The Case for Cross-Entity Delta Encoding in Web Compression (Extended)

Benjamin Wollmer[1,3], Wolfram Wingerath[2,3], Sophie Ferrlein[3], Fabian Panse[1], Felix Gessert[1,3], Norbert Ritter[1]

[1]*University of Hamburg, Germany, dbis-research@informatik.uni-hamburg.de*
[2]*University of Oldenburg, Germany, data-science@uni-oldenburg.de*
[3]*Baqend, Hamburg, Gemerany, research@baqend.com*

Abstract

Delta encoding and shared dictionary compression (SDC) for accelerating Web content have been studied extensively in research over the last two decades, but have only found limited adoption in the industry so far: Compression approaches that use a custom-tailored dictionary per website have all failed in practice due to lacking browser support and high overall complexity. General-purpose SDC approaches such as Brotli reduce complexity by shipping the same dictionary for all use cases, while most delta encoding approaches just consider similarities between versions of the same entity (but not between different entities). In this study, we investigate how much of the potential benefits of SDC and delta encoding are left on the table by these two simplifications. As our first contribution, we describe the idea of cross-entity delta encoding that uses cached assets from the immediate browser history for content encoding instead of a precompiled shared dictionary: This avoids the need to create a custom dictionary, but enables highly customized and efficient compression. Second, we present an experimental evaluation of compression efficiency to hold cross-entity delta encoding against state-of-the-art Web compression algorithms. We consciously compare algorithms some of which are not yet available in browsers to understand their potential value before investing resources to build them. Our results indicate that cross-entity delta encoding is over 50% more efficient for text-based resources than compression industry standards. We hope our findings motivate further research and development on this topic.

The extended version of our previously published paper [10] includes an additional section on the deltas of HTML files, a more detailed description of our approach (including a new visualization for the different dictionary strategies), a deeper discussion of compression efficiency, and details on additional future and ongoing work.

Keywords: Delta Encoding, Caching, Dictionary Compression.

1 Introduction

Every Web browser utilizes a local cache to reduce the payload of a website. But since cache entries are limited in their lifetime, they become useless in current schemes once they are stale. However, stale resources can still contain information that is useful for encoding related files efficiently. Delta encoding is an example of such an approach, which is generally used to update one entity to its newest version by sending a diff rather than the whole asset. Most proposals revolving around this mechanism focus on the similarities between versions of the same entity (single-entity data encoding).

In this work, we argue that modern websites comprise many pages that are very similar among one another (e.g. different product pages) and therefore lend themselves to *cross-entity delta encoding* as well [11]. We use Compaz [9] to evaluate the concept of cross-entity delta encoding and provide evidence on its potential benefits for payload savings to motivate further research on the topic. Sections 2 and 3 discuss and distinguish cross-entity delta encoding from existing work. In Section 4, we present quantitative results for the potential of cross-entity delta encoding to improve compression efficiency based on real-world high-traffic website traces. In Section 5, we drill down into HTML files and show the potential of different page types. Section 6, how different compression algorithms can further improve the shown approach. We discuss open challenges and conclude in Section 7.

This paper is an extended version of our previously published paper [10]. In this extended version, we further investigate the deltas of HTML files in a dedicated new Section 5. Furthermore, we explain our approach with additional details (including a new visualization for the different dictionary strategies in Figure 1), provide a deeper discussion of compression efficiency in Section 6, and present additional future and ongoing work in the outlook in Section 7.

2 Related Work

Delta Encoding. Mogul et al. proposed to use delta encoding in HTTP to update stale content [6], which is not implemented by any major browser. How well this scales depends on how much of the content changes between the two versions. They evaluated the delta calculation purely for updates of returning users, but did not consider deltas between different pages. Korn et al. proposed VCDIFF, a differencing algorithm [4]. They evaluated it similar to Mogul et al. by considering deltas between updates of the same file. Cloudflare's Railgun uses delta encoding to update the CDN content [3]. This approach is limited between server and CDN and also only considers updates between different versions.

