

Developing Bloom Filters for Web Archives' Holdings

Final Project Report

Project lead: Los Alamos National Laboratory (LANL) Research Library

Project partners: National and University Library in Zagreb (NSK)

Project Team Members: Martin Klein, project lead (LANL), Lyudmila Balakireva (LANL), Karolina Holub (NSK), Ingeborg Rudomino (NSK), Draženko Celjak (SRCE), Miro Milinović (SRCE)

Other Collaborating Institution: University of Zagreb University Computing Centre (SRCE)

Funding: 24,741 USD

Project website: <https://netpreserve.org/projects/bloom-filters/>

1. Background

The main goal of the project was to develop a framework for web archives to create Bloom filters based on their holdings of archived web resources. A Bloom filter (BF) is a data structure that, in our scenario, contains hash values of all (or a subset of) URLs, of which an archive has one or more archival copies. Two main use cases fall in scope for this project and are supported by a BF implementation:

- 1) Sharing of an archive's holdings (URLs) and
- 2) Querying the holdings of one or more archives.

Since URL strings are hashed before ingested into the BF, the index of an archive is not shared in plain text when the BF is shared with trusted parties. On the other hand, a BF implementation does allow queries for a URL to confirm if an archive indeed has one or more archival copies of that URL.

This collaborative project between Los Alamos National Laboratory (LANL) and the Croatian Web Archive (HAW), from the National and University Library in Zagreb (NSK), developed by NSK and University of Zagreb University Computing Center SRCE), aimed at developing software to create BFs, evaluate the scalability of the approach, pilot a search service based on BFs, and design a framework for archives to share their filters with trusted parties. In the remainder of this document, we will report on the work completed with respect to the individual deliverables, outline aspects of future work, and conclude with our recommendations for the use of BFs for the web archiving community.

2. Project deliverables

2.1. Software package that web archives can run on their end to create Bloom Filters

Our work for this part of the project began with a thorough investigation of several alternative software packages that support the implementation of BFs. After comprehensive testing and careful consideration, we decided in favor of the [Orestes Bloom filter library](#), which comes with various characteristics beneficial to our efforts. For example, it automatically determines the optimal number of hash functions to be deployed, which significantly streamlines its application. The library offers two "modes" of BF implementations:

- 1) a Redis database based and
- 2) a system memory based mode.

The Redis-based mode offers persistent storage and can potentially be distributed across systems for more efficient use. The memory-based mode does not provide persistent storage and is dependent on the corresponding process running. If the system is rebooted, for example, the memory-based BF needs to be recreated.

Before the BF is deployed, the library further requires two input parameters:

- 1) the maximum number of URLs that will be ingested into the BF and
- 2) the desired false positive rate (FPR).

We based our work on CDX files we received from HAW and initially created BFs for a CDX file created from the 2020 Croatian domain crawl. The CDX file contains approximately 180 million URLs, so we set the first input parameter cautiously to 200 million and decided to aim for a FPR of 1%. For this scenario, the library selected 5 hash functions to be applied for optimal operation of the BF. We have developed code to create such BFs for the Redis-based and the memory-based mode and shared it among project partners. Both LANL and HAW have run the code successfully on the same CDX files and we can offer a comparison of measured values shown in Table 1. Some of the numbers are identical, for example, the number of ingested entries and the size of the BFs, which serves as a confirmation of the quality of the code. Others are also identical such as the FPR, which is reassuring regarding the performance of the BFs regardless of the mode and the site of implementation (LANL or HAW).

We expected some discrepancies regarding the time to ingest all entries as it depends on the performance of the system where the code is run. LANL ran the code on an AWS virtual EC2 instance with a AWS Linux operating system, 16 CPUs, and 64 GB of memory. HAW, on the other hand, ran the code on a virtual VMware instance with a Debian Linux operating system, 8 CPUs, and 48GB of memory. It is important to note that we observed a significant difference in BF creation time between the two modes. Both at LANL and HAW the memory mode is much faster to complete the ingest. For the LANL comparison, the memory mode is taking less than 20% of the time the Redis mode takes. In the case of HAW, the memory mode is taking roughly 14% of the time the Redis mode takes.

Mode	Redis DB		Memory	
	LANL	HAW	LANL	HAW
Ingested entries	180,379,433	180,379,433	180,379,433	180,379,433
Time to ingest	7,273.6s (121.2min)	10,759.2s (179.3min)	1,382.7s (23m)	1505.5s (25.1min)
Size of the BF	246MB	246MB	246MB	246MB
Query response time	0.033ms	0.051ms	0.0008ms	0.0008ms
FPR	0.66%	0.66%	0.66%	0.66%
Time to serialize to CSV	16.8s	17.8s	2.6s	3.4s

