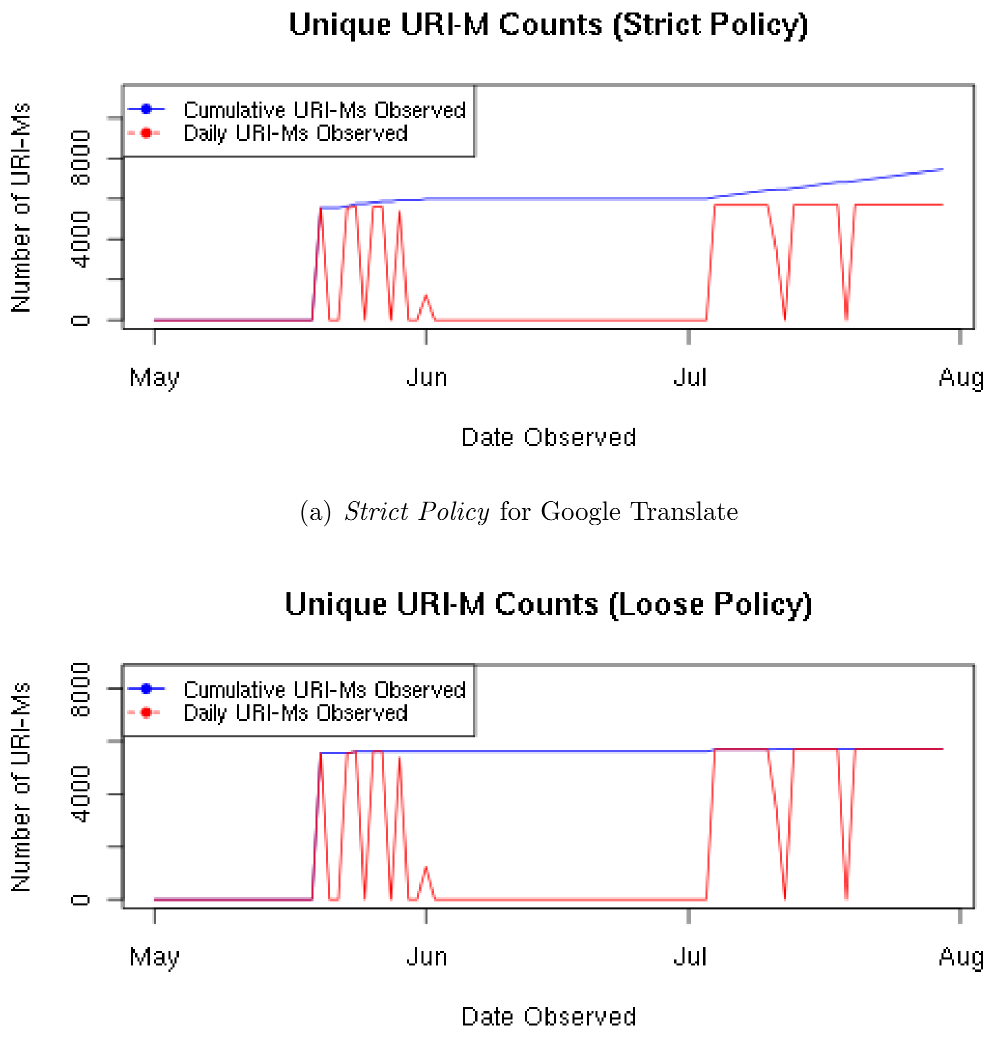
(a) *Strict Policy* for whitehouse.gov

(b) *Loose Policy* for whitehouse.gov

Figure 6: The `http://www.whitehouse.gov/administration/eopcea/` TimeMap is well-behaved, with identical *Strict* and *Loose Policy* graphs.

5 Evaluation

As observed in Section 4, Memento TimeMaps do not always change for the better. TimeMap cardinality can decrease due to various external influences.



(a) *Strict Policy* for Google Translate

(b) *Loose Policy* for Google Translate

Figure 7: TimeMap with duplicate Memento-Datetime values as shown by the *Strict* and *Loose Policies* for `http://translate.google.com/`.

The Memento framework proxies currently have a cache with an *infinite* TTL – TimeMaps are cached on their first occurrence and never automatically replaced. This optimization was critical during the initial deployment of the research prototype Memento Aggregaters.

