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]. 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. 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[2]commonly used to store web archive data. It is a plain text format that contains records of requests and responses made to URLs. When building a web archive that spans many domains the standard recommends appending records to WARC files until they reach 1 gigabyte, uncompressed. At this point they should be gzipped and stored. Using this recommendation our data set from Section 4 compresses down to 24.22505% 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 17.57617% of the original. Applying our strategy to the Internet Archive's collection would reduce their collection down to 1.45107 petabytes and their (hypothetical) Amazon Glacier costs to $\$15,215$ per month, a saving of $\$5,757$.

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

[2] `https://archive.org/about/faqs.php#9`
[2] `http://www.iso.org/iso/catalogue_detail.htm?csnumber=44717`

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[3]offers a free service called GitHub Pages[4]that allows users to host static web content for free. In order to evaluate different compression strategies on realistic data we downloaded 393 project repositories. These projects are stored in version control and so it is possible to iterate through the changes made to the files over time. Doing this we can repeatedly crawl a website as changes are made to it.

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 (i.e. http://projectname.github.io). When changes are pushed to the Git repository for the project the files and website are updated. Before hosting, a project's files are processed by a static website generator called Jekyll[5]. 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. When we have processed an entire project we tidy up the WARC files that Heritrix has produced to make sure they conform to the guidelines in the WARC standard. In particular we replace any identical response

records coming from the same URI with a revisit record. We also combine uncompressed WARC files until they reach a maximum of 1 gigabyte in size.

During this process we have crawled 393 domains, discovering 17,824 URIs. Heritrix sent 324,921 requests for pages and received 71,597 unique responses. Our archived data spans 1,950 days, from 19 October 2008 to 20 February 2014.

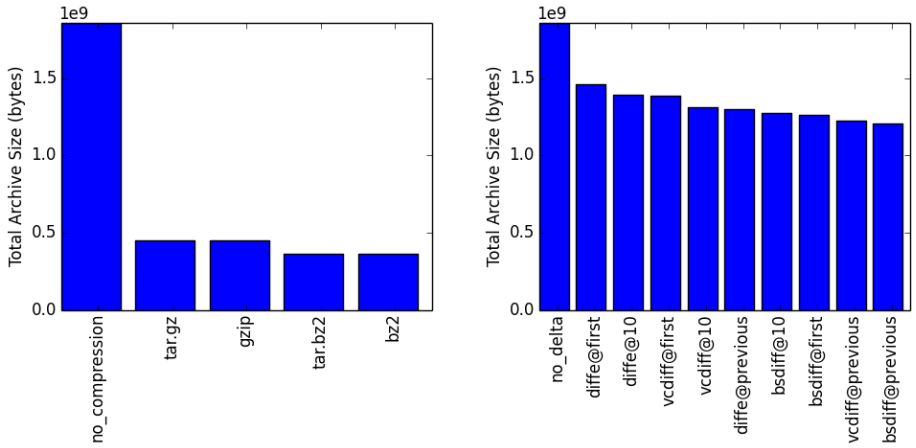
4.2 Compression Analysis

We now explore different strategies for minimising WARC file size using compression and delta algorithms. We compare two compression algorithms; gzip and bzip2, we also consider tar files that use these compression types. When compressing without tar, we treat each WARC file individually. With tar we combine all WARC files under a single domain into a tar file and then apply the compression algorithms. When applying a delta algorithm we consider every response record in a WARC file, choosing, if applicable, a second response record to compare against. We replace these response records with revisit records, the payloads of which contain the patches that can be used to re-create the original record. When choosing the response record to compare against we use three strategies. In one we compare against the first record ever recorded for that URI. In the second we compare against the record that was most recently recorded for that URI. In the third we compare against reference records that are stored every ten responses.

Storage restraints meant that we could not consider every domain at once, instead we consider each in turn. This means that each WARC file we consider contains only records from that domain. It also means that only 0 WARC files exceed 1 gigabyte size limit. We believe that this means that we are creating more WARC files than a production archive would. This makes our compressed archive sizes larger than they would be as the compression algorithms have less data to work with. The largest WARC file is 593,684,727 bytes, the smallest 4,352, the average WARC file is 4,725,962 bytes.

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 24.22505% of the original. We see that applying a bzip2 algorithm would compress the archive down to 19.77611%. Figure 2b shows

[5] `http://github.com/`
[5] `http://pages.github.com/`
[5] `http://jekyllrb.com/`



(a) Compression comparison.

(b) Delta comparison.

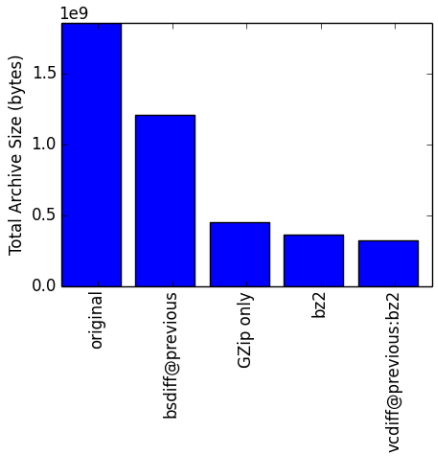
Figure 1: Total archive size for different delta and compression strategies.

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 64.99101% 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 17.57617% of the original. This best performing strategy produces a collection 72.55369% of the size of that of the recommended compression strategy.

