Archival Crawlers and JavaScript: Discover More Stuff but Crawl More Slowly

Justin F. Brunelle[*]
The MITRE Corporation
jbrunelle@mitre.org

Michele C. Weigle
Old Dominion University,
Department of Computer Science
mweigle@cs.odu.edu

Michael L. Nelson
Old Dominion University,
Department of Computer Science
mln@cs.odu.edu

ABSTRACT

The web is today's primary publication medium, making web archiving an important activity for historical and analytical purposes. Web pages are increasingly interactive, resulting in pages that are correspondingly difficult to archive. JavaScript enables interactions that can potentially change the client-side state of a representation. We refer to representations that load embedded resources via JavaScript as *deferred representations*. It is difficult to discover and crawl all of the resources in deferred representations and the result of archiving deferred representations is archived web pages that are either incomplete or erroneously load embedded resources from the live web. We propose a method of discovering and archiving deferred representations and their *descendants* (representation states) that are only reachable through client-side events. Our approach identified an average of 38.5 descendants per seed URI crawled, 70.9% of which are reached through an onclick event. This approach also added 15.6 times more embedded resources than Heritrix to the crawl frontier, but at a crawl rate that was 38.9 times slower than simply using Heritrix. If our method was applied to the July 2015 Common Crawl dataset, a web-scale archival crawler will discover an additional 7.17 PB (5.12 times more) of information per year. This illustrates the significant increase in resources necessary for more thorough archival crawls.

CCS CONCEPTS

• **Information systems** →**Digital libraries and archives;**

KEYWORDS

Web Archiving; Digital Preservation; Memento; Web Crawling

[*]Work performed in part during graduate work at Old Dominion University, Department of Computer Science.

ACM Reference format:
Justin F. Brunelle, Michele C. Weigle, and Michael L. Nelson. 2017. Archival Crawlers and JavaScript: Discover More Stuff but Crawl More Slowly. In *Proceedings of ACM Joint Conference on Digital Libraries, Toronto, Ontario, Canada, June 2017 (JCDL2017),* 10 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As the web grows as the primary medium for publication, communication, and other services, so grows the importance of preserving the web (as evidenced by recent articles in *The New Yorker* [28] and *The Atlantic* [27]). Web resources are ephemeral, existing in the perpetual *now*; important historical events frequently disappear from the web without being preserved or recorded. We may miss pages because we are not aware they should be saved or because the pages themselves are hard to archive.

On July 17, 2015, Ukrainian separatists announced via social media[1], with video evidence, that they shot down a military cargo plane in Ukrainian airspace. Evidence suggests that the downed plane was the commercial Malaysian Airlines Flight 17 (MH17). The Ukrainian separatists removed from social media their claim of shooting down what we now know was a non-military passenger plane. The Internet Archive [34], using the Heritrix web crawler [33, 41], was crawling and archiving the social media site twice daily and archived the claimed credit for downing the aircraft; this is evidence that Ukrainian separatists shot down MH17 [7]. This (among others [1]) is an example of the importance of high-fidelity web archiving to record history and establish evidence of information published on the web.

However, not all historical events are archived as fortuitously as the MH17 example. In an attempt to limit online piracy and theft of intellectual property, the U.S. Government proposed the widely unpopular Stop Online Piracy Act (SOPA) [37]. While the attempted passing of SOPA may be a mere footnote in history, the overwhelming protest in response is significant. On January 18, 2012, many prominent websites organized a world-wide blackout in protest of SOPA. Wikipedia blacked out their site by using JavaScript to load a "splash page" that prevented access to Wikipedia's content[2].

[1]VKonkakte, https://vk.com/strelkov_info, is a Russian social media site.
[2]Extended examples and figures are omitted in this paper but can be found in prior works [8, 18].

The Internet Archive, using Heritrix, archived the Wikipedia site during the protest. However, the archived January 18, 2012 page, as replayed in the Wayback Machine [44], does not include the splash page. Because archival crawlers such as Heritrix are not able to execute JavaScript, they neither discovered nor archived the splash page. Wikipedia's protest as it appeared on January 18, 2012 has been lost from the archives and without human efforts, would be potentially lost from human history.