We tested a continuum of TTL values to find the best life span of an entity in the cache. A TTL=0 will maximize the freshness of the TimeMaps in the cache, but will also maximize the load on the archives and unnecessarily delay consumer applications. A TTL=∞ will minimize the load on the archives, but also minimize the freshness of the TimeMaps in the cache. Therefore, caching a bad TimeMap is especially bad if we are unfortunate enough to receive a TimeMap with a cardinality of 0, which will remain in the cache until the cache is cleared.

We tested TTL=n, where n={0...92} with 92 effectively equaling ∞. We assume the Aggregater can store an infinite number of TimeMaps and focus only on the possible invalidation of a single TimeMap when its TTL has expired.

5.1 Caching Policies

To determine the most effective cache policy, we tested the *current*, *unconditional*, and *conditional* policies. The *current* caching policy simulates the operation of the Memento Aggregater's current cache with a TTL=∞ to never replace TimeMaps. The *unconditional* policy employs TTL values of 0, ∞, and n. TTL values set a time limit for cache replacement – items in a cache are not replaced or removed until a set time limit expires. This policy is used as the baseline measurement since it does not exploit knowledge of the TimeMaps' contents. The *conditional* policy employs TTL values of 0, ∞, and n, but only replaces TimeMaps in the cache when they have improved according to the *Loose Policy.* The Cases 2-5 are cases that will update an entry in the cache because the TimeMap gains mementos (according to Table 1).

5.2 Evaluation Measures

To measure the impact and quality of the cache replacement policies, the MemDays penalty is defined in Equation 8. The MemDays penalty is the sum of the number of mementos that are missed due to a cache hit when there is a better version of the TimeMap available over the time t of the experiment.

$$MemDays = \sum_{1}^{t} max((|\text{TM}_{live}\text{— - }|\text{TM}_{cache}\text{—}),\ 0) \tag{8}$$

MemDays provides a cumulative measure of the detrimental effects of the cache. This metric is the sum of fresh mementos missed for each day those mementos are not available in the cache. An example is provided in Figure 9. A TimeMap with $|TM| = 6$ is cached at time t=0. At t=1, $|TM|$=6. Since $|TM|$ remained consistent, the cumulative MemDay measure is 0. However, $|TM|$ increases to 8 at time t=2. Since the cached $|TM|$=6, the MemDay measure increases to 2 since 8-6=2. At time t=3, the difference between the cached and live $|TM|$ is still 2, increasing the cumulative MemDay to 4.

At time t=4, the live $|TM|$=8. The difference between the cached and live $|TM|$ is now 4, increasing the MemDay by 4 to 8. At t=5, the cache is updated with the new TimeMap $|TM|$=8, and therefore the cached and live $|TM|$ are equal, adding no additional MemDay penalty.