Shared Dictionary Compression (SDC). In a standard compression approach, like with deflate, the encoder reads the file and tries to find repeating strings from the previously read content. The previously read content is also referred to as the dictionary. Instead of just using the previously seen content, the dictionary could also be an external file. In SDC, the same dictionary is shared between multiple compression processes and can therefore improve the overall compression ratio further. Chan et al. suggest that Web pages with a similar URL path also may have similar content and Web pages may therefore be transmitted more efficiently as a differentials to previously visited Web pages [2]. They only consider HTML files and assume them to be uniquely identifiable by URL. While the approach is similar to ours, the presented results are not applicable to modern websites. First, today's HTML files are often personalized and thus not uniquely identifiable by URL. Second, some assets are static and uniquely identifiable by URL (e.g. JavaScript or CSS), but they are not considered. Butler et al. proposed Shared Dictionary Compression over HTTP (SDCH), where the server can actively push dictionaries to the client [5]. One of the key challenges here is to find the best dictionary, since the server has to predict which dictionary would be of use for the client. This dictionary may increase the payload for the first page, since it is pushed, but maybe only used later. LinkedIn reported that generating the dictionaries took them about 7 hours per deployment and could easily take days [7]. Therefore, they were forced to delay the generation of new dictionaries to every other week. This may be one of the reasons why SDCH was not widely adopted and removed from Chrome[1]. However, the key idea of sharing a dictionary led to Brotli, which was later developed at Google [1]. For Brotli, the shared dictionary is static and already part of the library

[1] groups.google.com/a/chromium.org/d/msg/blink-dev/nQl0ORHy7sw/HNpR96sqAgAJ

and never has to be generated or transferred over the network. Most browsers support Brotli, but without the custom dictionary functionality which could be used for cross-entity delta encoding. Zstandard would allow the same, but has no browser support at all.

3 Cross-Entity Delta Encoding

As shown in the previous section, delta encoding has so far mostly been evaluated to compute deltas between different versions of the same file or with a shared dictionary. Calculating deltas between files that share similar data could provide similar advantages. In contrast to SDCH, this would remove the need to create and maintain dictionaries as we use the raw files as dictionaries. This has some implications. First, the compression for one asset may deliver different results for different users, since the result depends on the dictionaries available in the client cache. Second, using client cache entries requires a cache state synchronization as the server needs to know which resources can be used for content encoding.

Dictionary Scope Strategies. As a basic rule, only those assets can be used for encoding, which have already been loaded by the client. We therefore consider three different strategies for our evaluation(see Figure 1). As the most powerful strategy, one could consider every previous asset (PA) as a potential dictionary, which was seen until the currently requested asset. This includes previously visited pages (page impressions, PIs) as well as assets from the currently requested PI. This strategy is difficult to implement in practice, because assets are typically not downloaded in sequence. We exclude the currently processed PI assets as another more practical strategy and only consider fully downloaded assets up to the previous PI (PP). As a third strategy, we exclusively consider assets of the entry page as dictionaries (EE). Due to a similar overall layout (e.g. same header), this could be a reasonable alternative with a fixed set of dictionaries, compared to the ever-growing dictionaries of previous strategies.

4 Compression Efficiency on Real-World Traces

In this Section, we examine how cross-entity delta encoding could affect the transferred size within a user journey. We start by creating a dataset collected from real websites and used them to compare different compression approaches.