The SOPA protest, like MH17, is an example of an important historical event. Unlike the MH17 example (which establishes our need to archive with high fidelity), the SOPA example is not well represented in the archives. This is a result of browsers implementing (and content authors adopting) client-side technologies such as JavaScript faster than crawlers can adapt to handle the new technologies. This leads to a difference between the web that crawlers can discover and the web that human users experience – a challenge impacting the archives as well as other crawlers (e.g., those used by search engines). Over time, live web resources have been more heavily leveraging JavaScript to load embedded resources [14]. Because JavaScript-dependent representations are not accessible to archival crawlers, their representations are not fully archived. Heritrix does not execute any client-side scripts or use headless or headful browsing technologies. Even though it does not execute JavaScript, Heritrix v. 3.1.4 does peek into the embedded JavaScript code to extract links where possible [23]. In contrast, Google's search engine crawlers execute JavaScript to discover new URIs to crawl but does not perform page interactions [9][3].

When the representation is replayed from the archive, the JavaScript will execute and may issue Ajax requests for a resource that is on the live web, which leads to one of two possible outcomes: the live web "leaking" into the archive leading to an incorrect representation [15], or missing embedded resources (i.e., returns a 400 or 500 class HTTP response) in the archive leading to an incomplete representation, both of which result in reduced archival quality [13]. When an archived deferred representation loads embedded resources from the live web via leakage, it is a *zombie* resource, leaving the representation incorrect, or *prima facie violative* [3].

We define *deferred representations* as representations of resources that rely on JavaScript and other client-side technologies to initiate requests for embedded resources after the initial page load [11]. We use the term *deferred* because the representation is not fully realized and constructed until *after* the JavaScript code is executed on the client. Note that the mere presence of JavaScript does not indicate that a representation will be deferred. A deferred representation may be interactive, but its reliance on Ajax and JavaScript to initiate HTTP requests for post-load resources makes the representation deferred.

[3]Search engine crawlers are not incentivized to index embedded resources such as CSS or images while archival crawlers are expected to discover and index these elements.

2 CONTRIBUTIONS

In this paper, we define a representation constructed as a result of user interaction or other client-side event without a subsequent request for the resource's URI as a *descendant*[4] (i.e., a member of the client-side event tree below the root). Client-side events may also trigger a request for additional resources to be included in the representation, which leads to deferred representations.

We explore the number and characteristics of descendants as they pertain to web archiving, and explore the cost-benefit trade-off of actively crawling and archiving descendants en route to a higher quality, more complete archive. Dincturk et al. [17] constructed a model for crawling Rich Internet Applications (RIAs) by discovering all possible descendants and identifying the simplest possible state machine to represent the states. We explore the archival implementation of their Hypercube model by discovering client-side states and their embedded resources to understand the impact that deferred representations have on the archives.

We evaluate the performance impacts of crawling descendants (i.e., crawl time, depth, and breadth) against the improved coverage of the crawler (i.e., frontier size) along with the presence of embedded resources unique to descendants in the Internet Archive, using Heritrix as our web crawler. We show that there are two levels in the interaction trees of our URI-Rs. Crawling the descendants leads to 15.6 times more embedded resources (70.9% of which are available via onclick events) which are largely unarchived (92% in s_1 and 96% in s_2).

Throughout this paper we use Memento Framework terminology. Memento [45] is a framework that standardizes Web archive access and terminology. Original (or live web) resources are identified by URI-R, and archived versions of URI-Rs are called *mementos* and are identified by URI-M.

3 RELATED WORK

Banos et al. [5] created an algorithm to evaluate archival success based on adherence to standards for the purpose of assigning an archivability score to a URI-R. In our previous work [24], we studied the impact of accessibility standards on archivability and memento completeness. We also measured the correlation between the adoption of JavaScript and the number of missing embedded resources in the archives [14].

In previous work [12], we assigned a quantitative metric to a previously qualitative measurement of memento quality and measured a reduction in memento quality caused by JavaScript. Ben Saad and Gançarski [6] performed a similar study regarding the importance of changes on a page. Gray and Martin [19] created a framework for high quality mementos and assessed their quality by measuring the missing embedded resources.

[4]The Dincturk Hypercube model [17] refers to these as AJAX states or client-side states.

Browsertrix [25] and WebRecorder.io [26] are page-at-a-time archival tools for deferred representations and descendants, but they require human interaction and are not suitable for automated web archiving. Archive.is [4] handles deferred representations well, but is a page-at-a-time archival tool and strips out embedded JavaScript from the memento. Stripping the embedded JavaScript leads to potentially reduced functionality in the memento and an inability to perform a post-mortem analysis of a page's intended behavior using the memento.

