Information Processing and Management 51 (2015) 114–129

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

Building a better mousetrap: Compressing mouse cursor activity for web analytics

Luis A. Leiva [a,*], Jeff Huang [b,1]

[a] PRHLT Research Center, Universitat Politècnica de València, 46022 Valencia, Spain
[b] Computer Science Department, Brown University, Providence, RI 02912, United States

ARTICLE INFO

Article history:
Received 26 December 2013
Received in revised form 16 October 2014
Accepted 22 October 2014

Keywords:
Mouse cursor tracking
Web analytics
Temporal-spatial data compression
Data simplification

ABSTRACT

Websites can learn what their users do on their pages to provide better content and services to those users. A website can easily find out *where* a user has been, but in order to find out *what* content is consumed and *how* it was consumed at a sub-page level, prior work has proposed client-side tracking to record cursor activity, which is useful for computing the relevance for search results or determining user attention on a page. While recording cursor interactions can be done without disturbing the user, the overhead of recording the cursor trail and transmitting this data over the network can be substantial. In our work, we investigate methods to compress cursor data, taking advantage of the fact that not every cursor coordinate has equal value to the website developer. We evaluate 5 lossless and 5 lossy compression algorithms over two datasets, reporting results about client-side performance, space savings, and how well a lossy algorithm can replicate the original cursor trail. The results show that different compression techniques may be suitable for different goals: LZW offers reasonable lossless compression, but lossy algorithms such as piecewise linear interpolation and distance-thresholding offer better client-side performance and bandwidth reduction.

1. Introduction

Websites that know what their users do on their pages can provide better content and services to those users. For example, search engines can re-rank search results using what people click as implicit feedback (either personalizing results for individuals from their click history, or using aggregated data from past searches to improve the overall ranking), e-commerce sites can learn what parts of the page deter potential customers, and social networking sites can use aggregated usage metrics to improve the usability of their application. A website can easily find out *where* a user has been on their pages through server access logs, but this yields an incomplete picture of what their users were actually doing. To find out *what* content is consumed and *how* it was consumed at the page level, the website can use client-side tracking to record richer interactions such as cursor movements and hovering, scrolling activity, and text highlighting. These interactions can be interpreted into higher-level behaviors like reading and marking interesting text with the cursor, quickly skimming the entire page, or moving the cursor out of the way to the side. To this end, page-level interactions provide researchers and practitioners with a way to gain additional insight of users' Web browsing behavior.

* Corresponding author. Tel.: +34 963878172.
E-mail addresses: llt@acm.org (L.A. Leiva), ipm@jeffhuang.com (J. Huang).
[1] Work partially conducted at the University of Washington.

http://dx.doi.org/10.1016/j.ipm.2014.10.005

Previous literature has shown value in using mouse cursor interaction data for applications such as: determining the relevance of search results (Guo & Agichtein, 2012; Huang, White, Buscher, & Wang, 2012b; Speicher, Both, & Gaedke, 2013), to track what users are reading on a page (Diaz, White, Buscher, & Liebling, 2013; Guo & Agichtein, 2010b; Hauger, Paramythis, & Weibelzahl, 2011; Hauger & Van Velsen, 2009), user modeling (Buscher, White, Dumais, & Huang, 2012; Leiva & Vidal, 2010), or potentially as a proxy for gaze tracking (Huang, White, & Buscher, 2012a; Huang, White, & Dumais, 2011; Rodden, Fu, Aula, & Spiro, 2008). Many commercial Web analytics services allow websites to track their users' mouse cursor interactions: ClickTale, LuckyOrange, MouseFlow, Mpathy, and Clixpy. Once set up on a website, the analytics service allows Web developers the ability to replay cursor movements from a user session, or generate heatmaps with aggregated cursor positions.

While recording cursor interactions can be done without interrupting the user, the overhead of recording the cursor movements and transmitting this data to the server can be substantial. For instance, swiping the mouse from left to right can generate a hundred {x, y} cursor coordinate pairs, which over a minute of interaction can lead to nearly 1 MB of data being sent to the analytics server. Compressing cursor activity is challenging because current transport protocols have no mechanism for compressing this type of high-frequency client-side generated data. In our work, we investigate methods to compress the trail a cursor makes on a page, taking advantage of the fact that not every cursor coordinate has equal value to the web developer. This problem is particularly important for situations where bandwidth is limited, such as on a mobile network or in developing countries. And while there are a limited number of services offering cursor tracking features, having efficient cursor tracking methods will benefit all Web users. One Web analytics company may be used by a thousand websites which in turn may serve a million users.

The contribution we make in this paper is a rigorous evaluation of 10 compression algorithms for 3-dimensional data (x-coordinate, y-coordinate, and time), evaluated with 4 metrics that represent different needs for the compressed data. The evaluation is conducted across both a dataset collected from a lab study and a dataset from a live web page, both datasets involving real users. We show that different compression techniques may be useful in different situations; the situations can reflect a desire for consuming less bandwidth, better client-side performance, more accurate replication of the original data, or a combination of all three objectives. With the reduction in data size, tracking mouse cursor interactions can finally become a practical, scalable technology. As a secondary contribution, we share the dataset we collected and the implementations of the compression algorithms we used in our experiments.[2] This will allow commercial analytics services to build on top of our work, and for other researchers to replicate our results.

2. Related work

Mouse cursor activity has been applied to different applications in prior work. An early study by Chen, Anderson, and Sohn (2001) found a relationship between cursor positions and where people were looking, suggesting a potential for using cursor tracking to substitute eye-tracking. Hauger et al. conducted a study over instructional pages about the game of "Go" (Hauger et al., 2011). They found that gaze and cursor positions were better correlated when the cursor was in motion and in sessions comprising a higher proportion of motion. Hauger et al. were able to predict which paragraph a user was reading, and to what extent, with 79% accuracy. Guo and Agichtein show some utility of the cursor position as a proxy of where a user is looking (Guo & Agichtein, 2010b) but Huang et al. (2012a) caution against assuming that the cursor approximates eye-gaze, as this is often not the case depending on time and the user's current action.

In information retrieval, cursor tracking can help determine the relevance of search results (Guo & Agichtein, 2012; Huang et al., 2011; Speicher et al., 2013). When a user is focusing on particular results with their cursor, even when not clicking, it may be a signal to the search engine that the search result is relevant to the user. Huang et al. take this further and incorporate cursor hovering and scrolling into a user model that is shown to help label search results for relevance (Huang et al., 2012b). Other researchers have developed user models as well for clustering web documents (Leiva & Vidal, 2010), inferring the reason a user abandons a search (Diriye, White, Buscher, & Dumais, 2012), predicting user attention in novel page layouts (Diaz et al., 2013), restyle the design of a website (Leiva, 2011), determining whether a user's shopping goal (Guo & Agichtein, 2010a), and identifying different types of searchers (Buscher et al., 2012). Additionally, since cursor interactions span a variety of behaviors, a website can learn more about what content attracts their users when the cursor is used to read, highlight, or mark text (Hauger & Van Velsen, 2009; Liu & Chung, 2007; Rodden et al., 2008). Broader interactions may also be interesting to website developers, such as which regions a user hovers over, when the user moves the cursor out of the way, how they fill out forms, or move towards scrolling and clicking (Cooke, 2006; Rodden & Fu, 2007).

2.1. Cursor tracking systems in academia

To our knowledge, Mueller and Lockerd developed the first approach to capture cursor data in the research literature (Mueller & Lockerd, 2001), examining the behavior of 17 participants. However, as an extended abstract, they provide little description of their system architecture, a section comprising two sentences, "Our system posts mouse movement data

[2] See http://hci.cs.brown.edu/mousetrap/.

(position, time) automatically with embedded scripting, and the data is analyzed and stored on a server. This collection technique is implemented using current technology and does not require any additional software on the user's browser." Cursor movements of 105 participants were collected for the purpose of usability evaluation by Arroyo, Selker, and Wei (2006). They used a distance-based approach to sampling the cursor coordinates, storing in a database tuples of cursor coordinates with their timestamps whenever the mouse moved outside an R px radius. In small-scale deployments such as these studies, the researchers did not have to worry much about the transmission and storage of the cursor activity. Atterer et al. captured "the user's every move" using an HTTP Proxy which were client-side interactions together with information related to DOM elements (Atterer, Wnuk, & Schmidt, 2006). Although they tested this on only 12 participants, the authors had the scalability of the system in mind: "capturing every change and sending it to the proxy for logging would not only lead to significant network overhead and a bloated log file, but also to a tremendous overhead on the client side", and decided to do so whenever a periodical scroll or a cursor event was detected.