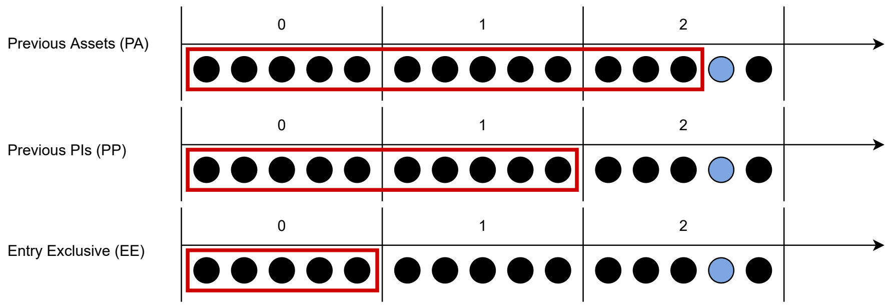
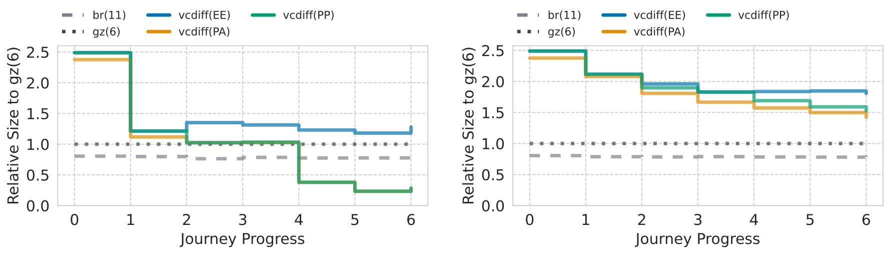


Figure 1: Visualization of the different strategies. The numbers indicate the current step in the journey, and the bubbles visualize the assets belonging to the step. The blue bubble indicates the current asset to be loaded, and the red box shows which assets are considered to be a dictionary.

Creating a Dataset. The potential of cross-entity delta encoding relies on the journey taken by a user, since it defines which dictionaries are in the cache. We created artificial ones, since we have no access to real user journeys. We assume that every website offers different kind of page types of which one is the main page type (e.g. a product in a shop or an article of a news/blog site). The other types could be types like a category site or the homepage. We further assume the users ultimate goal to be the content of this main page type. We start the navigation at the homepage and from there we try to hit different page types for Step 1 and 2. Step 3 then navigates to the main content. We name this part of the journey from now on the *cold phase*, since we hit distinct page types and the cache is cold in terms of available dictionaries. Step 4-6 are only recommendations from the previous main content and therefore navigate over potentially similar pages. In contrast, this path is from now on referred to as the *hot phase*, since it contains possible dictionary matches. We make sure that every step in the whole journey is unique. We used 40 of the traffic-heaviest websites according to SimilarWeb[2], providing the recommendation functionality.

Data Cleaning. This work focuses on text based content, therefore, we excluded every non-text asset, identified by the content-type header. We removed trivial cases, like the delta between two identical assets, since this is entirely preventable and would skew the results in favor of cross-entity delta encoding. Finally, we removed every third-party asset (other domain), since

[2] https://www.similarweb.com/de/top-websites/



(a) Relative Size per Step (b) Cumulative Relative Size

Figure 2: The left chart shows the compression size relative to gzip (6), while the right chart shows the cumulative size up until each step to account for different page sizes. Replacing every compression with VCDIFF only pays off in the hot phase (>3), but is on average still worse than the default gzip compression.

the provider would not have the possibility to change the compression for these kind of assets.

Calculating Deltas. The compression for the delta was done with open-VCDIFF to which we from now on will refer to as VCDIFF. We just brute-force every possible delta for a given asset and chose the smallest one. We only considered dictionaries which had at least the same type[3], e.g. text.

Comparing the Results. The data we collected was compressed by either Brotli, gzip, or no compression. To create a baseline against which we can compare, we uncompressed every asset and compressed it with the default gzip compression level (6), which on average is slightly higher than the results we got from the server. We also compare against Brotli (11), which may be impractical due to performance reasons, but represents the currently best compression ratio.

4.1 Enforcing VCDIFF

In our first experiment, we forced VCDIFF with the different strategies on every asset. It shows that regardless of the current step within our journey, just using the entry site will be outperformed in every step by gzip (see Fig. 2a). The results for PP and PE are similar in the cold phase and on par with gzip after leaving the entry page, but significantly improve and surpass even Brotli when entering the hot phase, with as low as 28% of gzip's size. This

[3] Still, the best dictionary was almost every time of the same subtype, e.g. text/html.