Table 1. Comparison of Redis and memory mode BF for one CDX file

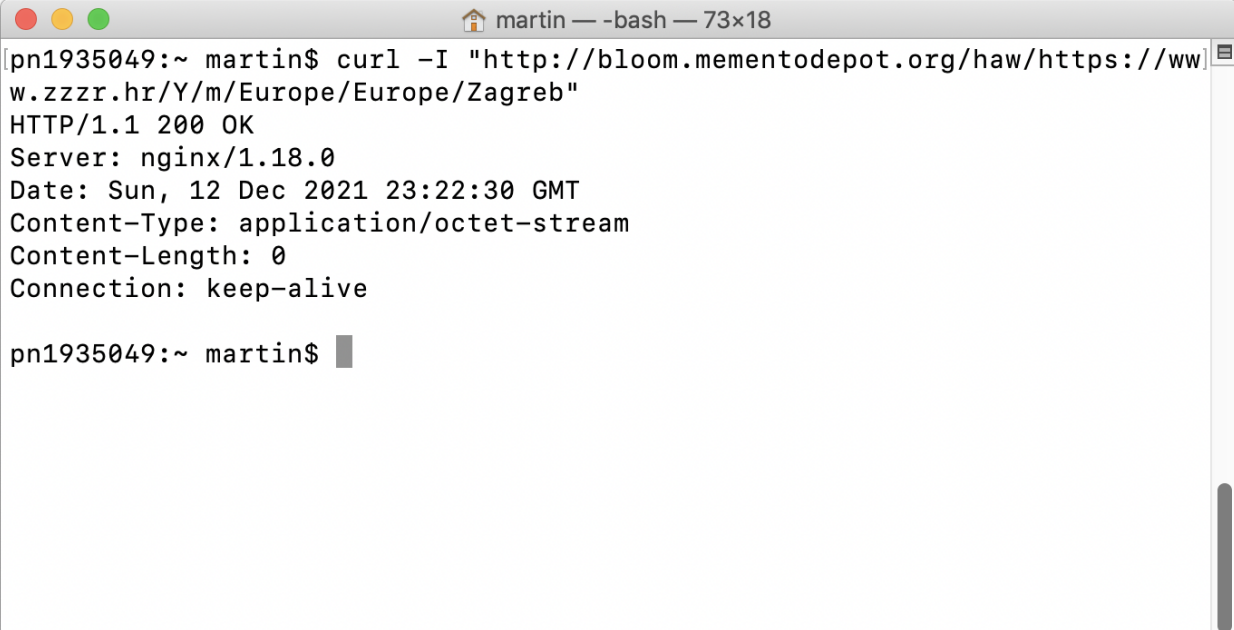
The code is currently undergoing a LANL-internal review process - a requirement for public release. We anticipate being able to release the code early 2022.

To demonstrate the use case of querying an archive's holdings, we implemented a simple pilot system that is based on BFs from HAW CDX files. The HTTP endpoint requires the URL of interest as a parameter and simply responds with a HTTP 200 ("OK") response code if the URL exists in the BF or a HTTP 404 ("File not found") if it does not. The pilot endpoint is available at:

```
http://bloom.mementodepot.org/haw/<URL>
```

where <URL> needs to be replaced with the URL of interest. Figure 1 shows a screenshot of an HTTP request for the URL `https://www.zzzr.hr/Y/m/Europe/Europe/Zagreb` against the pilot endpoint and its corresponding HTTP 200 response as the URL is indeed contained of the BF (and indexed in the CDX). Figure 2 shows a screenshot of an HTTP request for the fictitious URL

```
https://www.zzzr.hr/Y/m/Europe/Europe/Berlin
```

 against the same endpoint and its corresponding HTTP 404 response. As expected, the URL does not exist in the BF.A terminal window titled 'martin — -bash — 73x18' showing a successful curl command. The user 'pn1935049' is in the directory '~ martin'. The command is 'curl -I "http://bloom.mementodepot.org/haw/https://www.zzzr.hr/Y/m/Europe/Europe/Zagreb"'. The output is: 'HTTP/1.1 200 OK', 'Server: nginx/1.18.0', 'Date: Sun, 12 Dec 2021 23:22:30 GMT', 'Content-Type: application/octet-stream', 'Content-Length: 0', and 'Connection: keep-alive'. The prompt returns to 'pn1935049:~ martin\$' with a cursor.

```
pn1935049:~ martin$ curl -I "http://bloom.mementodepot.org/haw/https://www.zzzr.hr/Y/m/Europe/Europe/Zagreb"
HTTP/1.1 200 OK
Server: nginx/1.18.0
Date: Sun, 12 Dec 2021 23:22:30 GMT
Content-Type: application/octet-stream
Content-Length: 0
Connection: keep-alive

pn1935049:~ martin$
```

Figure 1. Query for URL of interest, contained in BF

A terminal window titled "martin" with a width of 73x18. The prompt is "pn1935049:~ martin\$". The user enters the command "curl -I \"http://bloom.mementodepot.org/haw/https://www.zzzr.hr/Y/m/Europe/Europe/Berlin\"". The output is: "HTTP/1.1 404 Not Found", "Server: nginx/1.18.0", "Date: Sun, 12 Dec 2021 23:22:47 GMT", "Content-Type: application/octet-stream", "Content-Length: 0", and "Connection: keep-alive". The prompt returns to "pn1935049:~ martin\$".

```
pn1935049:~ martin$ curl -I "http://bloom.mementodepot.org/haw/https://www.zzzr.hr/Y/m/Europe/Europe/Berlin"
HTTP/1.1 404 Not Found
Server: nginx/1.18.0
Date: Sun, 12 Dec 2021 23:22:47 GMT
Content-Type: application/octet-stream
Content-Length: 0
Connection: keep-alive

pn1935049:~ martin$
```

Figure 2. Query for URL of interest, not contained in BF

2.2. Investigation of scalability and optimization parameters for the creation of Bloom Filters

In the process of evaluating the software library that we ended up using, we conducted a number of tests to better understand its utility and options to optimize the BF creation. Specifically, we tested its ability to automatically choose the optimal number and types of hash functions. Figure 3 shows the results of our analysis of the number of applied hash functions (displayed on the x-axis). The red line, which corresponds to the right y-axis, indicates the FPR and we can observe that indeed utilizing 5 hash functions results in the best FPR. The green lines represent the measured response time when querying the BF for specific URLs. We can observe a slight linear increase in response time with an increasing number of hash functions used.