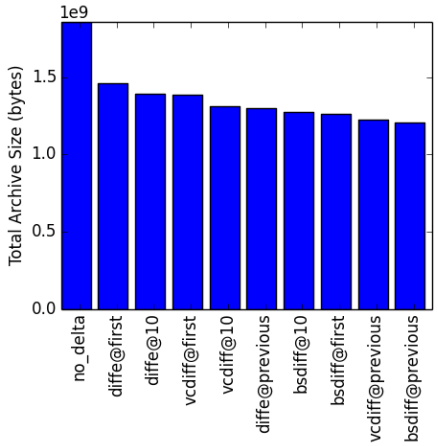
We see above that the bzip2 compression algorithm produces a smaller archive than gzip, in our direct comparison bzip2 compresses our data set down to 81.63497% of the size of gzip. In fact, bzip2 tends to more optimally compress data than gzip in many benchmarks[6]. However, those benchmarks show that bzip2 achieves its compression at a slower speed than gzip. There is a trade-off to be made between storage space and time.

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?

[6]e.g. `http://compressionratings.com/sort.cgi?txt1.brief+4np2`



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



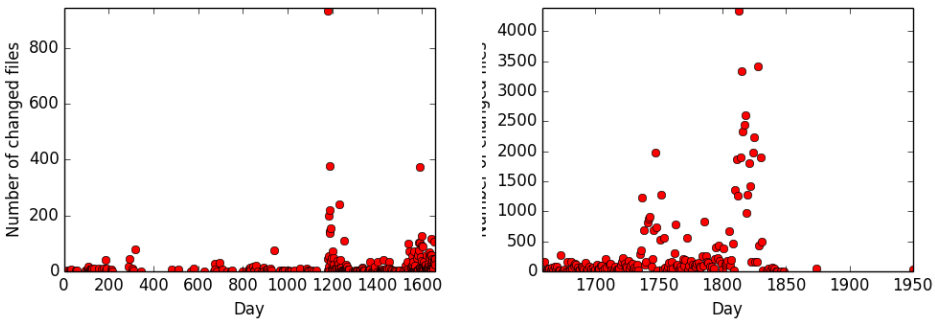
(b) Delta comparison.

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

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 files that changed in the archive per day. The data is split into two parts for clarity. The split is made on the day that GitHub pages launched their new naming scheme, projectname.github.io on 5 May 2013. Previously the naming scheme had been projectname.github.com. Our data crawl did not consider projects that had kept the old naming



(a) During the first 1,658 days.

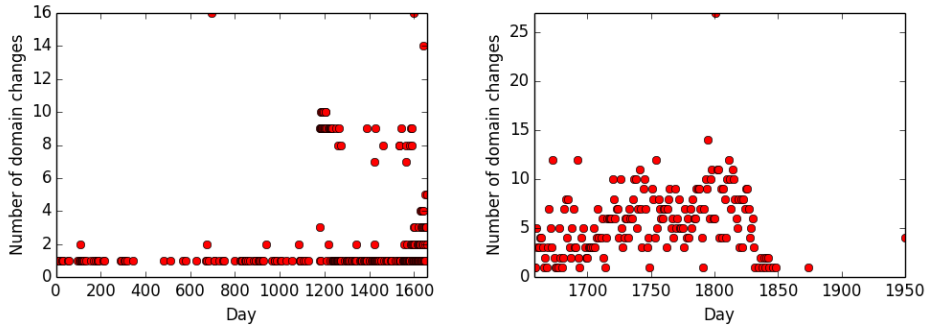
(b) During the last 292 days.

Figure 3: Number of files that have changed in the archive per day.

scheme. In the first $1,658$ days the average number of files changing per day was 4.28227, in the last 292 that increases to 220.88014.

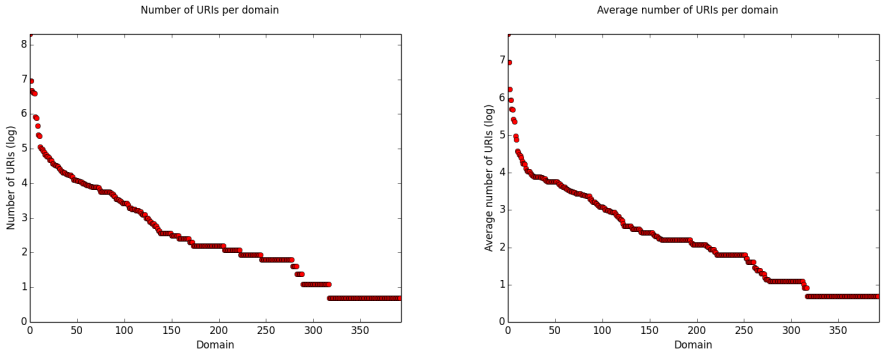
Each file change is an observation of a single file's contents differing from a previous crawl. This means that if a HTML file and a CSS file were updated simultaneously, we would record two changes. This might not align with an archivist's view of what a webpage change would be. Instead, they might 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 number of domains that change in the archive per day. During the first segment we see an average of 0.43667 domains changing per day, in the second segment this is 3.52055.

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 mean number of URIs observed at each domain over all crawls. The largest domain, beddebo-beddebo.github.io, has $4,061$ total URIs, we observed $2,234$ of those URIs, on average, when we crawled it. The smallest domain, paulhhowells-github.io, has 2 URIs, 2 on average.



(a) During the first 1,658 days. (b) During the last 292 days.

Figure 4: Number of domains that have changed in the archive per day.



(a) Total URIs. (b) Average URIs.

Figure 5: Number of URIs observed at each domain.

5 Experiment: National Archive Collection

1. Get data from major collection
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.