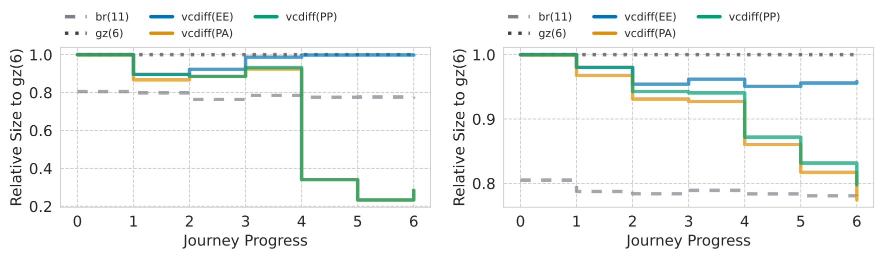
was expected, since we only look at previously visited page types. Still, due to the negative impact at the beginning, they cannot compete over the whole journey (see Fig. 2b).

4.2 Case-Specific VCDIFF

The previous experiment has shown that the results highly depend on the client cache. Therefore, we repeated the experiment but decided per asset to either use VCDIFF if it yields an improvement or stick with gzip (6) otherwise. Since the first PI still cannot leverage any dictionary, it mainly falls back to gzip (see Figure 3a). Just using the entry page as the dictionary source can slightly improve the compression ratio overall (~5%), but as Figure 3a indicates, this is mainly driven by similarities within the cold phase. Again, choosing the dictionary from previous assets provides the biggest impact within the hot phase and vastly outperforms Brotli. Stepping back to PP only slightly decreases the impact and can be an alternative to PA. Overall, using VCDIFF on specific assets scales well with the length of the journey, as soon as similar pages are visited. As Figure 3a and 3b show, this will converge against 25% of the payload for longer journeys.

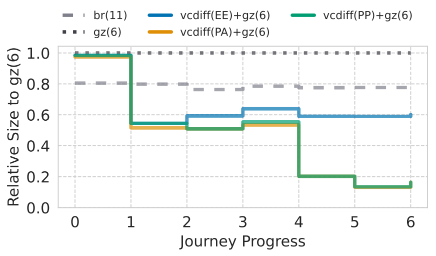
4.3 VCDIFF With Secondary Compression

VCDIFF files can still contain many common strings and could therefore benefit from secondary compression, which is currently not implemented in open-VCDIFF. This motivated our next experiment. We used VCDIFF as an

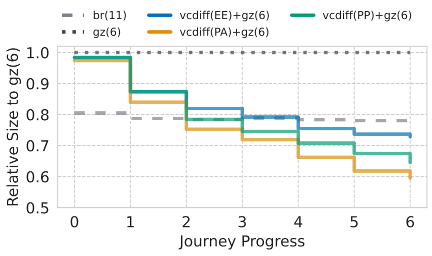


(a) Relative Size per Step (b) Relative Size per Step (Cumulative)

Figure 3: Using VCDIFF only when it actually provides an uplift yields small results in the cold phase (<4), but can even further improve the hot phase and overall eventually leads to results comparable with the maximum Brotli compression.



(a) Relative Size per Step



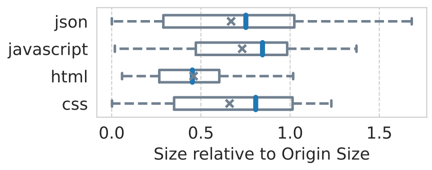
(b) Relative Size per Step (Cumulative)

Figure 4: While VCDIFF is still not competitive on the first page load, combining it with secondary compression outperforms Brotli on every following page load.

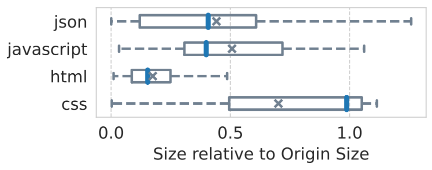
Opt-in on every text asset, but piped the VCDIFF output through gzip on level 6. Figure 4a shows that this approach drastically improves the results, as we now outperform Brotli even before entering the hot phase. Within the hot phase, we can compress the assets as low as 14% of gzip (6). Overall the cumulative size can be reduced to 58% of our baseline (see Fig. 4b).