Bloomfilter: 200 mln capacity;1247044840 bits (m/n ~6.23) and 0.05 target probability

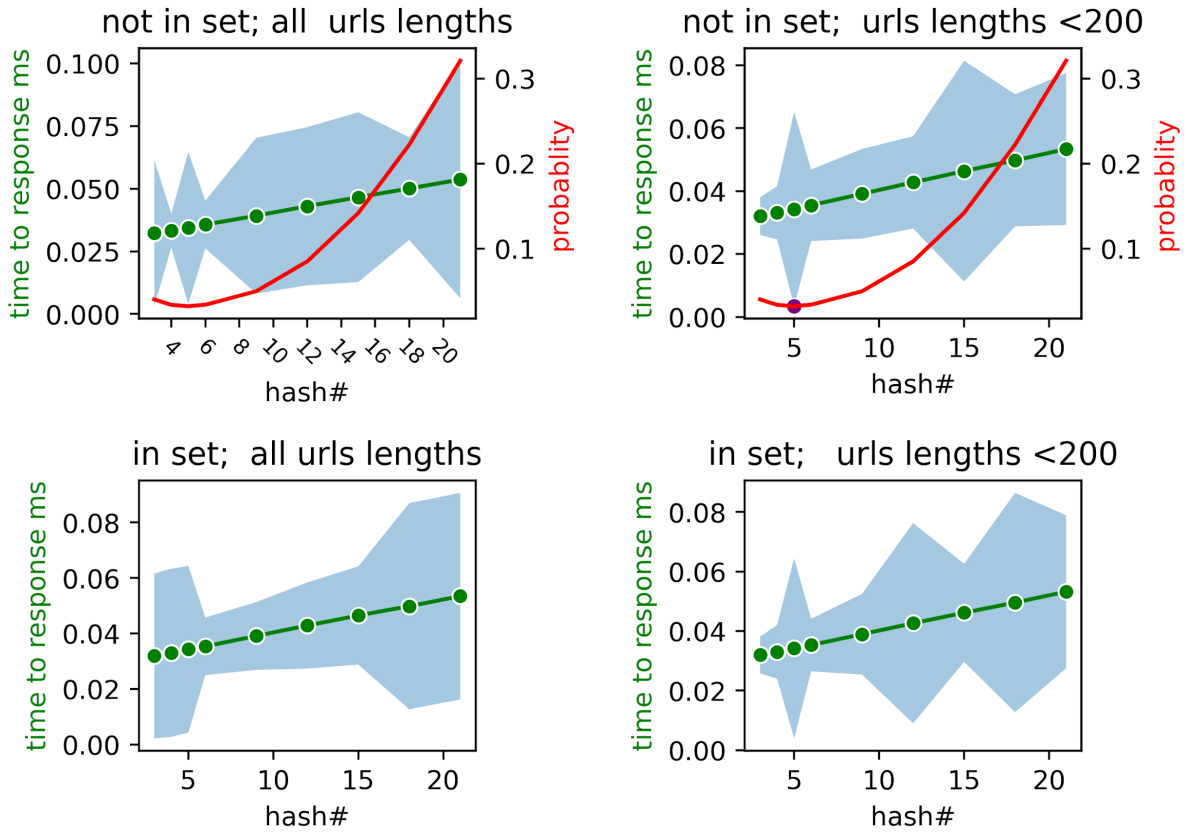


Figure 3. Comparison of number of hash functions, FPR, and retrieval time

Figure 4 highlights the outcome of our investigation into the types of hash functions used (x-axis). The red line, again representing the FPR, shows that some functions are much better suited for our task than others. Fortunately, the BF library we chose does the selection for us, so we can expect an optimal performance of the BF. The green lines again represent the measured response time, which is seemingly unaffected by the type of filter used.

Bloomfilter: 200 mln capacity;1247044840 bits (m/n ~6.23) and 0.05 target probability,k=5