Guo and Agichtein implemented a script in a browser toolbar and deployed their tracking system at the university library over a one month period. To reduce the amount of data captured, they took a simple approach of capturing 1 out of every 10 cursor coordinates, discarding the rest (Guo & Agichtein, 2008). Leiva and Vivó developed smt2ϵ, a cursor tracking system that captures the same information as the above systems but polls at regular time intervals in order to construct replays of user interactions (Leiva & Vivó, 2013). Guo and Agichtein (2010b) used a combination of these methods, based either on distance (every 5 pixels moved) or on polling (every 50 ms), whichever was more frequent. In a tracking script deployed in a commercial search engine, Huang et al. recorded cursor coordinates during pauses (>40 ms) rather than a continuous stream of cursor information, which yielded a close approximation of where the user cursor had been (Huang et al., 2011). In another study involving hundreds of thousands of searchers, Huang et al. combined paused-based sampling with the previously mentioned distance-based sampling on a large scale (Huang et al., 2012b). These cursor tracking systems that recorded data on a scale larger than a lab study used more aggressive sampling methods to reduce the data captured and to avoid spoiling the user experience.

These prior techniques show that no researcher captured cursor activity in their entirety (with the possible exception of Mueller & Lockerd (2001)), and relied on some sort of sampling. Other than Leiva and Vivó (2013), there is no past work that has considered cursor compression issues. In particular, there is a gap in the literature for understanding bandwidth requirements and how they might interact with various compression schemes. The past research each decided *ad hoc* which sampling method to use; our work takes a systematic approach to identifying the differences between a number of compression methods (many of which were used in the prior literature for cursor tracking) and evaluating the fundamental methods to address this choice.

2.2. Cursor tracking systems in industry

Many commercial web analytics services offer cursor tracking; e.g., Clicky, Piwik, Google Analytics, or LuckyOrange. Here, the developer must instrument the pages she wish to track by notifying the tracker which events should be captured, beyond regular clicks and page views. In order to support older browsers, these systems follow the technique described by Atterer et al. (2006), i.e., requesting an image having a query string with the URL-encoded data that should be stored on the server. This is so because images are not subject to the same-origin policy of the Web security model; therefore this technique makes it possible to submit data across different domains. Once set up on a website, the analytics service allows Web developers the ability to replay cursor movements from a user session, or generate heatmaps with aggregated cursor positions.

Cursor tracking is central to other analytics services that focus on client-side interactions, such as ClickTale, Clixpy, Mpathy, UserFly, or MouseFlow. It is worth noting that most of these cursor tracking systems privilege XMLHttpRequest as data transport mechanism, but also fallback to the previously commented cross-domain image technique in order to support older browsers. In the following we describe how do these systems approach mouse cursor tracking.

ClickTale captures all browser events and transmits the data when reaching 1 kB, perhaps to keep request URIs short and thus avoid 414 HTTP error codes[3] Thus, tracking data are buffered on the browser and periodically submitted to the server. UserFly (now defunct) and MouseFlow both use time-based polling of 100 ms, similar to previous academic work Guo and Agichtein (2010b), Leiva and Vivó (2013). In contrast, Clixpy (now defunct) sends the tracking data each 3 s to a Flash object, instead of an image, in order to allow cross-domain requests. Mpathy uses Flash socket connections to send the data whenever a browser event is registered. Using Flash TCP sockets reduces the number of requests incurred by logging, as in this case only one (persistent) connection would be necessary. However, Flash technology is deprecated and today is unavailable for most mobile devices. This could be improved by using WebSockets, unfortunately this technology is not yet widely adopted. Further, using plain JavaScript event logging (as e.g. in ClickTale or MouseFlow) ensures that even older browsers are compatible.

In this article, we focus primarily on web-based applications. However, mouse cursor tracking is also used in multiple desktop applications. For instance, there are applications that track user interface usage (Ranorex), capture screen video (Camtasia), add visual effects to the cursor (Compiz), or produce artistic images from cursor patterns (IOGraph). Among

[3] http://www.faqs.org/rfcs/rfc2616.html.

```
ID  timestamp      x    y    event      target
0   1338500487565  343  248  mousemove  //*[@id='content']/div[1]/div
0   1338500487573  350  262  mousemove  //*[@id='content']
0   1338500487581  355  269  mousemove  //*[@id='content']
0   1338500487589  359  282  mousemove  //*[@id='content']/div[2]
0   1338500487597  365  298  mousemove  //*[@id='content']/div[2]
0   1338500487628  365  298  mousedown  //*[@id='content']/div[2]
```

Fig. 1. A snippet of an example log file comprising mouse movement events on a web page.

these, our work can benefit desktop software that requires transmitting cursor data over the network. For instance, applications for remote desktop sharing and virtual network computing (VNC) such as Bomgar, LogMeIn, GoToMeeting, or TeamViewer. While desktop applications do not necessarily have the same performance concerns as web applications, bandwidth can still be a limiting resource that can be reduced by compressing the data. For instance, mouse cursor tracking may cause massive lag in VNC clients if tracking is not carefully handled.[4]

3. Method

We first sought to find or create datasets comprising mouse cursor activity. We developed a client-side tracking script with a small footprint (around 5 kB) that was capable of logging cursor interactions. This script could be deployed on any website to record data seamlessly. Similar to other cursor tracking scripts, ours captured `mousedown`, `mousemove`, and `mouseup` events; as well as mobile interactions via `touchstart`, `touchmove`, and `touchend` events. Whenever the browser fired one of the above-mentioned cursor events, the script logged the following information in the background: cursorID (to differentiate between multiple cursors in multi-touch clients), timestamp at which the event occurred, `x` and `y` cursor coordinates, name of the event, and the DOM element that relates to the event, in XPath notation. Fig. 1 shows an example of the captured data. From our experience, this data is sufficient for replaying user interactions on a web page.

As a pilot test, we deployed the cursor tracking script on a prototype website, http://portlandtaxis.com, for several weeks. This was a website we built from scratch with some information about taxi companies in Portland, with a grid layout similar to many existing websites. We used this prototype to ensure that the cursor data was being properly collected, even using CORS (XMLHttpRequests from one domain to another), and that cursor trails could be recreated from the collected data. Fig. 2 illustrates an example user's cursor trails overlaid on the prototype website.

3.1. Data

We used two datasets in our evaluation to determine whether the compression methods were generalizable. One dataset was from a lab study, which we will refer to as LAB while the other was the tracking script deployed on a website accessible to the public, which we will refer to as LIVE. Both datasets are summarized in Table 1.

The LAB dataset was collected by Feild et al. during an eye-tracking study of web searcher frustration in October 2009 (Feild, Allan, & Jones, 2010). The dataset consists of full interaction logs and sensor readings for 30 participants who were asked to conduct search tasks. We removed 5 outliers from this dataset for users whose time at the lab spanned over two consecutive days, either because of a power outage or equipment malfunction. This left 25 participants in the dataset. During the study, coordinates for the mouse cursor were recorded by a toolbar that captured cursor positions when they changed and stored them in logs.

The LIVE dataset was collected from a live website with the script described earlier that captured raw cursor events in JavaScript. Events were buffered at regular time intervals and sent to our server (online tracking). The website was an informational resource listing the best paper awards in computer science; the content on the page was lengthy and included numerous clickable links to navigate the page and to search for the award-winning papers. The data was gathered between June and September 2012, and totalled 12 K visitor logs. We removed unreasonable outliers from this dataset by filtering those logs having a browsing time outside 1.5 the interquartile range (IQR). This technique (Tukey, 1977) allowed us to deal with more consistent observations from the logs gathered. The outliers were likely cases where users removed their attention from the page and may have left their computer altogether. The remainder of the data comprised 10,471 interaction logs from 7064 unique visitors.

There are some notable differences between the LIVE and LAB datasets. The users in the LAB dataset were brought into a lab, asked to perform search tasks, and the data was recorded with an instrumented toolbar. In contrast, the users in the LIVE dataset were browsing *in situ* in their natural environments and the data was recorded using a tracking script on the website. Furthermore, the automated method to capture cursor activity in the LIVE dataset is easier to scale to a large numbers of subjects.

[4] http://www.realvnc.com/pipermail/vnc-list/2009-September/059982.html.

Table 1
Summary statistics for the two datasets in our study.