4.4 Impact on Different MIME-Types

We expected the HTML to gain the most benefit of cross-entity delta encoding and compared them with the other types. We grouped them by the step, as well as the subtype to make sure that the actual weight of the single assets were reflected. For the cold phase, we exclude the entry pages (step 0), since the previous experiments have already shown that cross-entity encoding is no



(a) Cold Phase (Excluding Entry)



(b) Hot Phase

Figure 5: Compression efficiency by MIME type: Using HTML for cross-entity delta encoding works well in almost all cases. Using other MIME types leads to mixed results in the cold phase (a), but yields high efficiency in the hot phase (b).

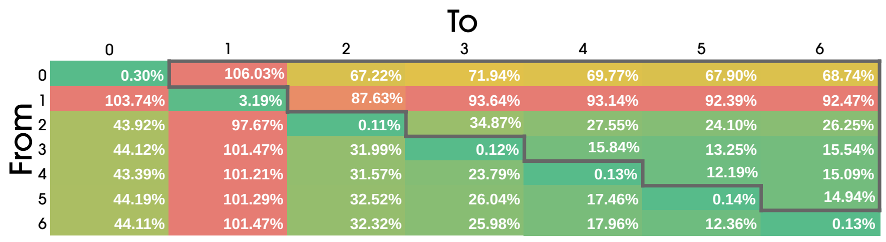


Figure 6: The heatmap shows the mean compression ratio of deltas from page type to another, only including the HTML files.

real alternative in that step. HTML benefits the most from cross-delta entity encoding (see Figure 5) and is a safe alternative in the hot phase. We excluded XML and plain text responses, since we had too few samples. SVGs could in some cases further reduced by 10% compared to gz (6). While the other types on average can still greatly benefit, they are site-specific cases and need further investigation. Note that the hot phase had only a few CSS samples and may not be representative, since most sites do not load additional CSS files at this point.

5 Optimizing HTML

As shown in Section 4.4, delta encoding is highly effective for HTML compression. In this section, we will closely examine the effects of delta encoding on HTML files.

Figure 6 indicates that the entry page can act as a dictionary for most page types (except page type 1) which is most likely due to a similar header and footer on the pages. This chart also confirms the suspicion that product pages are excellent dictionaries for other product pages. Furthermore, the category page (type 2) is, on average, another good dictionary for the product pages. This is no surprise since category pages are likely to show a preview of products and therefore share some similarities in stylings, which may be reusable. Figure 5 also includes deltas that were redundant for our journey, like the delta from page 6 to page 0, since we already visited page 0 at this point and did not have to fetch it again. We still included the redundant results to see if there is a page type that acts an excellent dictionary for all other page types, which was not the case.[4]

One of the takeaways from Figure 6 is that delta encoding provides a significant efficiency benefit over plain gzip on average, no matter from what product page to what other product page the user navigates. Our experiments indicate that this is not only the case on average. The box plot in Figure 7 shows the variation of compression ratio between the best and the worst possible choice for a delta, illustrating that basically any product page can be used to compute a delta for any other product page: In other words, the figure shows by how much the compression ratio would have suffered, if the worst-possible delta in that journey would have been chosen instead of the best-possible delta. Compression ratio was close to the gzip baseline only in a single case (far right), while the difference between best and worst was neglectable with around 7% on median and still below 15% for three quarters of all journeys.

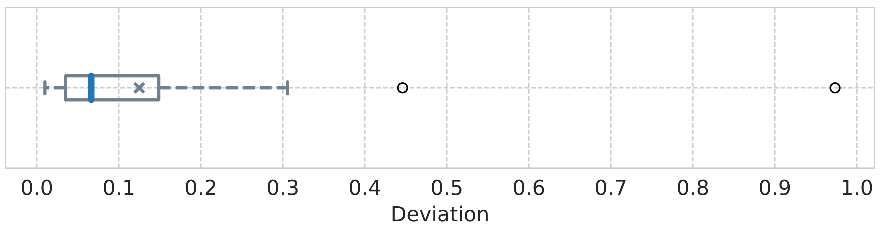


Figure 7: The box plot shows the deviation of compression ratio for all journeys, comparing delta encoding with the most efficient delta against delta and coding with the least efficient delta. Even in extreme cases, using the worst-possible delta is comparable with simply loading the target resource via other baseline gzip compression.