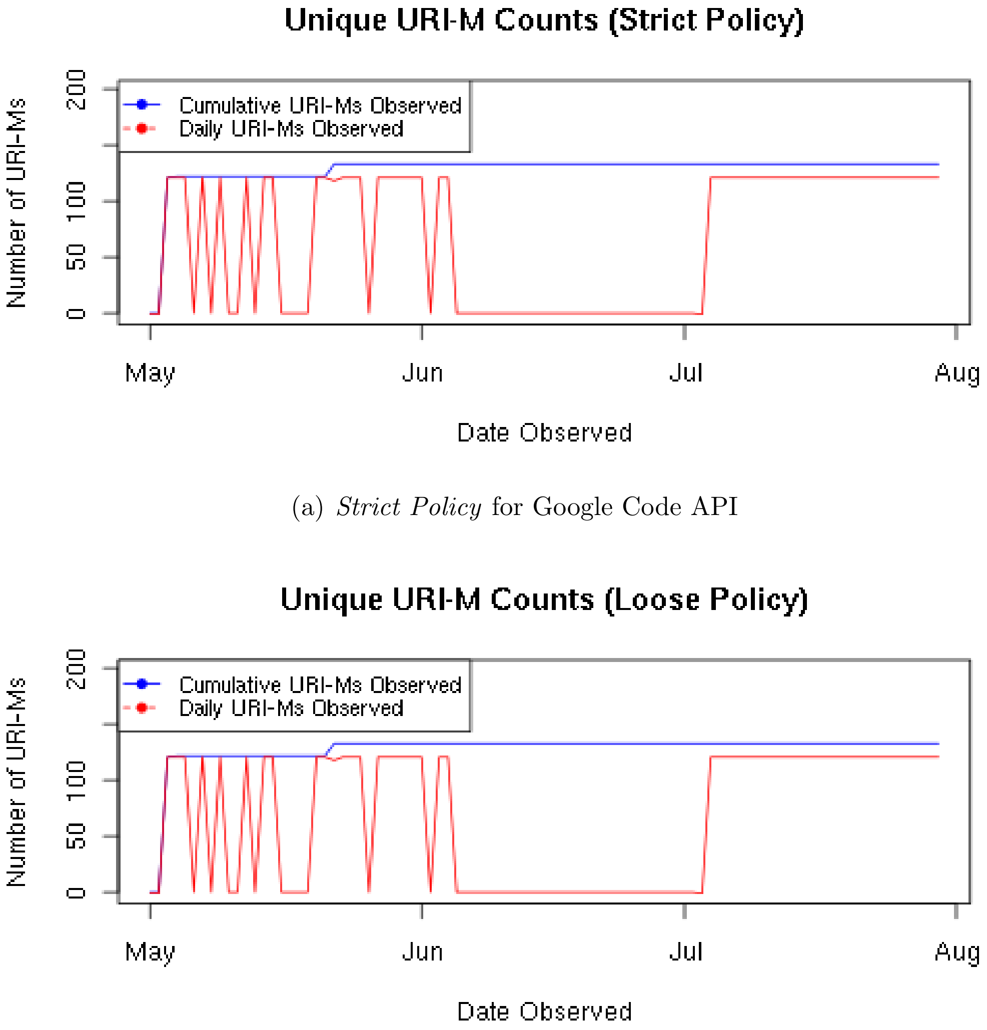
In this example, Q=2 because the archives contributed to the cached version twice at t=0 and t=5. The final accumulations of MemDay=8.

The complement to MemDays is the load on the repositories, or amount of queries Q (Equation 9). This is the sum of the number of times the cache requests a TimeMaps from the archives over the course of t days in the experiment, effectively measuring the cache misses due to expired TTL values. Notionally, this is inversely related to the MemDays measure in Equation 8. An optimal caching strategy can be determined by minimizing load on the Q and MemDays simultaneously.

$$Q = \sum_{1}^{t} \text{Cache Misses} \tag{9}$$

5.3 Results

At the conclusion of this experiment, we calculated the average number of TimeMap observations in which we observed a monotonically increasing trend. Only 80.2% of all observations either remained the same or provided an improvement to the existing TimeMap (Figure 3).



(a) *Strict Policy* for Google Code API

(b) *Loose Policy* for Google Code API

Figure 8: Poorly formed Memento-Datetimes appear in the TimeMaps for `http://code.google.com/apis/g/data/client-cs.html`

To recreate a series of TimeMaps, we used a three-month-long observation period to test the cache replacement policies. We ran a simulation to access the 4,000 TimeMaps with the three caching policies implemented with the

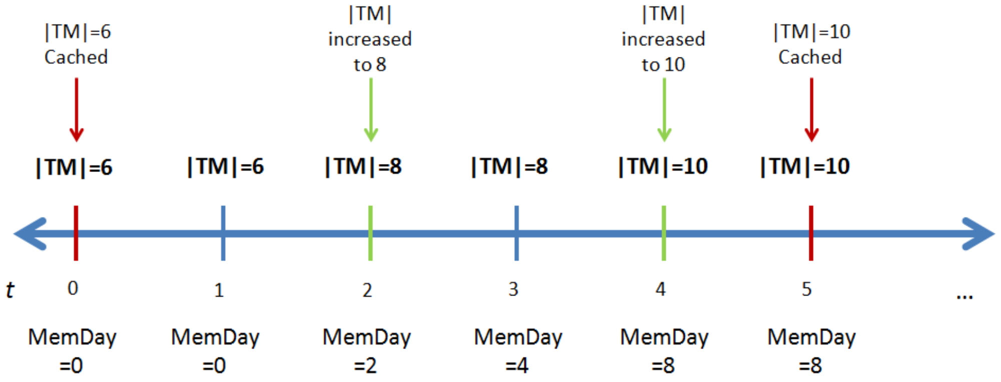


Figure 9: MemDay calculation example.

Loose Policy. The cache size is unlimited for the purposes of this experiment; our goal was only to test the behavior of the replacement policies as they relate to TimeMap change patterns.