Dataset	Sessions	Cursor trail size (kB)					Browsing time (min)				
		Min	Median	Mean	Max	SD	Min	Median	Mean	Max	SD
LAB	25	124	300.0	325.8	625	111.5	66	90.3	97.5	176	24.6
LIVE	10,471	0	8.0	19.7	804	36.8	0.1	0.6	1.5	12	2.2

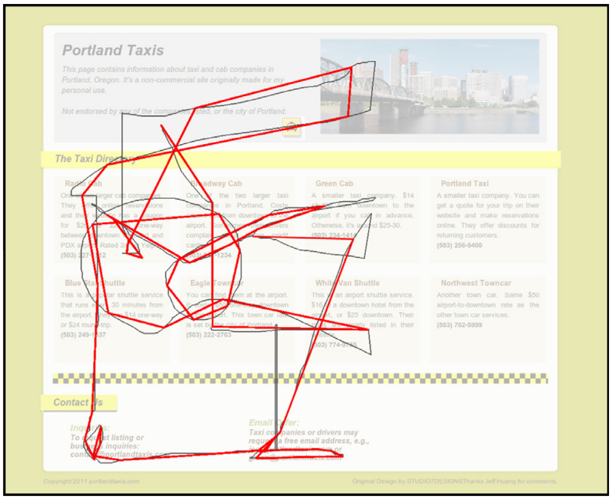

Fig. 2. An example of mouse cursor coordinates overlaid on the http://portlandtaxis.com website during a user browsing session. The thin gray line represents the actual movements made by the user's mouse cursor, and the thick red line represents a lossy-compressed version of the trail. As observed, most of the trail information is meaningfully preserved. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4. Compression

There are numerous ways in which the cursor data can be compressed before being sent to the server. The nature of the data to be encoded influences the effectiveness of a particular compression algorithm. Because of this, we selected lossy compression methods based on their actual usage in prior research involving cursor tracking (sampling), and lossless compression methods commonly used for reducing the size of web data. We cannot determine in advance which compression techniques will work best for cursor data, and therefore one of our contributions is the implementation and systematic evaluation of a diverse set of compression algorithms on the mouse cursor data sequences.

Note that mouse cursor data must be compressed on the client side in a separate script before being sent to the server because the web browser itself has no mechanism to compress uploaded data,[5] due to the lack of standard for detecting whether the server receiving the data supports compression.[6] Therefore, both the client and the server must "negotiate" how the data will be uploaded for each HTTP request, which requires an explicit HTTP connection (the "handshake") prior to decide a data compression mechanism. This can undermine the performance on the server side (see Section 6). Compressing cursor data on the client before sending them to the server is also convenient because anti-virus software, web proxies, and browser bugs can strip or mangle request headers; especially the Accept-Encoding header (Jain & Glasgow, 2012).

In principle, data compression can be approached with several independent methods; e.g., applying quantization, down-sampling, local encoding, removing redundancy, etc. However, there are two fundamentally different types of data compression: lossless and lossy. Lossless compression involves a transformation of the representation of a dataset such that it is possible to reproduce *exactly* the original dataset by performing a decompression transformation. Lossy compression is a representation that allows us to reproduce an approximation to the original dataset. In other words, lossless compression allows the server to recreate the exact cursor trail recorded, while lossy compression can typically compress the data into smaller files. The appropriate type of compression to use depends on the application of the mouse cursor data. We aim at addressing this choice with the following set of selected techniques.

[5] In contrast, downloaded content can be compressed because the Accept-Encoding request header allows the browser to indicate to the server that it can accept compressed content.

[6] http://www.w3.org/Protocols/rfc1341/5_Content-Transfer-Encoding.html.

4.1. Lossless compression

Five standard encoding algorithms comprise our set of lossless techniques: Delta, GZip, Huffman, LZW, and LZ4. These five algorithms were selected for their popularity and appropriateness for the data. First, they are commonly used in various compression utilities for multimedia and file compression. Second, these compression algorithms exploit statistical redundancy to represent data more concisely without losing information. This is especially appropriate for compressing cursor data because we are dealing with long sequences of numbers from a fixed set (integers only).

Delta compression refers to compressing data in the form of a sequence of differences between the original data and the follow-up changes performed to such data. A delta-compressed cursor point **c** at time t is

$$\Delta\mathbf{c}_t = \mathbf{g}_t - \mathbf{g}_{t-1} \quad \forall t > 0$$

where **g** is a d-dimensional point from the original cursor data.

Huffman compression (Huffman, 1951) is a form of statistical encoding, where a binary tree is built from data frequencies. Given an alphabet of n symbols $A = \{a_1, \cdots, a_n\}$ and a set of symbol weights (or probabilities) $W = \{w_1, \cdots, w_n\}$, the goal is to produce a prefix-free binary code (a set of codewords) with minimum expected codeword length; i.e., minimize $\bar{L} = \sum_{i=1}^{n} w_i l_i$, where l_i is a codeword of length i. As in other entropy encoding methods, the more frequent a symbol is encoded, the shorter its bit-sequence.

The remaining lossless encoding techniques (GZip, LZW, and LZ4) are based on the milestone algorithms LZ77 (Ziv & Lempel, 1977) and LZ78 (Ziv & Lempel, 1978). In these encoding algorithms, repeated strings are replaced by back-references linking to the previous location of that identical string. Concretely: *(1)* GZip (Deutsch, 1992) is based on the deflate algorithm, a combination of LZ77 and Huffman, using a sliding window during compression; *(2)* LZW (Welch, 1984) is an optimization of LZ78, in which data is encoded by means of explicit dictionary entries; *(3)* LZ4 is an improvement over LZP (Bloom, 1996), a variation of LZ77, using finite context Markov prediction. It is focused on compression and decompression speed, and provides a multi-pass scheme that performs specially well for log file compression.[7]

4.2. Lossy compression

It is possible to compress the cursor data further by using a series of techniques that reduce the number of data points such that every point **c** of the compressed cursor trail C is an element of the original cursor trail G. Therefore, a lossy-compressed cursor trail is a proper subset of the original cursor trail: $C \subset G$ and $C \neq G$.

We investigated five lossy compression techniques to be used in our evaluation, shown visually in Fig. 3. Two were standard resampling techniques: piecewise linear interpolation (RSL) (Meijering, 2002) and nonlinear interpolation (RSN) (Kuhn, Tomaschewski, & Ney, 1981a). One was a space quantization algorithm: distance thresholding (IDT) (Widdel, 1984). Two were time-based polling techniques, inspired by other researchers: sampling the cursor position after a fixed interval (time-based polling, TBP) (Guo & Agichtein, 2010b) and sampling after a pause longer than a fixed amount of time (pause-based polling, PBP) (Huang et al., 2011). The purpose of using these techniques is to simplify the original cursor trajectory, by removing redundancy and non-significant variations in the data. Further, to our knowledge, this choice of methods covers a wide spectrum of today's techniques used in cursor tracking systems, both in academia and industry.

RSL is the simplest reconstruction method, sampling the points from a trajectory uniformly; i.e., denoting $N = |G|$ and $M = |C| < N$, an RSL-interpolated cursor point satisfies

$$\mathbf{c}_m = \frac{(m+1)M}{(N+1)} \quad \forall m \in M.$$

In other words, every mth evenly spaced cursor coordinate is preserved from the original data, discarding the rest. This method is by far the most popular resampling solution, which dates back to the Babylonians (Meijering, 2002) and is used today in both commercial and open-source software (Freeman & Ambady, 2010).

RSN creates a resampled sequence by using piecewise linear interpolation on point distances accumulated along the original cursor trajectory. Formally, an RSN-interpolated point satisfies

$$\mathbf{c}_m = \mathbf{g}_{n-1} + (\mathbf{g}_n - \mathbf{g}_{n-1})\frac{(m-1)\lambda - L_{n-1}}{L_n - L_{n-1}}$$

where $\lambda = \frac{L_N}{M-1}$ and $L_n < (m-1)\lambda < L_{n-1}$, with $m \in M,\ n \in N$. In other words, RSN is an extension of RSL for point distances instead of raw point coordinates. RSN is a prominent preprocessing technique for speech and online handwriting recognition (Kuhn, Tomaschewski, & Ney, 1981b).

IDT is a popular method to reduce mouse cursor data (Arroyo et al., 2006; Mueller & Lockerd, 2001), that is, sampling coordinates when the cursor moves away from the previous point by a fixed number of ϵ pixels:

$$\mathbf{c}_n = \mathbf{g}_n \iff |\mathbf{g}_n - \mathbf{g}_{n-1}| > \epsilon \quad \forall n \in N.$$

[7] http://code.google.com/p/lz4/.

Fig. 3. An illustration of a cursor trail compressed with different lossy compression algorithms in a thick red line, drawn alongside the original cursor trail in gray. The compression ratio for each algorithm is set to 50%, removing half of the original cursor coordinates. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The idea behind IDT is that points too close to one another provide redundant information, because of its close proximity to the previously recorded point.

TBP is time-based polling, a sampling method that is an adoption of RSL in time instead of spatial dimensions, sampling time-equidistant points from the original cursor trajectory. In other words, the browser simply records the position of the cursor periodically, at fixed-time intervals, typically with short intervals; e.g., between 50 and 100 ms. This method is also very popular in the research literature (Guo & Agichtein, 2010b; Leiva & Vivó, 2013).

Similarly, PBP is a derivation of IDT but using time instead of spatial constraints. That is, only when the time difference between the current and previous cursor coordinates is above a given threshold, the current coordinate is preserved. Both TBP and PBP work thus as time quantization algorithms. In particular, PBP has proved to perform well on a large scale (Huang et al., 2011; Huang et al., 2012b), where cursor tracking should not affect the user's browsing experience.