6 Using Algorithms Beyond VCDIFF

Combining gzip (6) with VCDIFF already could on average reduce the journey size to 58%. But compression efficiency could be further improved by increasing the compression level and/or using different algorithms. We are

[4] The diagonal line shows the delta to itself, which is completely useless, but we included it for the sake of completeness.

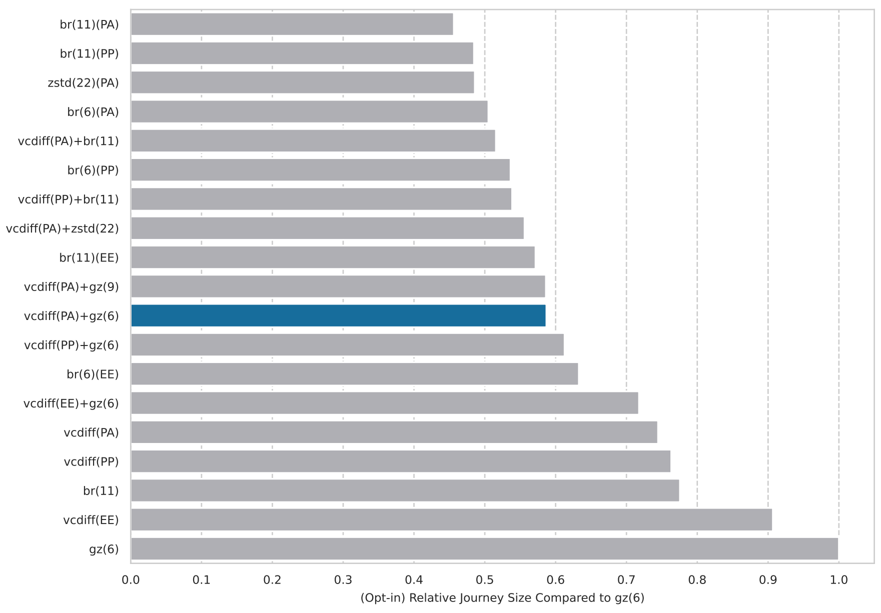


Figure 8: Replacing the secondary compression yields even better results. Alternatively, Brotli and zstandard also allow a more efficient use of custom dictionaries.

aware that a higher compression level may be impractical in an actual deployment, but should act as an upper limit. While increasing gzip to level 9 had almost the same result, replacing gzip (6) with Brotli (11) reduces the cumulative journey size to 51% (see Figure 8). As mentioned earlier, one could also directly use a custom dictionary with Brotli or zstandard (which is not supported by any browser). This reached the highest compression ratio and could reduce the result to 45% for Brotli (11) and 48% for zstandard (22), compared to gzip (6)[5]

As previously shown, the product pages are highly compressible with delta encoding. Figure 9 shows additional compression algorithms on these

[5] Due to limited space, we only present a few selected alternatives here and refer to https://icwe.compaz.info for an extensive overview.

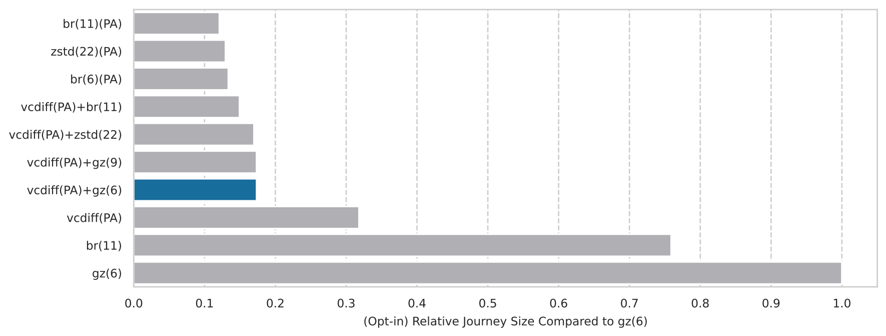


Figure 9: Looking only at the HTML of recommendations shows that the uplift between VCDIFF and the other types is still existing, but not as significant.