5.3.1 Missed Mementos

As with any cache replacement policy, it is necessary to determine how many updates are missed by utilizing a cache. Since the concept of replacement in this unique cache replacement experiment is limited to those TimeMaps that are *better* (based on the cases in Table 1) than the currently cached version, this experiment counts only those replacements that would result in an improved TimeMap being placed in the cache. The results of this experiment are provided in Figure 10.

A primary problem with the current caching algorithm is that TimeMaps with 0 mementos might be cached when TimeMaps with more than 0 mementos exist at another time. False 0-sized TimeMaps (i.e., a 404 response for a TimeMap) are especially detrimental to consumers of TimeMaps since this suggests that a URI-R is not archived when, in fact, it has mementos. This experiment measures the occurrence of these false 0-sized TimeMaps. Additionally, given the rudimentary caching strategies available, a false 0-sized TimeMap should not replace a TimeMap that lists mementos. This is a particularly detrimental occurrence of Case 6 (Table 1 – lost an archive and lost mementos) to consumers of TimeMaps.

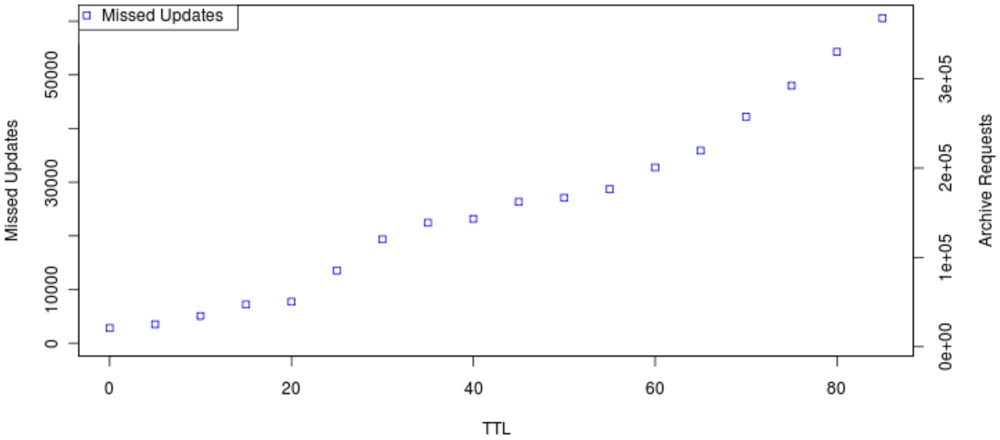


Figure 10: Number of missed cache replacements by TTL value.

As shown in Figure 11, the best TTL value to minimize the false 0-sized TimeMaps is 0. However, this induces an unnecessarily large load on the archives. This shows that TimeMaps normally remain constant. However, when they do change, they tend to get worse, as shown in Figure 3.

5.3.2 MemDays versus Q

The primary measure used in this experiment to determine how well the cache replacement policies are performing is the MemDay penalty (Section 5.1). Using *unconditional*, our experiment shows the MemDay penalty-saved trade-off in Figure **??**. The MemDays for *conditional* are shown in Figure **??**. Notice that *conditional* shows improvement over *unconditional* by accumulating fewer MemDays during the simulation. This is because *unconditional* has the potential to cache a TimeMap with $|TM|$ less than the one in the cache, while *conditional* ensures that the cached TimeMap has the highest $|TM|$.

As expected, lower TTL values result in fewer missed updates. However, lower TTL values lead to additional queries (Q) to the archives.

In *unconditional*, running a cache that updates TimeMaps at every transaction (TTL=0) results in MemDays=98,312, but Q=368,000 (368,000 re-