We also tested a variation of the *k*-means algorithm adopted for data sequences (Leiva & Vidal, 2013) such as the cursor coordinates we were compressing. However, it performed an order of magnitude slower than the other algorithms, making it impractical, so we excluded this and other hill-climbing strategies for online data processing (e.g. Hu & Ratzlaff, 2004) in the formal evaluation.

5. Evaluation

We evaluated the previously discussed compression techniques (Section 4) over the datasets described in Section 3.1. In order to better compare all techniques, we compressed 3-dimensional cursor data trajectories, i.e., (x, y, t) tuples, by extracting mouse and touch move events from the interaction logs. We excluded other events reported by the web browser because they were not necessary for recreating the full cursor trail. All experiments were performed offline in an i686 dual core CPU @ 2.8 GHz with 4 GB of RAM to maintain consistent performance results. All algorithms were implemented in Python, and were tuned the same way. No caching mechanism was implemented in order to log *all* page visits.

5.1. Evaluation metrics

For each combination of compression algorithm and dataset, we measured data size reduction (compressed size, compression ratio) and performance (compression time). For lossy compression, we also computed the distance between the compressed trail and the original trail from which it was derived. This measures how well the original trail can be replicated via the lossy-compressed trail, for situations where reproducing the original trail is useful. We define *trail replication* as the per-pixel similarity measure defined by the following distance metric:

$$\tau = \int_0^T \frac{1}{|\mathbf{g}|} \sqrt{\sum_{i=1}^{d} [\mathbf{c}_i(t) - \mathbf{g}_i(t)]^2 \mathrm{d}t} \quad (1)$$

where $\mathbf{c}$ represents the vector of compressed cursor coordinates, and $\mathbf{g}$ represents the vector of the original cursor coordinates. Eq. (1) therefore computes the sum of distances between the coordinates, normalized by the number of coordinates. Given that $\mathbf{c}$ and $\mathbf{g}$ have different lengths, to compute (1) we iterate over $\mathbf{g}$ and find the closest point in time from $\mathbf{c}$.

While a compression algorithm may not be able to reproduce the entire trail, it may capture the important points in the trail which may be sufficient for many applications. We can thus use the points before a click as a proxy for an important point in the trail, since this information is available in our log data. Therefore, for a second distance measure, (1) was computed only for coordinates occurring just before a click. This measure represents how well the compression algorithms perform when the points on the entire cursor trail are not equally important. The implicit assumption made by this measurement is that points are more important just before an action occurs, as suggested in previous work Guo and Agichtein (2010b), Hauger et al. (2011), Huang et al. (2012a), so it emphasizes evaluating the distance between the compressed cursor trail and the actual trail at points preceding a click event. Cursor sessions without clicks were excluded from this particular metric.

5.2. Lossless compression results

Fig. 4 provides an overview of the sizes that would be needed to transmit cursor data using lossless compression techniques. Relative compression performance is reported in Fig. 5. The results show that Delta, Huffman, and LZ4 behaved similarly, reducing the data size by around 50% in both the LAB and LIVE datasets. Overall, Delta compression seemed to perform well if the values were small, since they could be represented with fewer bytes. However, GZip and LZW were able to achieve the most compression, reduced the data size by approximately 80% in both datasets.

In terms of compression time (Fig. 6), our experiment showed that LZ4 performed extremely faster than the other methods, with a payload as low as 0.5 ms on average for LAB data and 0.1 ms for LIVE data. Huffman and LZW encoding performed similarly, requiring at least twice the time consumed by GZip and Delta compression. GZip performed significantly better than Delta compression in the LIVE dataset.

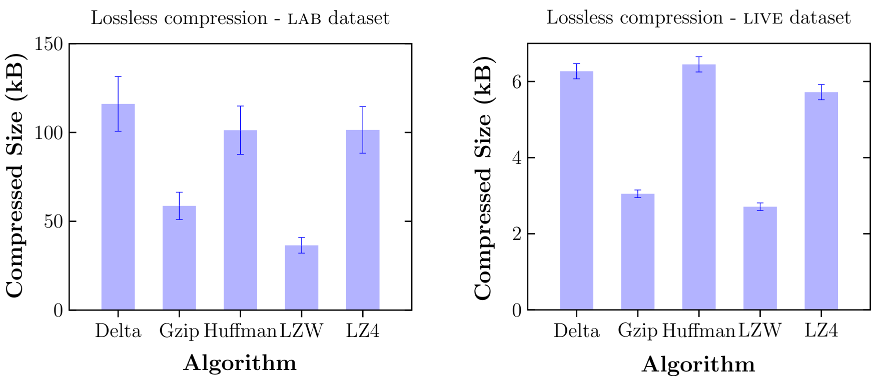

Fig. 4. Overall compressed data size for lossless algorithms, smaller is better. Error bars denote 95% confidence intervals.

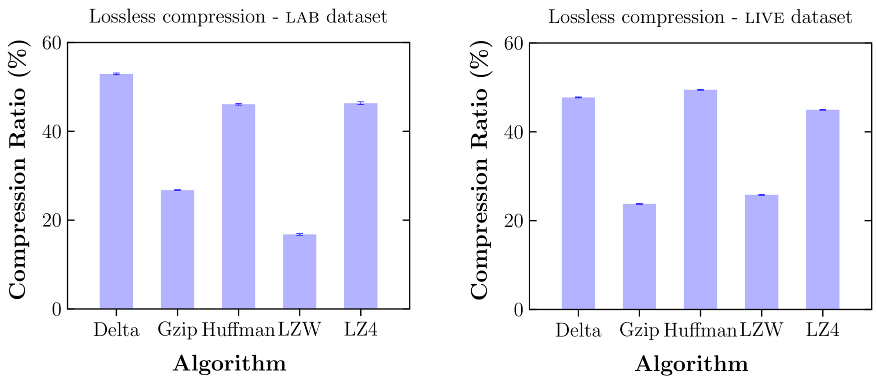

Fig. 5. Compression ratio for lossless algorithms, lower is better. Error bars denote 95% confidence intervals.

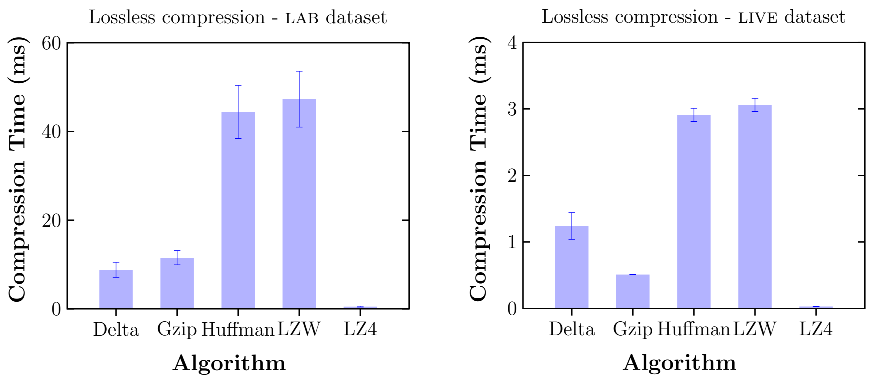

Fig. 6. Compression time for lossless algorithms, smaller is better. Error bars denote 95% confidence intervals.

Therefore, while LZW offered the greatest gains in compression, it was also relatively slow. GZip offered close to the compression ratio of LZW, while still achieving reasonable performance, and may be a suitable balance for compressing cursor activity.

5.3. Lossy compression results

For lossy compression algorithms, the compression ratio becomes a configurable independent variable in our experiments. We set the number of cursor points to be preserved as a percentage of the original data points, representing the option to choose an appropriate compression level for the application. We experimented with compression ratios ranging from 10% to 90%. A compression ratio of 10% means that a cursor trail is reduced to 10% of the original cursor trail length. Therefore, the lower the compression ratio the lower the number of compressed data points. A good lossy compression algorithm achieves good performance and replication results at a low compression ratio.

The lossy compression algorithms had different input parameters to lead to the resulting compression ratio. Both RSL and RSN take as input the number of points to be sampled, so no special setup was required. IDT takes as input a threshold distance between consecutive cursor coordinates (Widdel, 1984), for which we chose to be the distance that returned the desired number of points according to the selected compression ratio. Both polling techniques (TBP and PBP) do not have as input the number of data points. However, for TBP we set the poll interval proportional to the point percentages, which is also proportional to the compression ratio. For PBP, in principle it is not possible to create a similar dependency, as there is no direct way to relate a pause threshold to a desired number of points to be sampled. For that reason, we used a binary search strategy: a pause threshold was iteratively computed until reaching one that approximated the desired compression ratio. Because of this approximation, the computed pause threshold is suboptimal and thus may follow a non-linear trend; see Fig. 7.