HTML files. However, since we are already at such small file sizes, the overall effect is less significant than other file types (compare figure 8). [6]

7 Conclusion

Our results show that reusing cached assets for delta encoding can significantly reduce transferred bytes in the Web, even though we simulate new users who start with cold caches. This approach should thus be seen as a complement to existing compressions rather than a replacement. But it should be noted that potential uplift is even more significant for returning user who start their journeys in the hot phase and thus directly benefit from cross-entity delta encoding.

Open Challenges. In our experiments, we made several simplifying assumptions that do not hold in a real-world setting. First, we employ the perfect dictionary selection via a brute-force approach, but a more efficient heuristic would be required for practical use. Also, cache state synchronization remains challenging: The server does not only have to select the ideal dictionary for encoding, but also one that is already present in the client cache to enable decoding. Another open challenge is the lack of browser support for different aspects of delta encoding and shared dictionary compression. While all major browsers support generic Brotli and gzip, VCDIFF and Brotli with

[6] We only consider PA since we only look at HTML files; therefore, the result would be the same for the other two strategies.

a custom dictionary are currently not supported by any of them. While using HTML files as dictionaries was most effective in our evaluation, content that is generated per user makes it infeasible to keep all dictionaries (HTMLs) in the server. Reducing this complexity would require some kind of normalization to strip personlized content for encoding and decoding (cf. Dynamic Blocks [8]), akin to app shells in single-page applications.

Closing Thoughts. Despite a host of literature on delta encoding and shared dictionary compression from more than two decades of research, there is still a lot of untapped potential in existing compression technologies. Our results indicate that using the client cache as a dictionary for delta encoding can reduce the text payload by up to 86% for single pages and by 55-80% for user journeys over recommended content. But there is still further research needed in areas like dictionary selection and cache state synchronization. Lacking browser support for cross-entity delta encoding algorithms is another practical barrier, but could be added in platform-independent fashion with a service worker implementation. However, performance depends on the client device and is likely not comparable with native compression algorithms. Without native browser support, delta encoding only seems viable for scenarios where network efficiency is critical (e.g. for mobile users in data saving mode).

Finally, as shown in Section 5, almost every product was an excellent dictionary for all other product HTML files. This can significantly reduce the overhead of choosing a suitable dictionary for product pages. We plan to combine this observation with predictive preloading of HTML files by choosing a popular product as a dictionary that gets downloaded by the client so that every following product is a delta of this preloaded product. This is similar to SDCH[5], but the dictionary is also usable as a product preload since it is an actual HTML file. Furthermore, since the deltas have the size of a fraction of the current state-of-the-art compression, this will allow us to either safe bandwidth for product preloads or increase the cache hit rate by preloading multiple products for the cost (in terms of bytes) of one page.

References

[1] Jyrki Alakuijala, Andrea Farruggia, Paolo Ferragina, Eugene Kliuchnikov, Robert Obryk, Zoltan Szabadka, and Lode Vandevenne. Brotli: A general-purpose data compressor. *ACM TOI*, 37(1).

[2] Mun Choon Chan and T.Y.C. Woo. Cache-Based Compaction: A New Technique for Optimizing Web transfer. In *IEEE INFOCOM '99. Conference on Computer Communications.*

[3] Dane Orion Knecht, John Graham-Cumming, and Matthew Browning Prince. Method and apparatus for reducing network resource transmission size using delta compression.

[4] David G Korn and Kiem-Phong Vo. Engineering a differencing and compression data format. In *USENIX annual technical conference, general track*, pages 219–228, 2002.

[5] Bryan McQuade, Kenneth Mixter, Wei-Hsin Lee, and Jon Butler. A proposal for shared dictionary compression over http. 2016.