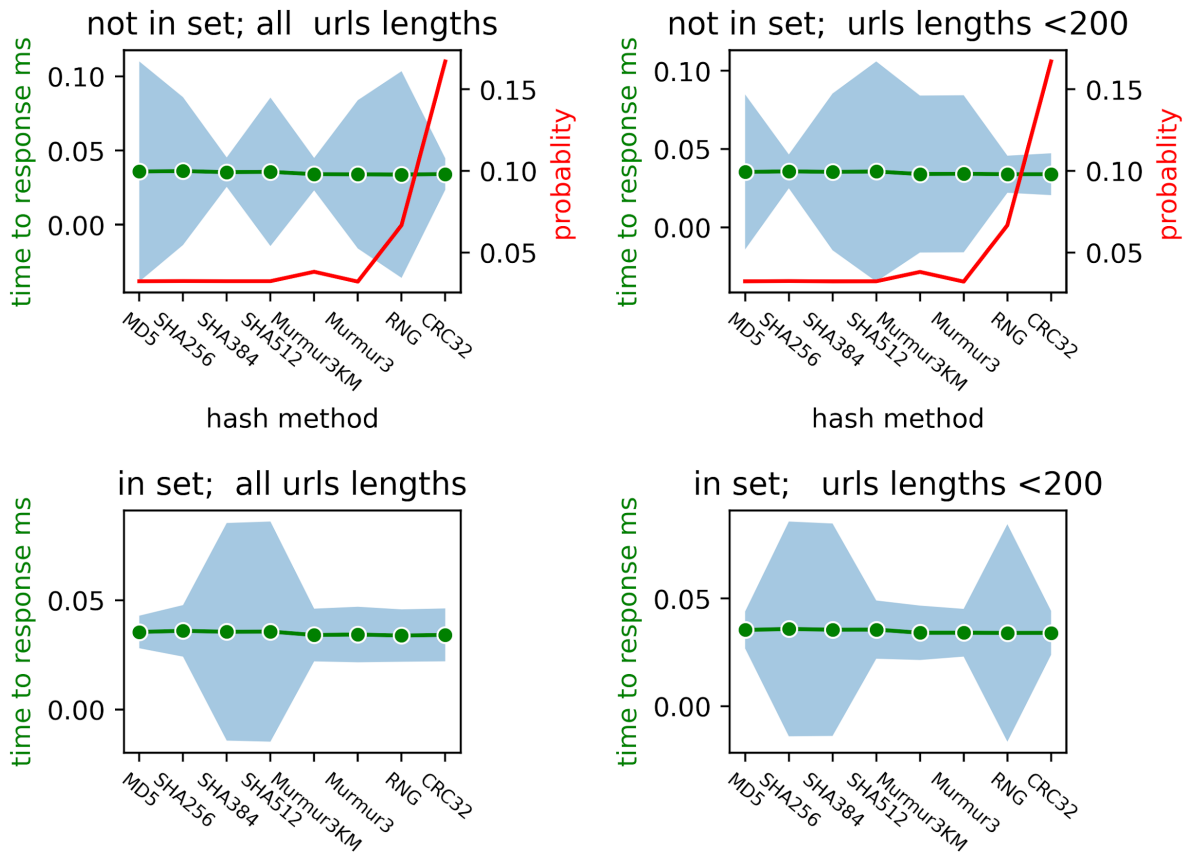


Figure 4. Comparison of types of hash functions, FPR, and retrieval time

As stated, we initially worked with one CDX file from HAW but we were also motivated to investigate how a BF solution could accommodate a scale such as the entire HAW index and take HAW's anticipated annual growth into account.

Currently, the entire HAW collection holds approximately 800 million URLs, with an anticipated growth of 200 million URLs per year. For the Redis mode, the BF library stores a set of bits for the BF in one key of the Redis database. Unfortunately, Redis has a 512MB limit for a key, meaning we can not exceed 512 MB for one filter. In addition, the BF library we utilize uses integer values, which comes with the even tighter restriction that a BF, regardless of whether it is implemented in Redis or memory mode, can not grow beyond 268.44MB.

For our scenario, the entire HAW collection at its current size would need roughly 1GB and therefore exceed the size of the BF with our library used as well as the Redis key limit. To address this scale and the anticipated growth, we considered a solution [previously proposed](#), named Dynamic Bloom Filters (DBF). DBFs consist of several homogeneous BFs with one active BF for insertion. However, when querying a URL, for example, all of the homogeneous BFs need to be checked one by one.

This leads to the major downside of DBFs - the array of BFs will have a higher total FPR, proportional to the FPR of the individual filters. In fact, with an 1% of FPR of individual BFs the FPR of the array of 20 BFs is estimated at 18%. An overview of estimated combined FPRs is shown in Figure 5.

