1 Introduction

Web archiving initiatives generate vast amounts of data. The Internet Archive advertise their collection to be almost 2 petabytes, growing at a rate of 20 terabytes per month[1]. As of October 2013 the British Library's archive of the UK web totalled 21 terabytes, growing by 4.5 terabytes over a one month period[2]. The cost of providing storage for large collections can be high. For instance Amazon's Glacier service, an "extremely low-cost storage service", advertise storage rates of \$0.01 per GB per month as of February 2014. At this rate The Internet Archive would pay \$20,972 per month, the British Library would pay \$215. Requests to browse the archive would incur additional costs, as would expanding the archive. This situation motivates us to ask how web archive data can be compressed in order to optimally reduce storage space.

The Web ARChive (WARC) file format is the ISO standard[3]commonly used to store web archive data. It is a plain text format that contains records of requests and responses of URLs, along with associated metadata, such as a list of links contained within the response data. The recommendation in the WARC standard is to append records to WARC files until they reach a size limit, at which point they should be gzipped and stored. The recommendation is that uncompressed WARC files should be no larger than one gigabyte. Using this recommendation our data set from Section 4 compresses down to 15.60484% of its original size. The WARC file format is extensible and the standard lists possible compression extensions. To our knowledge no such extension has been made publicly available and none are widely used. In this paper we explore possible extensions to the WARC format that would allow delta compression of consecutive records as well as different compression algorithms. We aim to: (i) reduce the total archive size and, (ii) allow easy partitioning of the database. The strategy that leads to the smallest total archive size compresses down to 8.87926% of the original. Applying our strategy to the Internet Archive's collection would reduce their (hypothetical) Amazon Glacier costs to $\$11,933$ per month, a saving of $\$9,039$.

[3]https://archive.org/about/faqs.php#9

[3]https://web.archive.org/web/20131017144821/http://www.webarchive.org.uk/ukwa/statistics

[3]http://www.iso.org/iso/catalogue_detail.htm?csnumber=44717

2 Background

1. WARC spec
2. gzip, bzip2, tar. What they do, why these ones?
3. vcdiff, bsdiff, diffe. What they do, why these ones?
4. internet archive, IIPC, BL. Collections, experience
5. Heritrix

3 Experiment: Generated Data

1. generate 1MB text data
2. apply change repeatedly
3. compression strategy
4. compare
5. very many changes, over time. What wins
6. More than just text data? What kinds of changes?

4 Experiment: GitHub Pages

The code hosting service GitHub[4]offers a free service called GitHub Pages[5]that allows users to host static web content for free. In order to evaluate different compression strategies on realistic data we conducted a crawl of GitHub Pages projects. These projects are stored in version control and so it is possible to iterate through the changes made to the files over time. This approximates the changes observed on crawled web pages over time and so forms a suitable data set for compression comparison. There were 27,507 suitable projects as of November 2013. Of these projects, X did not contain suitable web documents.

4.1 Data

The data set is made up of projects hosted on GitHub that conform to the projectname.github.io naming scheme. GitHub treats these projects as GitHub Pages projects and serves their content at the URL given by the name. Before hosting, a project's files are processed by a static website generator called Jekyll[6]. This software generates files from templates that use various markup languages, e.g. HTML, Markdown, or Liquid. When changes are pushed to the Git repository for the project the files and website are updated. To generate our data set we clone a project repository and iterate through every commit. For each commit we process the files using Jekyll and serve the results up locally. We then direct Heritrix to archive the site. When Heritrix has finished we stop serving and continue to the next commit.

During this process we have crawled 76 domains, discovering 5,406 URIs. Heritrix sent 96,622 requests for pages and received 35,233 unique responses. This means that for every three requests that Heritrix sent out one returned content that had not been seen before. Our archived data spans 1,142 days, starting 2010-09-13 running up to 2013-10-29.

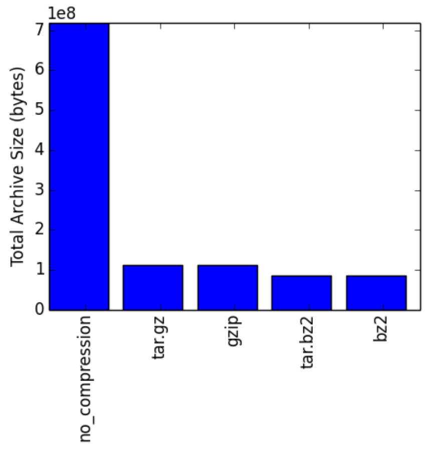
4.2 Compression Analysis

We now explore different strategies for minimising the archive size using compression algorithms. We begin with the WARC files that Heritrix has produced. Before compressing we first ensure that any response records that contain identical content are replaced with revisit records with the identical payload profile. We also combine files under the same domain, limiting the total file size to 1 gigabyte, as recommended by the specification. We consider three different delta algorithms; diff, bsdiff, and vcdiff. For each algorithm we use three different strategies; delta from the first ever record, from the immediately preceding record, or from a reference frame stored every 10 records. When we apply a delta to a record we leave the WARC headers intact and we only consider response-type records. We compare two compression algorithms; gzip and bzip2, we also consider tar files that use these compression types. When compressing, we treat each file individually, when using tar compression we consider all files under the same domain. Storage restraints meant that our experimental set up did not allow us to