As expected, the lossy compression techniques performed better as the compression ratio increased. Overall, it was observed that the data size reduction was linear with regard to the compression ratio (Fig. 7). As a consequence of this linear dependence, the higher compression ratio requires more time to compress the data, because more points are processed while sampling the original cursor trail. In this regard, polling techniques were found to be faster than IDT and RSN, and comparable to RSL for compression ratios below 50% (Fig. 8). It is worth noting that the slowest lossy compression algorithms are generally faster than the fastest lossless compression algorithms.

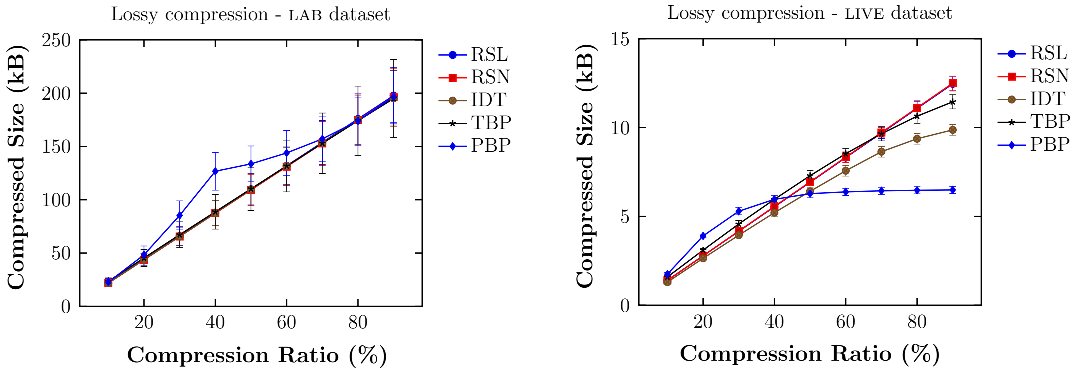

Fig. 7. Overall compressed data size for lossy algorithms (smaller is better), as a function of compression ratio (higher ratios mean less reduction of cursor data points). Error bars denote 95% confidence intervals.

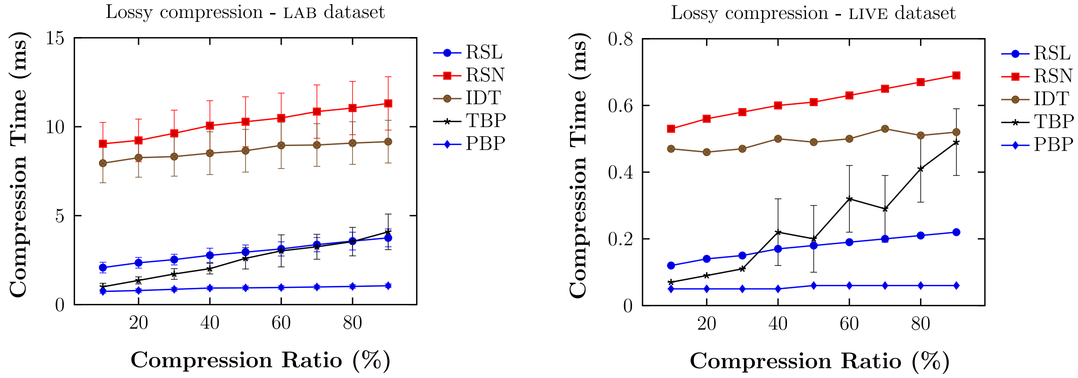

Fig. 8. Compression time for lossy algorithms (lower is better), as a function of compression ratio (higher ratios mean less reduction of cursor data points). Error bars denote 95% confidence intervals.

Fig. 9 shows how well the compressed cursor trails reproduce the original cursor trail. Polling techniques were worse at producing the original trail using the metric from Eq. (1). Their poorer performance compared to other lossy compression techniques may be explained by the fact that the timestamps of compressed points in TBP and PBP are usually assigned to coordinates that are distant from the original cursor position. This is still more pronounced for TBP, which appears to get worse as it includes more data. Therefore, while they are faster approaches, they may be less accurate for replicating the original cursor data. However, as observed, the performance for TBP and PBP will change for different polling intervals and pause thresholds, respectively.

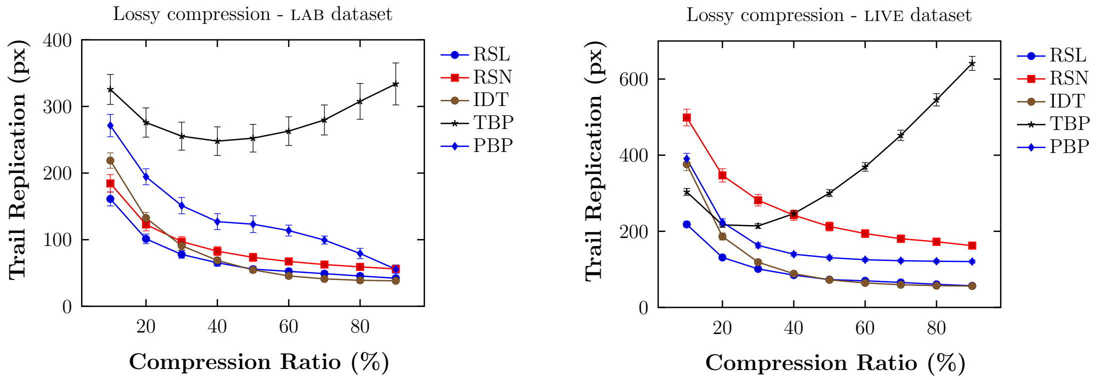

Fig. 9. Trail replication for lossy compression algorithms (lower is better), as a function of compression ratio (higher ratios mean less reduction of cursor data points). Error bars denote 95% confidence intervals.

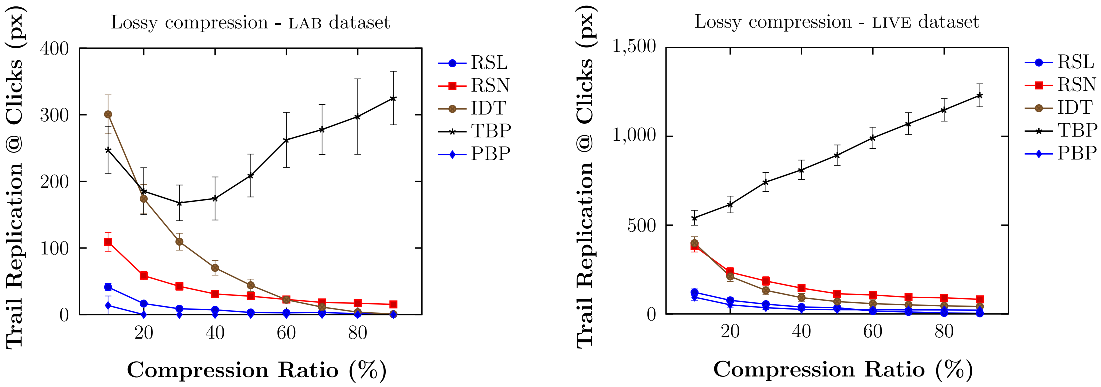

Fig. 10. Trail replication at clicks for lossy compression algorithms (lower is better), as a function of compression ratio (higher ratios mean less reduction of cursor data points). Error bars denote 95% confidence intervals.

The two resampling techniques (RSL, RSN) and space quantization (IDT) had an approximate distance of 50 px when the data was compressed by 60% or more in the LAB data. RSN had worse replication accuracy for the LIVE data, where RSL and IDT remained strong. From this, we can see that both RSL and IDT seem to be better approaches to preserving the original cursor trail. For the LIVE data, RSL and IDT outperformed the others for compression ratios above 80%. These results suggest that resampling techniques may be a convenient approach for replaying cursor trails, when reproducing the original cursor data accurately is important.

When we look at the distance between compressed coordinates and original coordinates only at points preceding a click (Fig. 10), all lossy compression techniques except TBP performed well for compression ratios above 50% for the LAB dataset. When the data was compressed substantially, IDT significantly diverged and both RSL and PBP retained their ability to reproduce the original cursor data during clicks for compression ratios below 10%. For this dataset, one trail was dropped from the analysis, since it did not have any click events. On the LIVE data, resampling techniques performed reasonably well for compression ratios above 30%, with distances below 100 px. RSL had the shortest average distance, and thus is the best compression algorithm for applications where reducing data size and reproducing the original coordinates at clicks are both important goals. For this dataset, 47% of the data (5006 interaction logs) did not contain click events and consequently were not considered in the analysis. Overall, these experiments shows that lossy compression techniques usually preserve important information for web analytics.

5.4. Lossy + lossless compression results

To further reduce the size of the cursor trails without additional loss in data quality, we investigated the combination of our lossy and lossless techniques. We devised a 2-pass procedure: a first pass of lossy compression, followed by a second pass of lossless compression. The original cursor data were lossy-compressed to 50% of the original size, since 50% has been shown to be a reasonable balance for all of our tested lossy compression techniques. Then, the lossy-compressed data were compressed further with each lossless compression technique, arriving at 25 combinations of compression algorithms in total. The results of these experiments are shown in Table 2 and Table 3.