Estimated fpr of Array of identical Bloomfilters

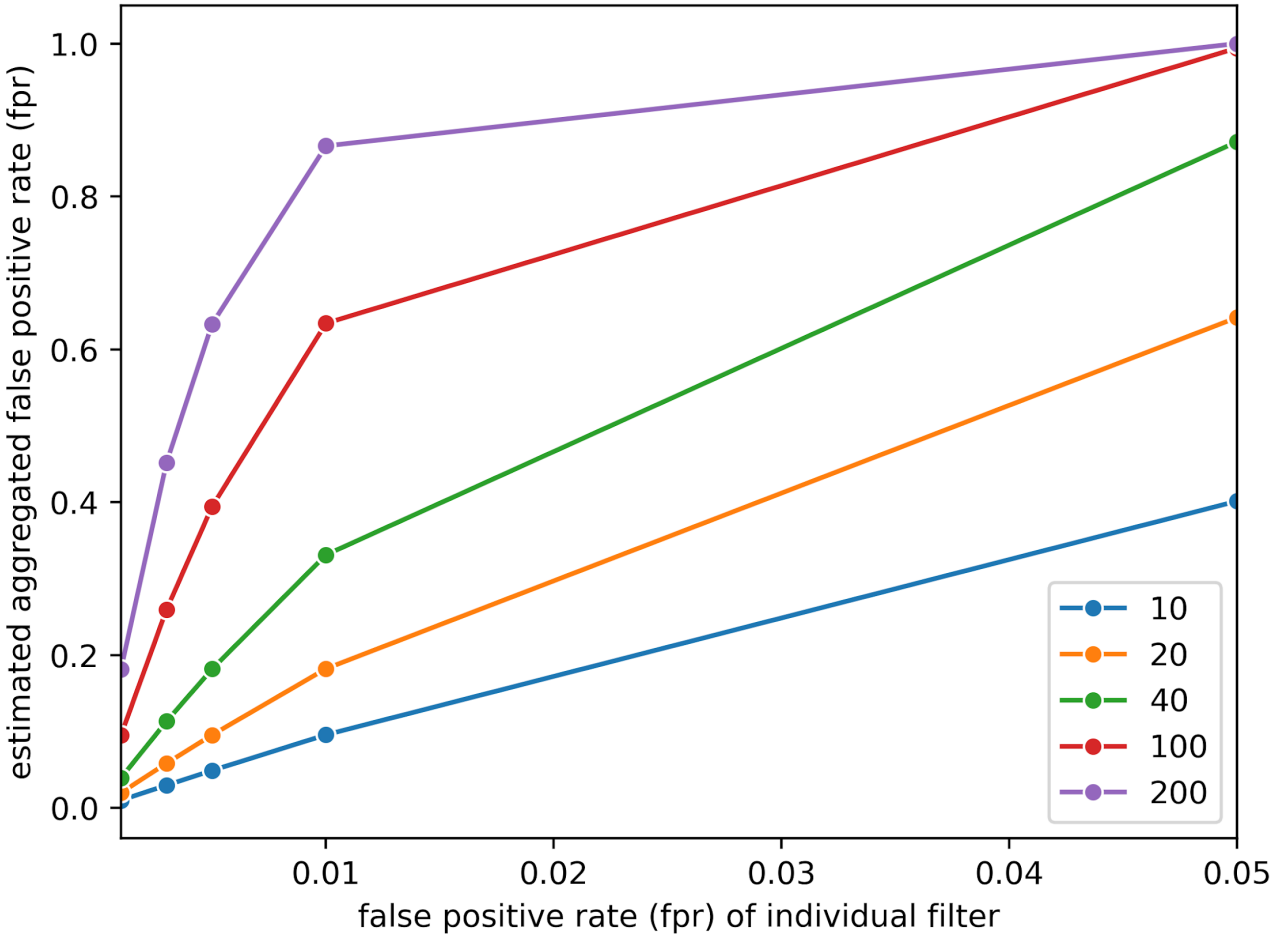


Figure 5. Estimated aggregate FPR for DBFs

If we chose this DBF solution, we would have to decrease our chosen FPR for the individual filter significantly in order to keep the performance of this array reasonable. With lowering individual FPRs, the size of each single filter increases as well, as shown in Figure 6.

Bloomfilter: 200 mln capacity, Murmur3, k=5

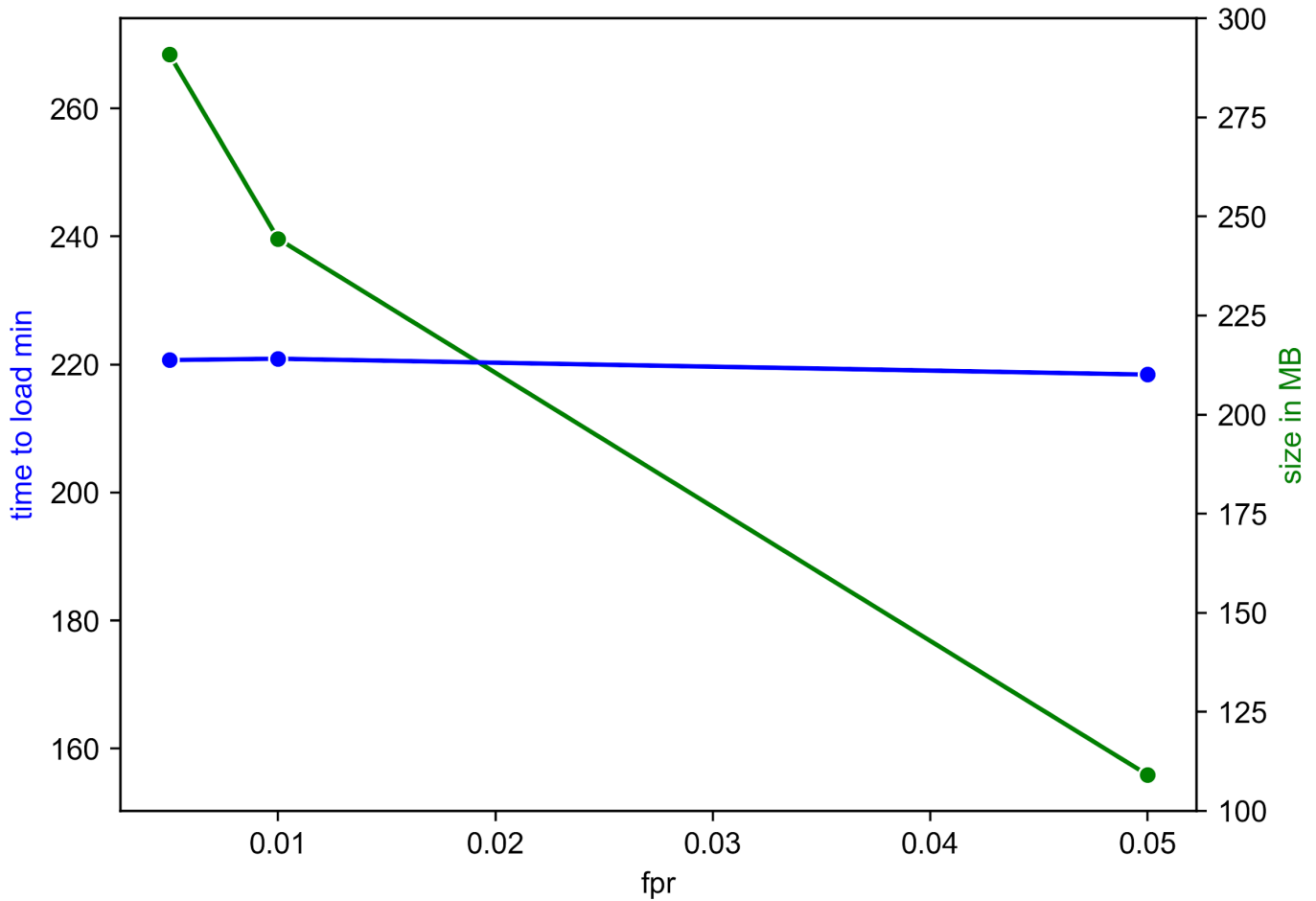


Figure 6. BF size in relation to FPR

At the moment the HAW collection would fit into four filters of 200 million URL capacity for each. Given the estimated growth, we can expect to need one new filter per year. As a solution to accommodate for these numbers, we propose a modified version of the DBFs. Specifically, our approach entails creating an array of 16 homogenous BFs upfront (now) and naming each filter using all possible hexadecimal characters. So the 16 filters would be named: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', and 'f'. We use a special routing rule for insertion that is based on the first hex character of the URL's MD5 hash value. So each URL will be inserted in its corresponding filter, matched by name. Figure 7 depicts this model by means of the example scenario to ingest the URL `https://haw.nsk.hr/en` into the BF named '3', given that the URL's md5 hash value starts with that character.