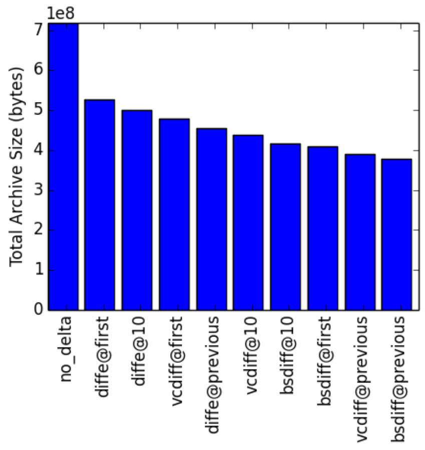
[6]`http://github.com/`

[6]`http://pages.github.com/`

[6]`http://jekyllrb.com/`



(a) Compression comparison.

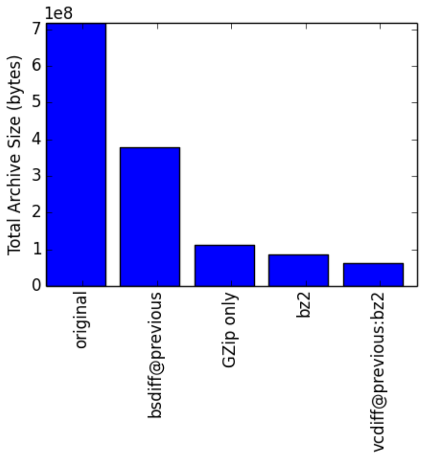


(b) Delta comparison.

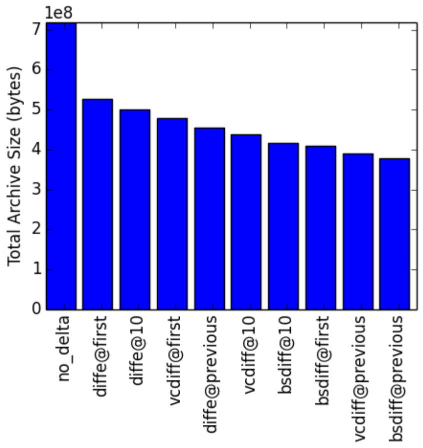
Figure 1: Total archive size for different delta and compression strategies.

consider the archive as a whole, we instead consider each domain individually. This means that each WARC file we consider contains only records from that domain. It also means that many WARC files do not reach the 1 gigabyte size limit. When creating a tar file we can only combine files from the same domain. We believe that this means that our compressed archive sizes are slightly larger than they would be in a production environment.

Figure 2 shows the total archive size for different compression and delta algorithm strategies. The WARC specification recommends that files are compressed using gzip. Applying this strategy to our archive compresses the files down to 15.60484% of the original. We see that applying a bzip2 algorithm would compress the archive down to 11.85044%. Figure 2b shows the effect of applying delta algorithms to the data. The best performing delta strategy uses bsdiff to calculate the delta between each consecutive record. Using this strategy we see a 52.69142% reduction in total archive size. Figure 2a compares the total archive sizes when using compression only, delta only, and a combination of the two. By applying a delta algorithm to the response records and then compressing the resulting files we can further reduce the total archive size. We see that if we apply the vcdiff algorithm to each consecutive record and then compress everything using bzip2 we can reduce the total archive size to 8.87926% of the original. This best performing strategy produces a collection 56.90070% of the size of that of the recommended compression strategy.



(a) Total archive size for different delta and compression strategies.



(b) Delta comparison.

Figure 2: Total archive size for different delta and compression strategies.

1. what are the pros and cons of each strategy. e.g. speed, partitioning, do we gain little over the other strategies in terms of size but lose on speed?

2. additionally:

   (a) average size of warc record types, before and after.

