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 30.63897% 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 archive. The strategy that leads to the smallest total archive size compresses down to 23.29247% of the original. Applying our strategy to the Internet Archive's collection would reduce their collection down to 1.52045 petabytes and their (hypothetical) Amazon Glacier costs to \$15,943 per month, a saving of \$5,029.

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?

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

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

4 Experiment: GitHub Pages

The code hosting service GitHub[3]offers a free service called GitHub Pages[4]that hosts users' static web content for free. Each web site is stored in a Git repository, also hosted by GitHub. In order to evaluate different compression strategies on realistic data we downloaded $1,261$ GitHub Pages repositories. As these web sites are stored in version control it is possible to iterate through the changes made to the files over time. Doing this we can repeatedly crawl a web site as changes are made to it. We can then explore how different compression strategies perform. In particular we are interested in the total size of the archive after compression.

4.1 Data

The data set is made up of repositories hosted on GitHub that conform to the projectname.github.io naming scheme. GitHub treats these repositories as GitHub Pages repositories and serves their content at the URL given by the name (i.e. http://projectname.github.io). When changes are pushed to the repository the files and website are updated. Before hosting, the files are processed by a static website generator called Jekyll[5]. To generate our data set we clone a 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 repository 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 $1,261$ domains, discovering $63,179$ URIs. Heritrix sent $1,302,772$ requests for pages and received $193,912$ unique responses. Our archived data spans $1,874$ days, from 19 October 2008 to 06 December 2013.

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

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

file contains only records from a single domain. It also means that few WARC files exceed the 1 gigabyte size limit. In fact 0 WARC files exceeded the one gigabyte limit. In archiving our 1, 261 domains we created 1, 261 WARC files. In a production system these files would have been concatenated and stored in 6 files. The larger a file, the more data a compression algorithm has to work with, we believe that this may increase the size of our compressed archive over the production. However, because we store all domain related material in the same file we believe that compression algorithms have a higher quality of data to work with. This would have the effect of decreasing our compressed archive size over a production system. In a seminar in 2013 the British Library stated that their recent archiving initiatives were stored compressed down to roughly 40% of the original size. Our gzip compression reduces our archive down by 30.63897%, we believe that this indicates that our results are similar enough to a production archive to be generalisable. The largest WARC file is 593, 684, 727 bytes, the smallest 4, 352, the average WARC file is 5, 033, 449 bytes.

Figure 1 shows the total archive size for different compression algorithms. The WARC specification recommends that files are compressed using gzip. Applying this strategy to our archive compresses the files down to 30.63897% of the original. We see that applying a bzip2 algorithm would compress the archive down to 25.79215%. Figure 2 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 71.15207% reduction in total archive size. Figure 3 shows the total archive size attainable when using a combination of delta and compression algorithms. 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 23.29247% of the original. This best performing strategy produces a collection 76.02238% of the size of that of the recommended compression strategy.

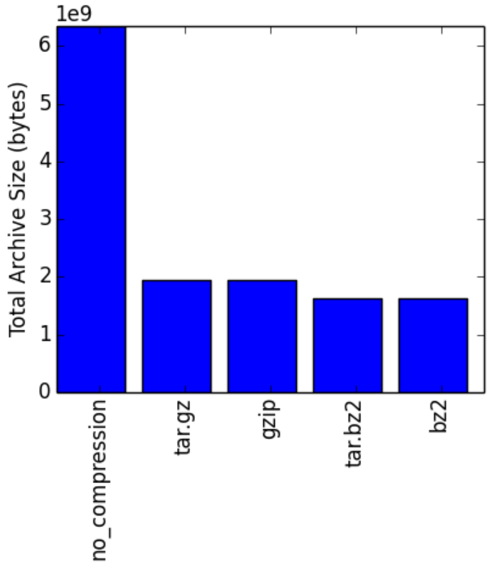

Figure 1: Compression comparison.