Since the distribution is done by hash values, the filters will have an equal population of URLs and hence the same probability for insertion. The URL lookup will also be routed for appropriate BF based on the URL first hex character, avoiding parallel querying, which was the main bottleneck in previously proposed DBFs. The initial memory requirement for the 16 filters with a desired FPR of 1% is 3.8 GB and with a FPR of 5% it comes to 1.6 GB. Also, with a high re-crawling rate of the same URL the filters can stay even longer below the calculated capacity.

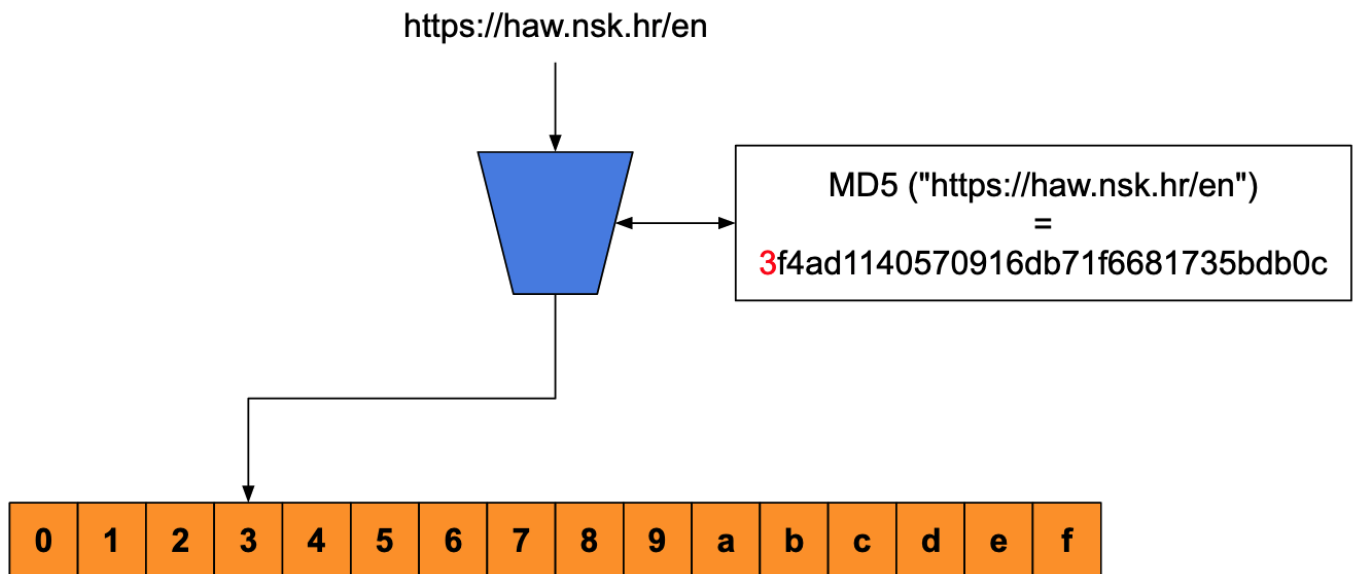


Figure 7. Schematic depiction of our modified DBFs approach

In this scenario, HAW could have a FPR of 1-5% or less for the next 10 years without any changes to the infrastructure. After 10 years for each BF leaf, a new filter can be added with a total of 32 filters. For each URL, two lookups would now need to be done. Alternatively, the BF array could be rebuilt with, for example, URL distribution to 32 filters.

We have created code to implement this modified DBF concept and shared it among our project partners. The code is currently undergoing LANL-internal review and once approved, we will release it as open source. The pilot endpoint described in Section 2.1 is, in fact, an example for such an implementation. It serves results from 16 BFs in accordance with the modified DBF approach and encompasses the entire HAW index - currently more than 800 million URLs.

2.3 Pilot federated search system for archived web resources based on Bloom Filters

Similar to the pilot outlined in section 2.1, we have implemented a system that is based on BFs created from multiple CDX files obtained from various different web archives. Specifically, next to the CDX files from HAW, we created separate BFs from CDXs from the Internet Archive and the British Library. The federated search system, upon receiving a lookup request for a URL of interest, queries each BF and offers the results in the response. The service returns a HTTP 200 response code and lists the web archives and their corresponding status regarding the requested URL in the response body.

The pilot federated search endpoint is available at:

`http://bloom.mementodepot.org/bloom/agg/<URL>`

Where `<URL>` needs to be replaced with the URL of interest. Figure 8 shows a screenshot of the response from the request for the URL: `https://wordpress.com/robots.txt`, which is part of all three BFs and Figure 9 shows the response for the request for the URL:

`http://www.econ.yale.edu/smith/econ116a/keynes1.pdf` for which only the Internet Archive and the British Library have archival copies but HAW does not.