While the notion of using headless browsing tools has been previously demonstrated by Archive-It's use of Umbra to crawl a human curated set of pre-defined URI-Rs [38] and Brozzler's headful or headless crawling [22], our work focuses on measuring the extent of the archival challenges posed by JavaScript and the impact of using such headless crawling tools on mementos. We proposed a two-tiered crawling approach for autonomously archiving deferred representations that uses Heritrix and PhantomJS [16]. We measured the performance impact of incorporating a headless browsing utility in an archival workflow. Our work demonstrates that PhantomJS [36] and its headless browsing approach can be used in conjunction with Heritrix to grow Heritrix's crawl frontier by 1.75 times and better archive deferred representations, but crawls 12.15 times slower than Heritrix alone. We build on this effort by enhancing the PhantomJS branch of the archival workflow to learn and execute interactions on the client similar to the Hypercube model.

Several efforts have studied client-side state. Mesbah et al. performed several experiments regarding crawling and indexing representations of web pages that rely on JavaScript [30, 32] focusing mainly on search engine indexing and automatic testing [31]. Singer et al. developed a method for predicting how users interact with pages to navigate within and between web resources [42]. Rosenthal spoke about the difficulty of archiving representations reliant on JavaScript [35, 39]. Rosenthal et al. extended their LOCKSS work to include a mechanism for handling Ajax [40]. Using CRAWLJAX and Selenium to click on DOM elements with onclick events attached and monitor the HTTP traffic, they capture the Ajax-specific resources.

In this work, we build on these past investigations to understand the multiple states that can be discovered on the client by mapping interaction trees and the additional resources required to build the descendants.

4 DESCENDANT MODEL

Dincturk et al. [17] present a model for crawling RIAs by constructing a graph of descendants (they refer to these as "AJAX states" within the Hypercube model; we use a tree structure and therefore refer to these as *descendants*). A RIA is a resource with descendants and potentially a deferred representation. The work by Dincturk et al. focuses on Ajax requests for additional resources initiated by client-side events which leads to deferred representations with descendants. Their work, which serves as the mathematical foundation for our work, identifies the challenges with crawling Ajax-based representations and uses a *hypercube strategy* to efficiently identify and navigate all client-side states of a deferred representation. Their model defines a client-side state as a state reachable from a URL through client-side events and is uniquely identified by the state's DOM. That is, two states are identified as equivalent if their DOM (e.g., HTML) is directly equivalent.

The hypercube model is defined by the finite state machine (FSM) $M = (S, s_0, \Sigma, \delta)$ (Equation 1), where

- S is the finite set of client states
- $s_0 \in S$ is the initial state reached by dereferencing the URI-R and executing the initial on-load events
- $e \in \Sigma$ defines the client-side event e as a member of the set of all events Σ
- $\delta : S\text{x}\Sigma \rightarrow S$ is the transition function in which a client-side event is executed and leads to a new state

$$\begin{aligned} &s_i, s_j \in S \\ &\delta(s_i, e) = s_j \\ &e = \text{client-side event} \\ &j = i + 1 \end{aligned} \tag{1}$$

Dincturk et al. define a graph $G = (V, E)$ in which V is the set of vertices $v_i \in V$ where v_i represents a descendant s_i. Edges represent the transitions, or events e such that $(v_i, v_j; e) \in E$ IFF $\delta(s_i, e) = s_j$. A path P is a series of edges that constitute a series of transitions from s_0 to s_i via $e_{i...j}$. In effect, P is a series of descendants derived from s_0 with one descendant at each level of the tree.

We make use of the FSM, mathematical model, and the notion of establishing a graph of state transitions in an RIA presented by Dincturk et al. However, we adapt portions to apply to web archiving rather than to state identification and navigation. Because our application of this FSM is web archiving, our goal is to identify all of the embedded resources required by the representation to build any descendant as a result of user interactions or client-side events, archive them, and be able to replay them when a user interacts with the memento. Our model differs in two aspects: the first difference is in our calculation of state equivalency (see Section 5; we consider states equivalent only if they require the same set of embedded resources) and the second difference is our notation for states reached after a series of events, described further in this section. In short, we use a tree representation rather than a hypercube to represent the notional model of our state transitions; our intent is to discover all possible states reachable from all possible events to ensure we discover all possible embedded resources. The need for the exhaustive breadth and depth of the interactions lends itself to a tree rather than the hypercube which is designed to eliminate identical states reached by different events.

The representation returned by simply dereferencing a URI-R is defined as $URI\text{-}R_{s_0}$. Subsequent descendants $URI\text{-}R_{s_i}$ and $URI\text{-}R_{s_j}$ are derived from $URI\text{-}R_{s_0}$ through a series of events $e_{i...j} \in \Sigma$. We define a descendant $URI\text{-}R_{s_i}$ as a client-side state originating at $URI\text{-}R_{s_0}$ as transitioning via events e such that $\delta(s_0, e) = s_1$.