As can be observed in the tables, in terms of overall compressed data size, using LZW as a second compression pass leads to the smallest sizes and consistently provides the higher savings, ranging between 50% and 100% of the lossy-compressed data size achieved in a first-pass; c.f. Fig. 7. This behavior was observed in both datasets, which is in line with the experiments conducted in Section 5.2.

However, LZW was also one of the slowest performing algorithms, beating only Huffman compression. In this regard, LZ4 was found to be much faster than the remainder of the tested lossy + lossless combinations. Here, this algorithm achieved the same effect as GZip in terms of compressed data size. Therefore, it can be considered a good candidate to re-compress lossy-compressed cursor data in a second pass. Nevertheless, all combinations took up around 1–2 ms to complete in the LIVE dataset, and therefore all can be considered fast enough for production-ready websites or web-based applications.

6. Discussion

Lossless compression is necessary when exact cursor trail reproduction is essential, such as if the user activity is being recorded for auditing financial transactions, e.g. did the user really intentionally click the "Transfer Money" button or was it by accident? Additionally, having the original data allows the website to capture and use data later even when the application is undetermined at the time of capture. But our results show that lossy compression produces greater space reduction and can take less time. For most applications such as usability testing or for improving the efficacy of a website, the data

Table 2
Combination of 2-pass (lossy + lossless) compression over the LAB dataset, both in terms of compressed data size and compression time. The best result in each case is highlighted in bold face. −ΔSize denotes the data size saving due to the second pass (lossless compression), higher is better. ΔTime denotes the time overhead incurred by the second pass compression, lower is better.

2-Pass compression	Size (kB)		−ΔSize (kB)		Time (ms)		ΔTime (ms)	
	M	SD	M	SD	M	SD	M	SD
RSL baseline	109.8	14.7	–	–	2.9	0.4	–	–
RSL + Delta	62.2	21.0	47.6	16.6	7.7	2.6	4.3	1.5
RSL + GZip	30.0	10.0	79.8	27.6	9.6	3.5	6.2	2.3
RSL + Huffman	51.2	17.5	58.6	20.1	29.2	9.8	25.7	8.7
RSL + LZW	**19.3**	5.8	**90.6**	31.8	24.5	8.6	21.1	7.4
RSL + LZ4	30.0	10.0	79.8	27.6	**3.7**	1.2	**0.3**	0.1
RSN baseline	109.1	14.7	–	–	10.2	1.4	–	–
RSN + Delta	58.1	20.0	51.1	17.5	15.4	5.4	4.3	1.4
RSN + GZip	22.4	8.5	86.8	29.1	15.9	5.9	4.8	2.0
RSN + Huffman	50.9	17.5	58.3	20.0	36.8	12.6	25.6	8.7
RSN + LZW	**18.7**	5.7	**90.5**	31.8	32.2	11.2	21.0	7.3
RSN + LZ4	22.4	8.5	86.8	29.1	**11.4**	4.0	**0.2**	0.1
IDT baseline	109.6	14.7	–	–	8.65	1.2	–	–
IDT + Delta	61.2	20.2	48.5	17.4	13.2	4.4	4.3	1.4
IDT + GZip	28.7	9.5	80.9	28.1	15.4	5.4	6.6	2.4
IDT + Huffman	51.1	17.6	58.5	20.1	34.6	11.8	25.7	8.7
IDT + LZW	**19.0**	5.7	**90.6**	31.9	30.0	10.3	21.1	7.3
IDT + LZ4	28.7	9.5	80.9	28.1	**9.2**	3.1	**0.3**	0.1
TBP baseline	110.2	20.3	–	–	2.6	0.6	–	–
TBP + Delta	100.0	46.7	10.2	6.8	7.2	4.1	4.5	3.2
TBP + GZip	37.5	17.2	72.8	34.6	8.9	4.4	6.2	3.3
TBP + Huffman	53.1	25.0	57.1	26.7	28.7	13.2	26.1	12.1
TBP + LZW	**22.2**	9.3	**88.0**	42.4	25.4	11.7	22.8	10.5
TBP + LZ4	37.5	17.2	72.8	34.6	**3.0**	1.3	**0.3**	0.2
PBP baseline	133.6	16.8	–	–	0.9	0.1	–	–
PBP + Delta	70.9	22.4	62.8	20.4	6.6	3.0	5.6	2.7
PBP + GZip	35.2	11.1	98.5	31.7	8.5	2.8	7.5	2.5
PBP + Huffman	62.3	20.0	71.4	22.8	32.2	10.3	31.2	10.0
PBP + LZW	**22.8**	6.6	**110.9**	36.2	27.3	9.1	26.4	8.8
PBP + LZ4	35.2	11.1	98.5	31.7	**1.3**	0.4	**0.3**	0.1

savings from compression outweigh the ability to keep every event. Within lossy compression algorithms, RSL is better for higher levels of compression when reducing the data and bandwidth is important, while IDT is better for situations where moderate compression is preferred. When performance is critical or the website is accessible to a broad range of devices, polling-based approaches such as PBP may be suitable. Furthermore, we have shown that is possible to reduce further the size of lossy-compressed cursor data performing a second pass of lossless compression, reaching most of the time significant savings both in terms of data size and compression time.

One opportunity for further optimization is to investigate more sophisticated approaches of lossless compression. For instance, Huffman compression approaches optimality as the number of symbols increases; however the potential space saving is then penalized with the size of the Huffman tree, which must be transmitted to the server in order to reconstruct (decode) the compressed data. Therefore, the size of a Huffman-compressed cursor trajectory plus its corresponding decoding tree is typically higher than the size of the data encoded with GZip or LZW. Therefore, some hybrid strategy would be worth trying in a future; e.g., using Huffman to compress cursor data and LZW to compress Huffman trees.

Performance can affect the user experience of using the website or application, since running additional scripts on a web page may cause a short delay. On desktop applications, modern computers can generally handle small-scale compression without any noticeable effect. However, on web browsers and especially on mobile devices, the time spent compressing the data may outweigh the benefits from reducing the data sent across the network. Mark8t, an e-marketing blog noted in a comparison of cursor tracking services[8] that "In the end, all of these tools will give you the insight you need..." but the now defunct Userfly "causes sites speed performance issues." Algorithms that depend on intense compression may cause the web page to lag during user interaction, which concerns website developers. One potential solution is to perform the compression when the user is idle, hiding any lag from the user experience.

Performance incurred by client-side cursor data logging can be further improved by following these recommendations: (1) make use of asynchronous XMLHttpRequest. Asynchronous HTTP requests are non-blocking and therefore allow for regular interaction with the page. (2) Issue a small number of HTTP requests. HTTP requests introduce an important load

[8] http://blog.mark8t.com/2011/09/10/.

Table 3
Combination of 2-pass (lossy + lossless) compression over the LIVE dataset, both in terms of compressed data size and compression time. The best result in each case is highlighted in bold face. −ΔSize denotes the data size saving due to the second pass (lossless compression), higher is better. ΔTime denotes the time overhead incurred by the second pass compression, lower is better.

2-Pass compression	Size (kB)		−ΔSize (kB)		Time (ms)		ΔTime (ms)	
	M	SD	M	SD	M	SD	M	SD
RSL baseline	6.9	0.2	–	–	0.1	<0.1	–	–
RSL + Delta	3.4	5.8	3.6	6.8	0.6	4.6	0.4	4.5
RSL + GZip	1.7	2.9	5.2	9.7	0.5	1.0	0.2	0.6
RSL + Huffman	3.3	5.9	3.6	6.7	1.9	3.3	1.7	2.9
RSL + LZW	**1.5**	2.2	**5.4**	10.5	1.7	2.8	1.4	2.4
RSL + LZ4	1.7	2.9	5.2	9.7	**0.2**	0.4	**0.0**	0.0
RSN baseline	6.9	0.2	–	–	0.6	<0.1	–	–
RSN + Delta	3.1	5.5	3.8	7.2	1.1	5.9	0.4	5.4
RSN + GZip	**0.9**	1.6	**6.1**	11.1	0.8	1.5	0.1	0.3
RSN + Huffman	3.3	5.9	3.6	6.8	2.3	4.1	1.7	2.9
RSN + LZW	1.3	1.9	5.6	10.8	2.1	3.6	1.4	2.3
RSN + LZ4	**0.9**	1.6	**6.1**	11.1	**0.7**	1.2	**0.0**	0.0
IDT baseline	6.4	0.2	–	–	0.4	<0.1	–	–
IDT + Delta	3.1	5.1	3.4	6.2	0.9	5.0	0.4	4.7
IDT + GZip	1.6	2.7	4.8	8.6	0.8	1.5	0.2	0.5
IDT + Huffman	3.1	5.2	3.3	6.0	2.1	3.5	1.6	2.6
IDT + LZW	**1.4**	2.0	**5.0**	9.3	1.9	3.1	1.4	2.2
IDT + LZ4	1.6	2.7	4.8	8.6	**0.5**	1.0	**0.0**	0.0
TBP baseline	7.2	0.3	–	–	0.2	0.1	–	–
TBP + Delta	5.8	10.6	1.5	3.2	0.6	6.1	0.4	4.9
TBP + GZip	2.3	4.1	5.0	9.6	0.5	3.8	0.3	0.6
TBP + Huffman	3.6	6.6	3.7	7.1	2.0	5.3	1.8	3.2
TBP + LZW	**1.8**	2.7	**5.5**	11.0	1.8	5.0	1.6	2.7
TBP + LZ4	2.3	4.1	5.0	9.6	**0.3**	3.6	**0.0**	0.0
PBP baseline	6.2	0.2	–	–	<0.1	<0.1	–	–
PBP + Delta	3.0	5.0	3.2	5.7	0.5	6.3	0.5	6.3
PBP + GZip	1.4	2.4	4.8	8.4	0.3	0.5	0.2	0.5
PBP + Huffman	3.0	5.0	3.3	5.7	1.6	2.6	1.5	2.5
PBP + LZW	**1.4**	1.9	**4.9**	8.8	1.4	2.1	1.3	2.1
PBP + LZ4	1.4	2.4	4.8	8.4	**0.1**	0.1	**0.0**	0.0