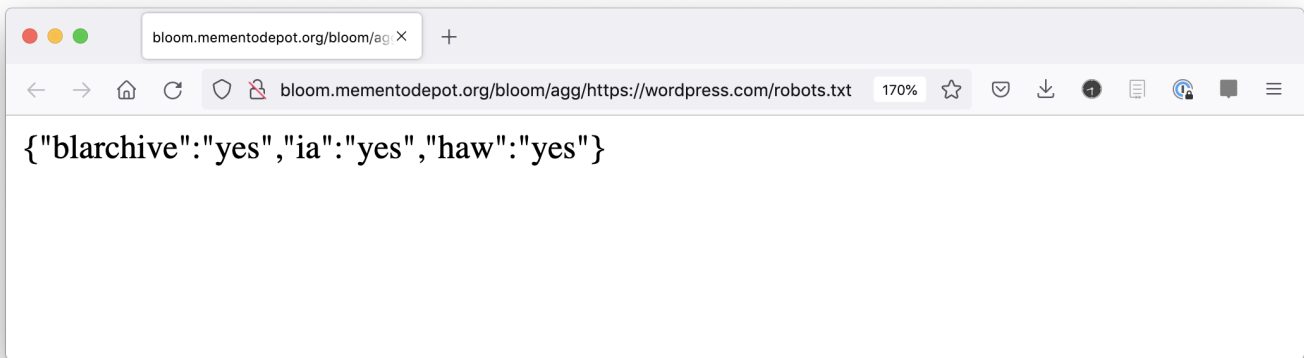


Figure 8. Query for `https://wordpress.com/robots.txt` against federated search, contained in all three BFs

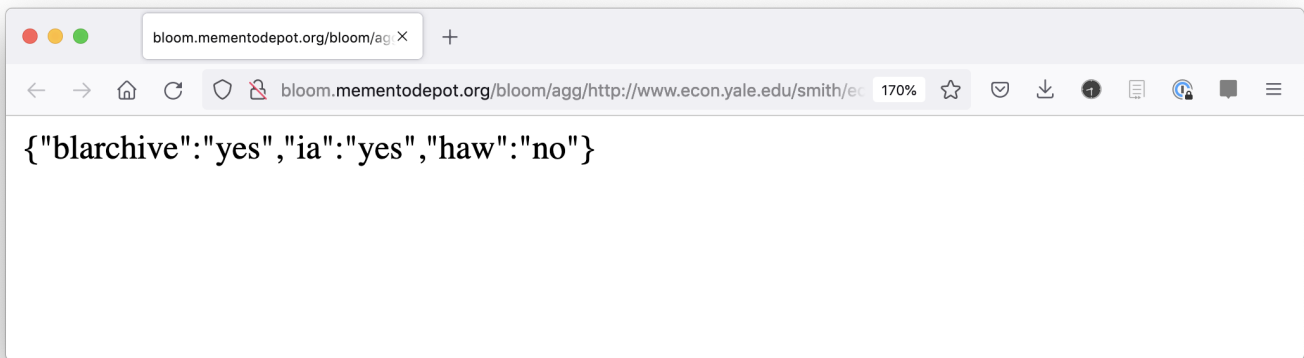


Figure 9. Query for `http://www.econ.yale.edu/smith/econ116a/keynes1.pdf` against federated search, contained in two of three BFs

2.4 Feasibility analysis of incremental Bloom Filters

According to our experiments, updating an existing BF is possible, regardless of whether the Redis or the memory mode is deployed. Naturally, the procedure of adding entries to the BF is different for both modes and specific optimizations may be required to have the changes immediately be visible. However, the main constraint is the predefined maximum of URLs. For one, this upper bound can not be edited once the BF is created and for two, it can not be exceeded by the operation to add entries into the BF.

Depending on the use case for which a BF is deployed, this constraint may prohibit the adoption of a BF-based system.

2.5 Conceptual design of a framework for web archives to share Bloom Filters

From the perspective of HAW's production environment it is not preferred to run services for which there is no in-house expertise that would ensure the support over a longer period of time. The concern with the BF implementation using a Redis database was the lack of expertise. Also, since HAW already supports some kind of lookups through the Wayback interface, it is more important to openly share HAW's BFs with other services than to establish another lookup service. Based on the above, the Redis based solution was not

necessary since the goal of creating and sharing BFs could be achieved using the memory based solution described in 2.1.

We used a memory based solution to create BFs for targeted CDX files and to serialize the BFs to a plain text file (text/csv). The last row of Table 1 shows the time it takes for Redis and memory based BF implementations to serialize the data structure into a plain text file. Not surprisingly, the memory mode is much faster but the Redis mode is not prohibitively slow.

We used one folder to store all serialized BF files. Subfolders could also be used if needed to better organize BF files. The folder is shared on the web using the same web server that is used for serving the archives's other content (Apache HTTP Server). Any usual web server could be used for that purpose.

As a simple solution to advertise and expose BF files we suggest the Sitemap protocol, more precisely the sitemap.xml file. Since the protocol [allows to put the sitemap file in subdirectories](#), the sitemap.xml file could be hosted in the folder where the BF files are as well and be edited separately from other parts of the web content. The Sitemaps protocol also allows for extensions of the XML to be included with any additional namespace by specifying the namespace in the root XML element. We used the Dublin Core schema to include additional metadata that describe the exposed BF files. The HAW's sitemap describing its BF is depicted in Figure 10.