[6] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential Benefits of Delta Encoding and Data Compression for HTTP. *SIGCOMM CCR*, 1997.

[7] Omer Shapira. SDCH at LinkedIn., 2015. Accessed: 2022-01-20.

[8] Wolfram Wingerath, Felix Gessert, Erik Witt, Hannes Kuhlmann, Florian Bücklers, Benjamin Wollmer, and Norbert Ritter. Speed Kit: A Polyglot & GDPR-Compliant Approach For Caching Personalized Content. In *ICDE, Dallas, Texas*, 2020.

[9] Benjamin Wollmer, Wolfram Wingerath, Sophie Ferrlein, Felix Gessert, and Norbert Ritter. Compaz: Exploring the Potentials of Shared Dictionary Compression on the Web. In *22th International Conference on Web Engineering, ICWE*, 2022.

[10] Benjamin Wollmer, Wolfram Wingerath, Sophie Ferrlein, Felix Gessert, and Norbert Ritter. Compaz: Exploring the potentials of shared dictionary compression on the web. In *Proceedings of the 22nd International Conference on Web Engineering (ICWE)*, 2022.

[11] Benjamin Wollmer, Wolfram Wingerath, and Norbert Ritter. Context-Aware Encoding & Delivery in the Web. In *20th International Conference on Web Engineering, ICWE*, 2020.

Biography

Benjamin Wollmer is a data engineer at Baqend as well as a PhD student at the databases and information systems group (DBIS) at the University of Hamburg. His PhD thesis is supervised by Norbert Ritter and his research interests revolve around efficient data transmission and compression algorithms in the Web. As a data engineer at Baqend, Benjamin is also part of the team that develops and operates the real-user monitoring and analytics solution built into Speed Kit to generate performance-critical insights.

Wolfram "Wolle" Wingerath is professor for data science at the University of Oldenburg (UOL) and Data Science Advisor at Baqend. Before joining UOL in 2022, Wolle headed Baqend's data engineering team and was responsible for developing and operating Baqend's real-user monitoring pipeline for zero-latency analytics. His research interests revolve around data-intensive applications and high-performance web infrastructure, but he also has a passion for hands-free coding to increase productivity (https://handsfree-coding.gi.de). Wolle published several books, articles, and tutorials on these topics together with his colleagues and frequently talks about his research at scientific and developer conferences.

Sophie Ferrlein is a Data Scientist at Baqend where she turns web tracking data into actionable web performance insights and data products. Based on her B.Sc. in Computer Science for Media Applications and her B.A. in Journalism, she focuses her professional efforts on communicating data effectively.

Fabian Panse currently heads the databases and information systems group (DBIS) at University of Hamburg as a substitute professor where he has been a postdoctoral researcher before. Fabian has been doing research in the fields of deduplication, uncertain data management and data quality since 2009. During this time, he wrote several papers that address the problems of measuring data quality, evaluating duplicate detection algorithms, and test data generation.

Felix Gessert is the CEO and co-founder of the Backend-as-a-Service company Baqend. During his PhD studies at the University of Hamburg, he developed the core technology behind Baqend's Web performance service. Felix is passionate about making the Web faster by turning research results into real-world applications. He frequently talks at conferences about exciting technology trends in data management and Web performance. As a Junior Fellow of the German Informatics Society (GI), he is working on new ideas to facilitate the research transfer of academic computer science innovation into practice.

Norbert Ritter is the Dean of the Faculty for Mathematics, Informatics, and Natural Science of the University of Hamburg, but headed the databases and information systems group (DBIS) as a full professor until August 2022. Norbert received his PhD from the University of Kaiserslautern in 1997. His research interests include distributed and federated database systems, transaction processing, caching, cloud data management, information integration, and autonomous database systems. He has been teaching NoSQL topics in various database courses for several years. Seeing the many open challenges for NoSQL systems, he, Felix, Wolle, and Benjamin have been organizing the annual Scalable Cloud Data Management Workshop (https://scdm.cloud) to promote research in this area.