increase on the server. Not only is there a header block for both the request and the response, but there is also a three-way handshake due to the TCP/IP layer, which underlines the transmission protocol. (3) Enable persistent connections, e.g. using the Keep-Alive header. This is expected to help reducing both latency and network traffic, as well as saving CPU and memory usage on the server side.

What are the implications of these results for overall performance in collecting cursor interaction data? In current computing systems, storage time is minimal, and the main overhead occurs from compression time (CPU-limited) and data transmission time (network-limited). Informally, we may estimate the total processing time as the sum of compression time and data transmission time. Compression time was typically under 1 ms under the LIVE condition, while transmission time on a home broadband connection (3 Mbit up, 20 ms ping), comes out to 33 ms for 5 kB of data. Thus network transmission time is probably an order of magnitude larger than compression time, and for in-home environments it makes sense to choose a better compression method over a faster performing one (see Fig. 11 as an example). However, in situations where compression time is substantially higher such as older smartphones or smaller devices, more performance-efficient compression algorithms can be considered.

On a related note, we should mention that there are different approaches to encode the data prior compression, some of which could possibly make a difference. For instance, mouse cursor data can be stored in JSON, XML, or simply in a raw JavaScript array. While we did not have the chance to test the performance of these encoding structures, we suspect the raw array format is the most appropriate. However, the other formats provide redundancy in the syntax regardless of the mouse coordinates values and thus any compression over these formats would have a good head start.

One avenue we believe can be investigated further is the human physiology of targeting. When moving the cursor to perform an action, humans start with an initial impulse, also called the ballistic phase, followed by corrective phases (Woodworth, 1899). With this in mind, it may be possible to treat cursor movements that aim for a target differently. The pause-based polling method is likely to capture the points between phases, which are likely to be around 135 ms (Carlton, 1981) because of the time it takes to see the current state and adjust the movement (Fig. 12). But perhaps the cursor movements can be converted to direction and acceleration values for each submovement phase instead. This approach would greatly compress the data by reducing the cursor movements to the forces applied to the mouse by the user. It may also be possible to represent the direction with more bits of information than the amplitude of movement, because humans are better at aiming in the right direction than they are at applying the correct amount of force (Gordon, Ghilardi, & Ghez, 1994).

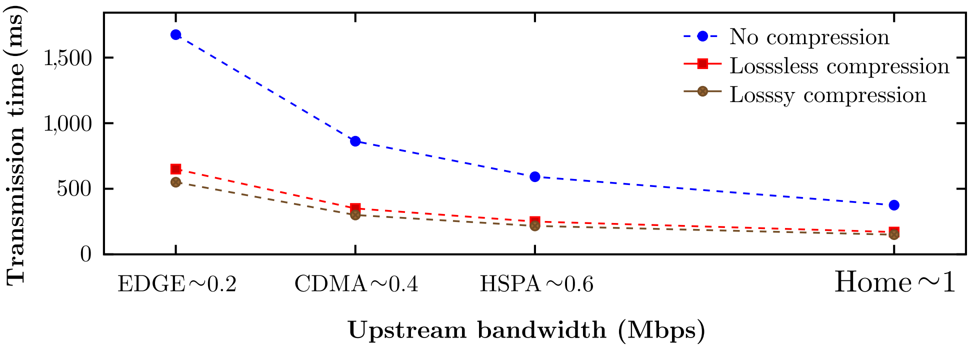

Fig. 11. Estimated tradeoff between transmission time and bandwidth for common broadband connections.

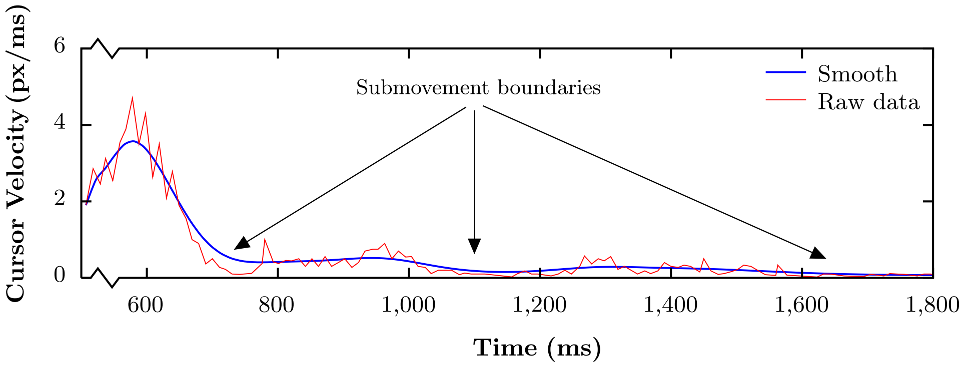

Fig. 12. Plot of a single cursor movement towards a target, illustrating the large initial ballistic movement followed by smaller corrective movements. The short pauses between movements are reasonable places to capture a cursor position, as is done in pause-based polling (PBP).

From a user perspective, there may be privacy concerns on websites that deploy cursor analytics. Most users are not aware that websites naturally have the capability to record mouse cursor and scrolling activity. Cursor tracking has primarily positive benefits to the user, as it can lead to websites improving their usability and presenting more relevant content. One option is to only record the areas of interest that the cursor hovers over. However, there is still a possibility that cursor activity can reveal users' intentions that are meant to be private. For example, a user may search for information about bad hiring practices, and the part of the page they focus on may reveal that they are unsatisfied with their employer's hiring practices. In these cases, websites should ask for consent from the user before collecting this data, and maintain a strict policy for protecting the gathered data.

Beyond cursor trails, our compression methods may be applicable to other types of trails. For example, trails of eye movements on a web page during an eye-tracking lab study may be compressed if they need to be stored efficiently. On touch-enabled devices, tracking which parts of the page people have tapped, swiped, or pinched may be interesting, especially since many touch devices have weaker processing and bandwidth capabilities. Huang and Diriye argue that tracking the viewport area in the browser window is especially important on devices with smaller screens, since the user has to zoom and pan before reading the content (Huang & Diriye, 2012). These interactions can reveal which regions of the page held the user's attention, and for how long. Related to this, Leiva showed the value of leveraging touch and mouse tracking information to incrementally introduce subtle changes to the CSS of the page (Leiva, 2011, 2012), which enables a novel paradigm of self-adaptive websites. Applications like these put forward the need to research additional cursor compression strategies.

We believe that sharing evaluations of compressing on cursor data, along with the implementation of the compression algorithms may benefit researchers and commercial analytics services. The potential reductions in bandwidth and improvements in performance may affect many users who visit websites that record cursor data. Our implementations of the lossy and lossless compression algorithms described in the paper are publicly available at http://hci.cs.brown.edu/mousetrap/, as well as the anonymized LIVE dataset for other researchers to replicate our experimental results.

7. Conclusion

Tracking mouse cursor activity benefits websites that seek richer page-level interaction data from their audience. These analytics data can be applied to improve the usability and efficacy of a website. Currently, commercial analytics services and web researchers capture cursor movements using different approaches. Applying compression can optimize for (1) reducing data size and therefore bandwidth between the client and the server, (2) having minimal effect on client-side performance, or (3) reproducing the original data the most accurately.