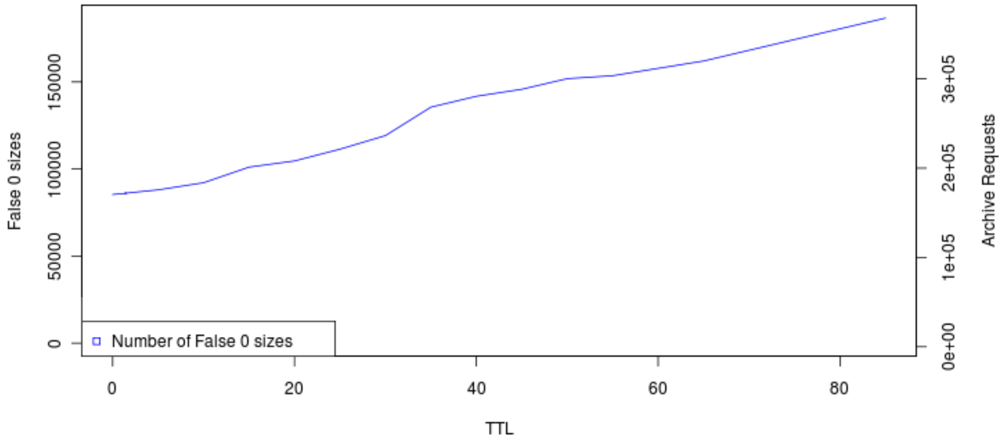


Figure 11: False 0-sized TimeMaps by TTL value for *conditional*.

quests made to the archives to update the cache). That is, 98,312 mementos are missed because the cache was updated with worse, lower-cardinality TimeMaps. When TTL=∞, MemDays=4,010,406 and
Q=4,000. TTL=∞ is the *current* policy of the Memento Aggregater.

The optimal TTL value for *unconditional* can be found at the intersection of the Q line (red), and the MemDay line (blue) in Figure **??**. At this point, MemDays=784,895 due to cache staleness, but Q=92,000. Subsequent data produces a worse trade off between requests to the archives and mementos missed by the users. The data shows that the best TTL value for a TimeMap cache, located at the intersection of the MemDay and Q lines, is 10 days.

In *conditional*, running a cache that updates TimeMaps at every transaction (TTL=0) results in MemDays=0 (or 0 mementos being missed), but Q=368,000 (368,000 requests made to the archives to update the cache). The *conditional* policy improves upon *unconditional* by 98,312 MemDays when TTL=0. Because *conditional* ensures the TimeMap with the highest cardinality remains in the cache, there are no MemDays incurred, as opposed to *unconditional* which blindly assumes improvement to TimeMaps when they change; this is an improper assumption as shown by the change patterns in Table 1. When TTL=∞, MemDays=4,010,406 and Q=4,000, identical to *unconditional*; in both policies, a TTL=∞ caches the first TimeMap available and never replaces it, accumulating MemDays for the duration of the experiment.

The optimal TTL value for *conditional* can be found at the intersection of the Q line (red), and the MemDay line (blue) in Figure **??**. At this point, MemDays=588,368 due to cache staleness (only 7,207 missed TimeMap updates), but Q=23,000. Subsequent data produces a worse trade off between requests to the archives and mementos missed by the users. The data shows that the best TTL value for a TimeMap cache, located at the intersection of the MemDay and Q lines, is 15 days. The optimal TTL for *conditional* improves upon the optimal TTL for *unconditional* by reducing Q by 69,000 and MemDay by 196,527 due to the ability to only update the cache with *better* TimeMaps. This is also intuitively satisfying since TimeMaps change less frequently (every 30 days on average shown in Figure 3) than our optimal TTL of 15 days.

6 Conclusions

The study outlined in this paper observed 4,000 TimeMaps for URI-Rs over the course of three months and measured their consistency over time. We show that TimeMaps are monotonically increasing only 80.2% of the time. Due to this evidence, a new *conditional* caching algorithm must be implemented to ensure users have access to the best possible TimeMaps. The current Memento Aggregater operates with a TTL=∞, which produces MemDays=4,010,406 and Q=4,000; We have proposed a caching algorithm that implements a TTL value of 15 days and a *conditional* replacement policy in which responses that are smaller than previous responses are not cached. TTL=15 produces MemDays=588,368 and Q=23,000. This shows an improvement on the current Memento Aggregater operation by 3,422,038 MemDays while only making 19,000 more queries to the archives. It improves upon the *unconditional* policy by an observed Q=69,000 and MemDay=196,527.

7 Acknowledgments

This work supported in part by the NSF (IIS 1009392) and the Library of Congress. We also thank Robert Sanderson (LANL) for the Memento Aggregater research prototype and Ahmed AlSum (ODU) for his assistance with the cache-less Memento Proxies. Memento is a joint project between the Los Alamos National Laboratory Research Library and Old Dominion

University

References

[1] S. Ainsworth, A. Alsum, H. SalahEldeen, M. C. Weigle, and M. L. Nelson. How much of the Web is archived? In *JCDL '11: Proceedings of the 11th annual international ACM/IEEE Joint Conference on Digital Libraries*, pages 133–136, 2011.