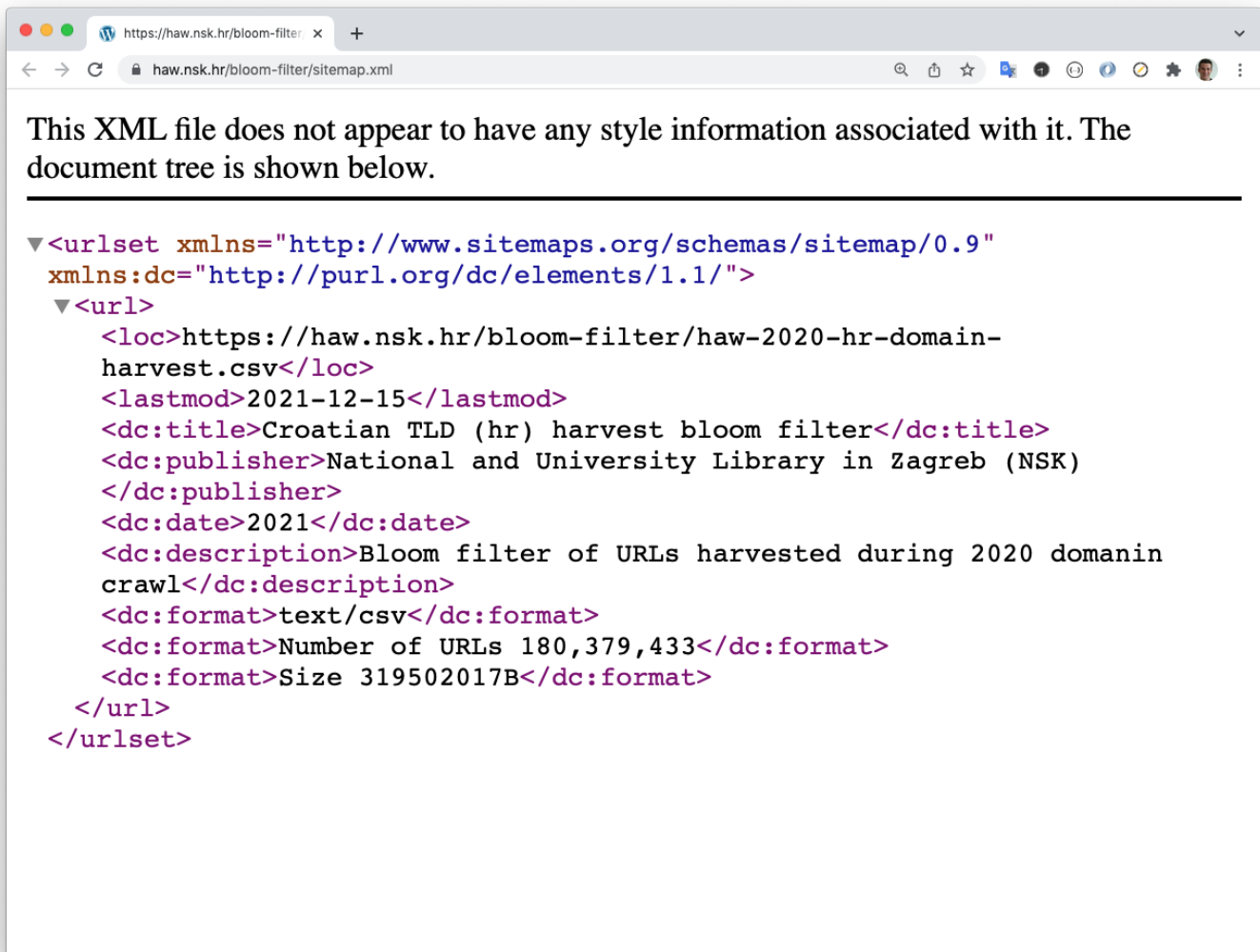


Figure 10. HAW sitemap with added metadata elements, available at: <https://haw.nsk.hr/bloom-filter/sitemap.xml>

The process of sharing an archive's BF has the following steps:

- 1) BF creation in memory
- 2) Serialization of BF into plain text file
- 3) Creation/update of sitemap, possibly with added metadata.

As with all sitemaps, it remains the responsibility of the services interested in consuming content (in our case aggregators, for example) to discover the data and obtain a copy. However, value-added services such as registries that provide an overview of hosts and locations are conceivable. [Well-known URIs](#) may be another good option to standardize the discovery of BF sitemaps.

Once a service has discovered the sitemap, the workflow is simple: parse the sitemap XML file, extract the URL of new or updated BF files, get the serialized plain text file, and ingest the file into the local BF implementation.

3. Future work

While this project has proven very informative and we are incredibly pleased with the results, we do see a few aspects of future work. For example, we would like to conduct more experiments with CDX files of various sizes, created with different software tools, and from different domains and time frames. Testing our BF creation software on diverse CDX files will almost certainly robustify the code base.

As we have seen in the HAW environment, a Redis-based implementation may be a blocking requirement for some organizations. Enhancing the code to support other similarly well-performing databases is certainly of interest. Further code improvements entail a proper parallelization of the BF creation process to better utilize high-performance computing environments.

In addition, related [web archiving profiling work](#) by Sawood Alam provides promising ground for a comparative analysis. We have engaged in preliminary discussions with Sawood and bilaterally expressed great interest in this endeavor.

During the final webinar, an idea related to further scalability testing was brought up. It entails truncating the URLs to be ingested into BFs in order to reduce the overall amount of URLs. Obviously, that would raise the FPR but for scenarios where that is tolerable, it might be a good compromise to make.

4. Conclusion

When we proposed this project, we were aware of previously highlighted [shortcomings of BFs](#) but also of their application in areas very relevant to the IIPC such as for [duplicate detection for a web crawler](#). We are encouraged by the findings of our investigation and are very pleased with the progress we made in developing a robust piece of software to create BFs both in the Redis and memory mode. We have showcased that BFs can, in fact, be used for a local or federated lookup service of URLs from web archives. BFs can also be serialized into plain text files that are well-suited for sharing with trusted partners. We have further proposed a modified DBF solution that, to an extent, addresses scalability of BFs for our use cases.

However, the application of BFs for very large CDX indexes, such as those of the Internet Archive, the Library of Congress, or the British Library is likely not a reasonable endeavor. We consider BFs much better suited for smaller archives and for individual collections as part of archives' holdings. Collections could, for example, be of a specific topic, in a particular language, from an individual domain, or from a defined timeframe. As such, BFs could be used for live lookups into existing and continuously growing collections to avoid crawling duplicate URLs, for example when building a topic collections such as the end of term collection. Such a setup could be particularly interesting in a distributed crawling environment when index information can be shared in near real-time.