We investigated 10 compression algorithms, half of them lossless and the other half lossy. Lossless compression algorithms are capable of reducing the data size while also reproducing the original cursor activity; we found that among lossless methods, LZW performed well. However, lossy compression can offer greater gains in compression as well as improved performance, at the expense of being able to exactly replicate the original trail. For the same compression levels, the naïve method of time-based polling replicated the original trail worse than recently developed methods. Piecewise linear interpolation and distance-thresholding could reproduce the original trail more accurately at the same levels of compression. Pause-based polling, which has been previously deployed on a large scale, remains superior in terms of client-side performance; this compression method is especially suitable for wide distribution when some users may be using slower computers. Furthermore, we have shown that significant savings can be achieved if lossy-compressed mouse cursor data are lossless-compressed in a second pass, with the same data quality as cursor activity compressed with a single lossy method.

For the different goals or applications, there is a suitable compression algorithm available to websites who wish to deploy cursor tracking. We believe our findings improve upon the state-of-the-art methods for mouse cursor tracking, making cursor tracking a viable tool for better understanding the user and improving usability on websites.

Acknowledgments

We thank Henry Feild for sharing his eye-tracking study dataset. We also thank Susan Dumais, Qi Guo, Anna Kazeykina, and Jacob Wobbrock for fruitful discussions. We thank as well the anonymous IPM reviewers for their valuable suggestions and comments on our manuscript. This work has been partially supported by a Google Research Award, the Facebook Fellowship, and the 7th Framework Program of the European Union through CasMaCat (FP7 ICT-2011.4.2, Grant 287576) and tranScriptorium (FP7 ICT-2011.8.2, Grant 600707) research projects.

References

Arroyo, E., Selker, T., & Wei, W. (2006). Usability tool for analysis of web designs using mouse tracks. In *Proceedings of CHI extended abstracts* (pp. 484–489).
Atterer, R., Wnuk, M., & Schmidt, A. (2006). Knowing the user's every move: User activity tracking for website usability evaluation and implicit interaction. In *Proceedings of WWW* (pp. 203–212).
Bloom, C. (1996). LZP: A new data compression algorithm. In *Proceedings of DCC* (pp. 425–425).
Buscher, G., White, R. W., Dumais, S., & Huang, J. (2012). Large-scale analysis of individual and task differences in search result page examination strategies. In *Proceedings of WSDM* (pp. 373–382).
Carlton, L. G. (1981). Processing visual feedback information for movement control. *Journal of Experimental Psychology: Human Perception and Performance, 7*, 1019–1030.
Chen, M. C., Anderson, J. R., & Sohn, M. H. (2001). What can a mouse cursor tell us more?: Correlation of eye/mouse movements on web browsing. In *Proceedings of CHI extended abstracts* (pp. 281–282).
Cooke, L. (2006). Is the mouse a poor man's eye tracker? In *Proceedings of STC* (pp. 252–255).
Deutsch, L. P. (1992). Gzip compressed data format specification. Available at <ftp://ftp.uu.net/pub/archiving/zip/doc/>.
Diaz, F., White, R. W., Buscher, G., & Liebling, D. (2013). Robust models of mouse movement on dynamic web search results pages. In *Proceedings of CIKM* (pp. 1451–1460).
Diriye, A., White, R. W., Buscher, G., & Dumais, S. (2012). Leaving so soon?: Understanding and predicting web search abandonment rationales. In *Proceedings of CIKM* (pp. 1025–1034).
Feild, H. A., Allan, J., & Jones, R. (2010). Predicting searcher frustration. In *Proceedings of SIGIR* (pp. 34–41).
Freeman, J. B., & Ambady, N. (2010). Mousetracker: Software for studying real-time mental processing using a computer mouse-tracking method. *Behavior Research Methods, 42*, 226–241.
Gordon, J., Ghilardi, M. F., & Ghez, C. (1994). Accuracy of planar reaching movements. i. Independence of direction and extent variability. *Experimental Brain Research, 99*, 97–111.
Guo, Q., & Agichtein, E. (2008). Exploring mouse movements for inferring query intent. In *Proceedings of SIGIR* (pp. 707–708).
Guo, Q., & Agichtein, E. (2010a). Ready to buy or just browsing?: Detecting web searcher goals from interaction data. In *Proceedings of SIGIR* (pp. 130–137).
Guo, Q., & Agichtein, E. (2010b). Towards predicting web searcher gaze position from mouse movements. In *Proceedings of CHI extended abstracts* (pp. 3601–3606).
Guo, Q., & Agichtein, E. (2012). Beyond dwell time: Estimating document relevance from cursor movements and other post-click searcher behaviour. In *Proceedings of WWW* (pp. 569–578).
Hauger, D., Paramythis, A., & Weibelzahl, S. (2011). Using browser interaction data to determine page reading behaviour. In *Proceedings of UMAP* (pp. 147–158).
Hauger, D., & Van Velsen, L. (2009). Analyzing client-side interactions to determine reading behaviour. In *Proceedings of ABIS* (pp. 11–16).
Hu, J., & Ratzlaff, E. (2004). Probability table compression using distributional clustering for scanning n-tuple classifiers. In *Proceedings of ICPR* (pp. 533–536).
Huang, J., & Diriye, A. (2012). Web user interaction mining from touch-enabled mobile devices. In *HCIR workshop*.
Huang, J., White, R., & Buscher, G. (2012a). User see, user point: Gaze and cursor alignment in web search. In *Proceedings of CHI* (pp. 1341–1350).
Huang, J., White, R. W., Buscher, G., & Wang, K. (2012b). Improving searcher models using mouse cursor activity. In *Proceedings of SIGIR* (pp. 195–204).
Huang, J., White, R. W., & Dumais, S. (2011). No clicks, no problem: Using cursor movements to understand and improve search. In *Proceedings of CHI* (pp. 1225–1234).
Huffman, D. A. (1951). A method for the construction of minimum redundancy codes. In *Proceedings of IRE* (pp. 1098–1101).
Jain, A., & Glasgow, J. (2012). Use compression to make the web faster. Available at <http://developers.google.com/speed/articles/use-compression>.
Kuhn, M. H., Tomaschewski, H., & Ney, H. (1981a). Fast nonlinear time alignment for isolated word recognition. In *Proceedings of ICASSP* (pp. 736–740).
Kuhn, M. H., Tomaschewski, H., & Ney, H. (1981b). Fast nonlinear time alignment for isolated word recognition. In *Proceedings of the IEEE international conference on acoustics, speech, and signal processing (ICASSP)* (pp. 736–740).
Leiva, L. A. (2011). Restyling website design via touch-based interactions. In *Proceedings of MobileHCI* (pp. 599–604).
Leiva, L. A. (2012). ACE: An adaptive CSS engine for web pages and web-based applications. In *Proceedings of WWW developer track*.
Leiva, L. A., & Vidal, E. (2010). Assessing user's interactions for clustering web documents: A pragmatic approach. In *Proceedings of HT* (pp. 277–278).
Leiva, L. A., & Vidal, E. (2013). Warped k-means: An algorithm to cluster sequentially-distributed data. *Information Sciences*, 237.
Leiva, L. A., & Vivó, R. (2013). Web browsing behavior analysis and interactive hypervideo. *ACM Transactions on the Web, 7*.

Liu, C. C., & Chung, C. W. (2007). Detecting mouse movement with repeated visit patterns for retrieving noticed knowledge components on web pages. *IEICE – Transactions on Information and Systems*, 1687–1696.

Meijering, E. (2002). A chronology of interpolation: From ancient astronomy to modern signal and image processing. *Proceedings of the IEEE, 90*, 319–342.

Mueller, F., & Lockerd, A. (2001). Cheese: Tracking mouse movement activity on websites, a tool for user modelling. In *Proceedings of CHI extended abstracts* (pp. 279–280).

Rodden, K., & Fu, X. (2007). Exploring how mouse movements relate to eye movements on web search results pages. In *SIGIR workshop on web information seeking and interaction* (pp. 29–32).

Rodden, K., Fu, X., Aula, A., & Spiro, I. (2008). Eye–mouse coordination patterns on web search results pages. In *Proceedings of CHI extended abstracts* (pp. 2997–3002).

Speicher, M., Both, A., & Gaedke, M. (2013). Tellmyrelevance! predicting the relevance of web search results from cursor interactions. In *Proceedings of CIKM* (pp. 1281–1290).

Tukey, J. W. (1977). *Exploratory data analysis*. Addison-Wesley Publishing Co.

Welch, T. (1984). A technique for high-performance data compression. *IEEE Computer, 17*, 8–19.

Widdel, H. (1984). Operational problems in analysing eye movements. In A. G. Gale & F. Johnson (Eds.), *Theoretical and applied aspects of eye movement research* (pp. 21–29). New York: Elsevier Science Ltd.

Woodworth, R. (1899). Accuracy of voluntary movement. *Psychological review, III*(3), 1–114.

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory, 23*, 337–343.

Ziv, J., & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory, 24*, 530–536.