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 84.18089% 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 can be slower than gzip (in one benchmark, 4.5x slower to decompress compressed data). In a web archive setting, where long-term storage is a high priority, we be-

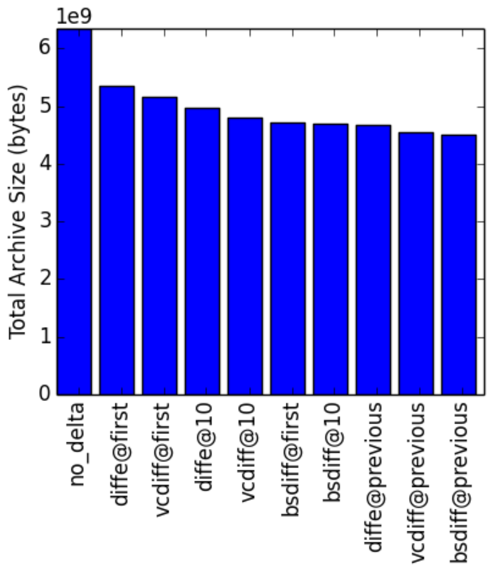

Figure 2: Delta comparison.

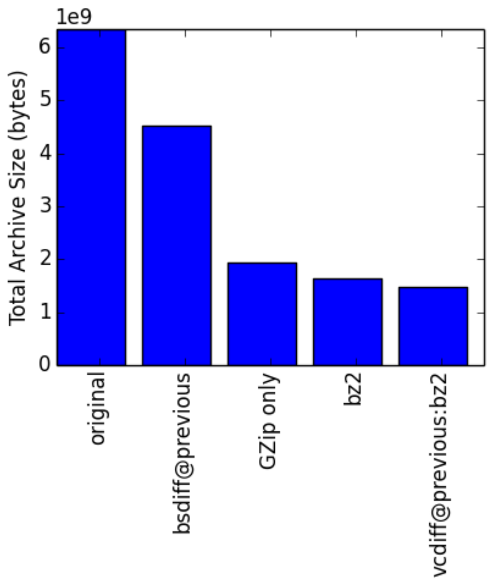

Figure 3: Total archive size for different delta and compression strategies.

lieve that bzip2 is the better choice of compression algorithm. The space savings it provides can be significant when considering the scale of an archive and the increased access time will be of negligible cost to the archive owner. When comparing delta algorithms we see that calculating the delta between a record and the immediately preceding record produces the smallest archive, no matter the delta algorithm. This is to be expected as it is likely that consecutive records will be quite similar in the majority of cases, producing smaller deltas. The downside to this strategy is that recreating a record requires access to every preceding record, each must be patched in turn. As a solution to this problem we introduced a reference frame every 10 records that is stored in full. Each record is compared to its closest preceding reference frame, recreating a record requires access to, at most, 10 previous records. Using reference frames introduces overheads, on average the archive size increased by 5.46623% when compared to strategies using immediately preceding records. A second alternative delta strategy is to compare every record to the first record ever recorded for a URI. In this case every record can be recreated using only one previous record (the first). We would expect this strategy to under perform because as changes are made to the file the delta between it and its original must get more complicated. This is the case for the diff and vcdiff algorithms, surprisingly bsdiff produces a smaller archive when comparing to the first record than when it compares to the reference frame. Using the bsdiff algorithm to produces deltas between a record and the first record creates an archive 4.27673% larger than that created when comparing to the previous record.

[6]e.g. `http://compressionratings.com/sort.cgi?`

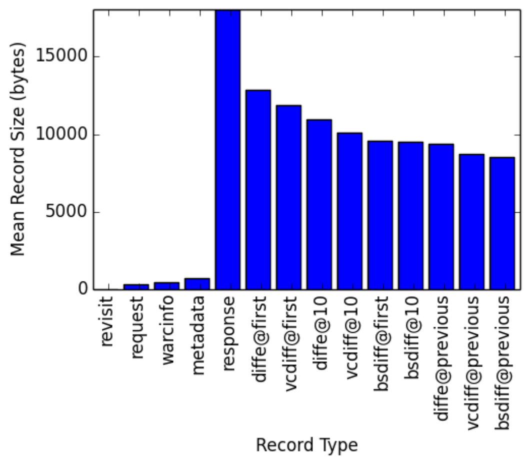

Figure 4: Size of records in WARC file by record type.

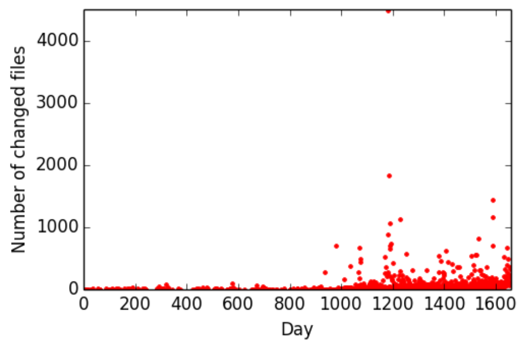

Figure 5: Changes during the first 1,659 days.