   (b) counts of warc record types

   (c) percentage space taken up by each type

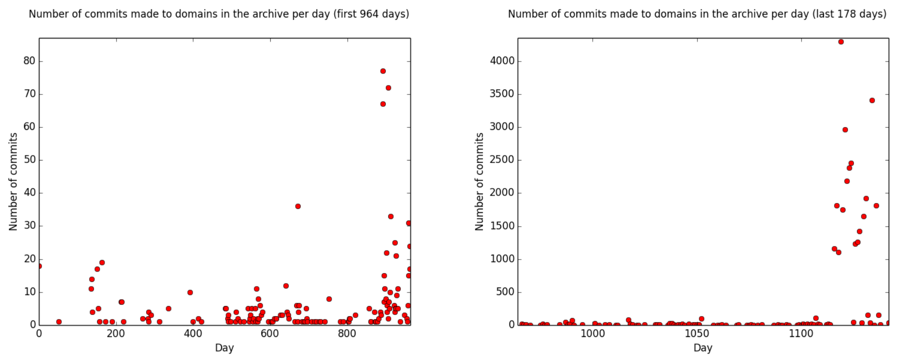
   (d) savings introduced by compression, why not compress the other types?

   (e) compression performance by content type

   (f) refer to content analysis for frequencies of content type. e.g. is it worth finding an algorithm that compresses images really well? or can we ignore them because HTML and its high change frequency trump image for storage issues.

4.3 Content Analysis

Figure 3 shows the number of changes made to all files in the archive per day. The data is split into two parts for clarity. The split is made on the day that



(a) During the first 1658 days. (b) During the last 171 days.

Figure 3: Number of commits made in the archive per day.

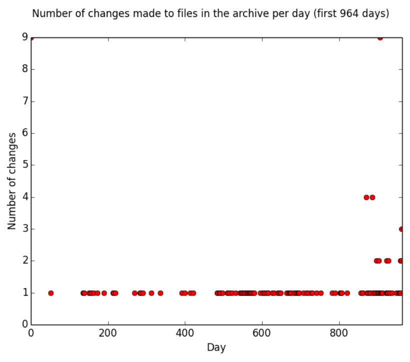
GitHub pages launched their new naming scheme, projectname.github.io. Previously the naming scheme had been projectname.github.com. Our data crawl did not consider projects that had kept the old naming scheme, this is why the numbers increase. In the first 1658 days the average number of changes per day was 0, in the last 170 that increases to 58.

Each change above is an observation of a single file's contents differing from a previous crawl. This means that if a HTML file and a CSS file were updated simultaneously, we would record two changes. This might not align with an archivist's view of what a webpage change would be. Instead, they might expect to record a single change for updates made in a short period of time, no matter how many files were edited. In addition, we iterate through every commit made to the project but changes are only made public for every push. There may be several commits made to a project before a push. Considering this, we can instead assume that a maximum of one change is made to a domain per day. Figure 4 shows the updated number of changes made to the archive per day. During the first segment we expect to have a domain updated in the archive once every 12 days, in the second segment this is once every day.

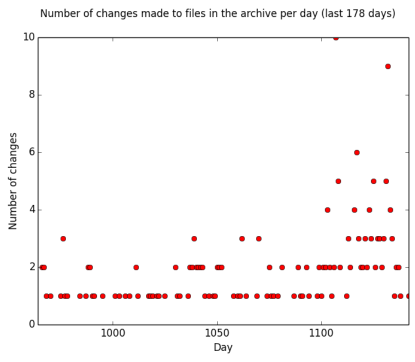
Figure 5 shows the number of URIs observed in each domain. Figure 5a shows the total number of URIs ever observed at a domain. Figure 5b shows the average of the number of URIs observed at each domain over all crawls.

5 Experiment: National Archive Collection

1. Get data from major collection

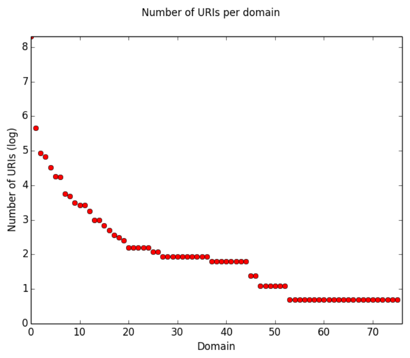


(a) During the first 1658 days.

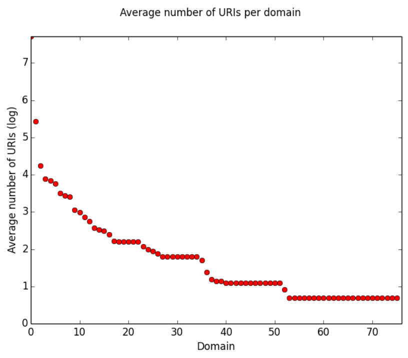


(b) During the last 171 days.

Figure 4: Number of changes made in the archive per day.



(a) Total URIs.



(b) Average URIs.

Figure 5: Number of URIs observed at each domain.

2. BL, archive.org, etc.
3. Apply best strategy from previous section
4. What real-world savings can we demonstrate?

6 Conclusion

The defaults in the WARC standard do not take advantage of the fact that many documents on the web will have many minor changes made to them over time. By using a delta algorithm as well as a compression algorithm we can reduce the total archive size by nearly half again.