[2] Alyssa N. Knutson. Proceed With Caution: How Digital Archives Have Been Left in the Dark. http://www.btlj.org/data/review/24-437-473.pdf, 2009.

[3] T. Berners-Lee. Cool URIs don't change. http://www.w3.org/Provider/Style/URI, 1998.

[4] B. Brewington, G. Cybenko, D. Coll, and N. Hanover. Keeping up with the changing Web. *IEEE Computer*, 33(5):52–58, 2000.

[5] P. Cao, J. Zhang, and K. Beach. Active cache: caching dynamic contents on the web. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 373–388, 1998.

[6] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the 26th international conference on very large data bases*, pages 200–209, 2000.

[7] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul. Rate of change and other metrics: a live study of the World Wide Web. In *USITS'97: Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, USITS'97, 1997.

[8] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. *Software: Practice and Experience*, 34(2):213–237, 2004.

[9] Internet Engineering Task Force. Requirements for Internet Hosts – Application and Support, October 1989. http://www.ietf.org/rfc/rfc1123.txt.

[10] A. Iyengar and J. Challenger. Improving web server performance by caching dynamic data. In *USITS'97: Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, USITS'97, 1997.

[11] E. Jaffe and S. Kirkpatrick. Architecture of the Internet Archive. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, 2009.

[12] F. McCown, J. A. Smith, M. L. Nelson, and J. Bollen. Lazy preservation: reconstructing websites by crawling the crawlers. In *WIDM '06: Proceedings of the 8th annual ACM international workshop on Web information and data management*, pages 67–74, 2006.

[13] F. F. Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour Information Technology*, (3):153–163.

[14] M. L. Nelson. Memento-Datetime is not Last-Modified. http://ws-dl.blogspot.com/2010/11/
2010-11-05-memento-datetime-is-not-last.html, 2011.

[15] M. L. Nelson and B. D. Allen. Object persistence and availability in digital libraries. *D-Lib Magazine*, 8(1), January 2002.

[16] A. Ntoulas, J. Cho, and C. Olston. What's new on the web?: the evolution of the web from a search engine perspective. In *Proceedings of the 13th international conference on World Wide Web*, pages 1–12, 2004.

[17] Peter B. Hirtle. Digital Preservation and Copyright. http://fairuse.stanford.edu/
commentary_and_analysis/2003_11_hirtle.html, 2012.

[18] R. Sanderson, H. Shankar, S. Ainsworth, F. McCown, and S. Adams. Implementing Time Travel for the Web. *Code4Lib Journal*, 13, 2011.

[19] T. Schwartz, M. Baker, S. Bassi, B. Baumgart, W. Flagg, C. van Ingen, K. Joste, M. Manasse, and M. Shah. Disk failure investigations at the Internet Archive. *14th NASA Goddard, 23rd IEEE Conference on Mass Storage Systems and Technologies*, May 2006.

[20] H. Tweedy, F. McCown, and M. L. Nelson. A Memento Web Browser for iOS. In *JCDL '13: Proceedings of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries.*

[21] H. Van de Sompel, M. L. Nelson, and R. Sanderson. HTTP framework for time-based access to resource states – Memento draft-vandesompel-memento-05. https://datatracker.ietf.org/doc/ draft-vandesompel-memento/, 2012.

[22] H. Van de Sompel, M. L. Nelson, R. Sanderson, L. L. Balakireva, S. Ainsworth, and H. Shankar. Memento: Time Travel for the Web. Technical Report arXiv:0911.1112, 2009.

[23] H. Van de Sompel, R. Sanderson, M. L. Nelson, L. L. Balakireva, H. Shankar, and S. Ainsworth. An HTTP-Based Versioning Mechanism for Linked Data. In *Proceedings of the Linked Data on the Web Workshop (LDOW 2010)*, 2010. (Also available as arXiv:1003.3661).

[24] J. Wang. A survey of web caching schemes for the Internet. *SIGCOMM Comput. Commun. Rev.*, 29(5):36–46, Oct. 1999.

[25] P. Webster. Surfing the web in time: Mementos. `http://britishlibrary.typepad.co.uk/webarchive/2013/01/surfing-the-web-in-time-mementos.html`, 2013.

[26] D. Wessels. *Squid: the definitive guide.* O'Reilly Media, Incorporated, 2004.

[27] H. Zhu and T. Yang. Class-based cache management for dynamic web content. In *IEEE INFOCOM*, pages 1215–1224, 2000.