Figure 4 shows the mean size of each record type in our archive. For response records we also show the mean size after a delta strategy has been applied. The response records take up the majority of a WARC file. Without delta, the mean response record is 18042.12673 bytes, 24x the next largest record type, metadata. With bsdiff applied to response records the mean size drops by % to bytes, x the mean metadata size.

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

   (b) compression performance by content type

   (c) 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.

`txt1.brief+4np2`

4.3 Content Analysis

Figure 5 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 scheme. In the first $1,659$ days the mean number of files changing per day was 39.65521, in the last 215 that increases to 595.92558.

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

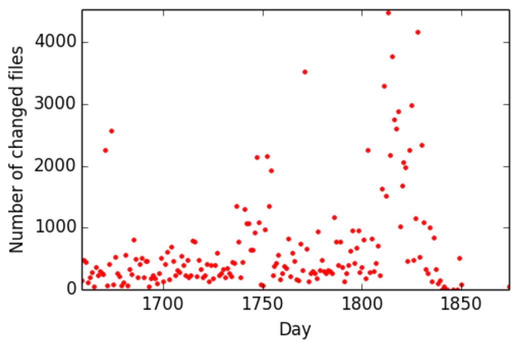

Figure 6: Changes during the last 215 days.

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 7 shows the number of domains that change in the archive per day. During the first segment we see a mean of 2.71609 domains changing per day, in the second segment this is 15.00000.

Figure 9 shows the total number of URIs ever observed at a domain. Figure 10 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 3, 102 of those URIs, on average, when

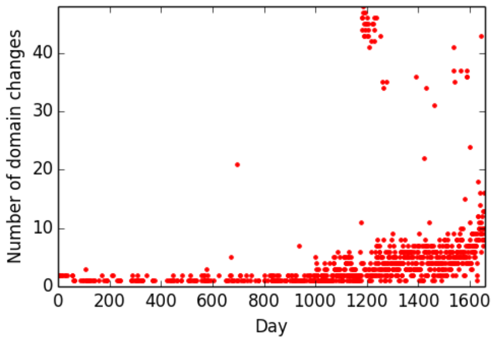

Figure 7: Domain changes during the first 1,659 days.

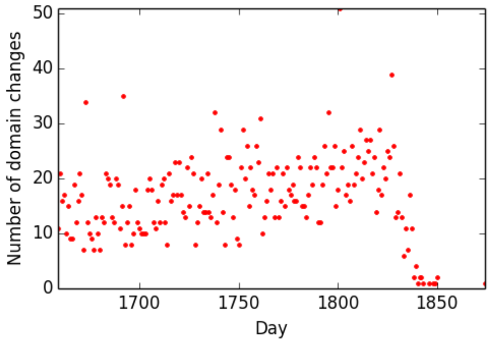

Figure 8: Domain changes during the last 215 days.

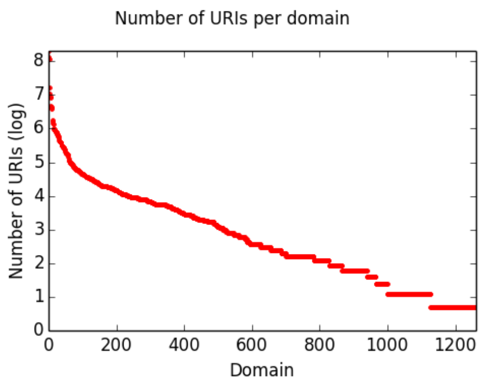

Figure 9: Total URIs observed at each domain.

we crawled it. The smallest domain, still3-still3.github.io, has 2 URIs, 2 on average.

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.

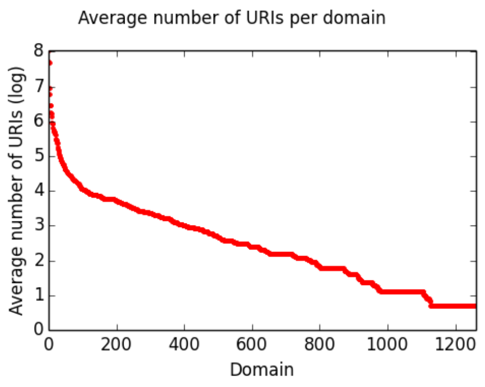

Figure 10: Average URIs observed at each domain.