Since our model focuses on web archiving rather than testing and because executing JavaScript by a crawler is slow, the effort expended by a crawler to discover states and embedded resources becomes important. We adapt the notation in our model from the Dincturk et al. model to identify states reached by the execution of an event (e.g., s_1 as a state reaching from s_0 by executing some e_1) in order to differentiate between candidate members of a path through a deferred representation as opposed to JavaScript-based interactions not part of a deferred representation. We define our paths through G as the set of embedded resources required to move from s_0 to s_i (that is, starting at s_0, an event moves the state to some vertex V_n in s_1).

We present a generic interaction tree of descendants in Figure 1. When we dereference a URI-R, we get a representation from the server; this is s_0. If there are two interactions available from s_0 (in this example, an onclick and an onmouseover event), we can execute the interactions to get to V_a or V_b from our root s_0 (note that the events required an external image to be retrieved). In this example, V_a (reached from s_0 via $onclick_1$) and V_b (reached from s_0 via $onmouseover_1$) are descendants of s_0 and are both s_1 in P from s_0. If new interactions are available from V_a, we can reach V_c and V_d, which are both s_2 in P from s_0 (similarly, we can reach V_e and V_f from V_b, peers of V_c and V_d).

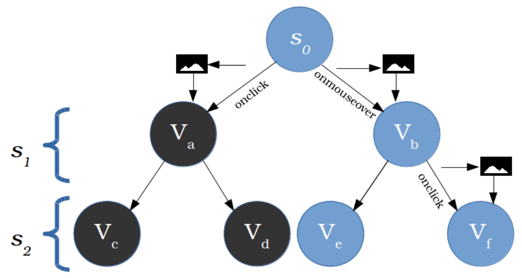

Figure 1: A generic, three-level client-side state tree with interactions as state transitions.

5 STATE EQUIVALENCY

Due to the archival focus of this study, we have a different concept of state equivalence than the Hypercube model. While Dincturk establishes state equivalence based on the DOM (using strict equivalence based on string comparison), we consider the embedded resources required to construct a descendant. We consider states to be equivalent if they rely on the same embedded resources. As such, we define the set of embedded resources for a descendant s_n as R_n.

Any two states with identical unordered sets of embedded resources are defined as equivalent. Two paths are identical if, over the course of each $s_n \in P$, the cumulative set of embedded resources required to render each descendant is identical. With a path P being the series of descendants from s_0 to s_n (meaning there is a path P_n for every leaf node in the tree), we define the set of embedded resources over the entire path as RP in Equation 2.

$$RP = \sum_{i=0}^{n \in P} R_n \tag{2}$$

We traverse all states within the interaction tree to understand what embedded resources are required by each state. If a state s requires a new embedded resource that has not yet been added to the crawl frontier, it is added as part of path P. From RP, we identify the archival coverage (using Memento). We also identify the duplicate URI-Rs by canonicalizing, trimming fragment identifiers from the URI-R, and using string comparisons to determine equality.

As an example, we present the state tree of a Bloomberg.com page in Figure 2. At s_0, the page has a menu at the top of the page with a mouseover event listener. Mousing over the labels initiates Ajax requests for JSON data, and the data is used to populate a sub-menu (s_1). The sub-menu has another mouseover menu that requests images and other JSON data to display new information, such as stock market data and movie reviews (s_2). Note that s_1 and s_2 are very broad given the number of menu items. This is an example of P through two levels of mouseover interactions that leads to new JSON and image embedded resources. While archival crawlers are motivated to discover and store the images, search engine crawlers are more interested in the initial representation rather than the embedded resources.

The page also has onclick events (not shown in Figure 2). These onclick events also lead to descendants at s_1, but not s_2. However, the onclick events lead to equivalent descendants that we identify as equivalent.

6 APPROACH

To measure descendants, we needed to construct a tool that can crawl, uncover, and understand descendants and deferred representations. We have previously shown that PhantomJS is an effective utility for crawling deferred representations [16]. We constructed a PhantomJS-based utility that dereferences a URI-R, identifies the interactive portions of the DOM (i.e., the DOM elements with event listeners), and constructs a tree of descendants, reached via initiating interactions and client-side events (just as in the Hypercube model). PhantomJS records the set of embedded resources requested by the client; in a production system, this would be the set of resources added to the Heritrix crawl frontier.

Because PhantomJS is closely tied to the DOM and client's JavaScript engine, race conditions and other event listener assignments prevents PhantomJS from understanding the entirety of events available on a representation. As such, we leveraged VisualEvent, a bookmarklet that is designed to visually display the DOM elements that have event listeners and JavaScript functions attached, to understand which

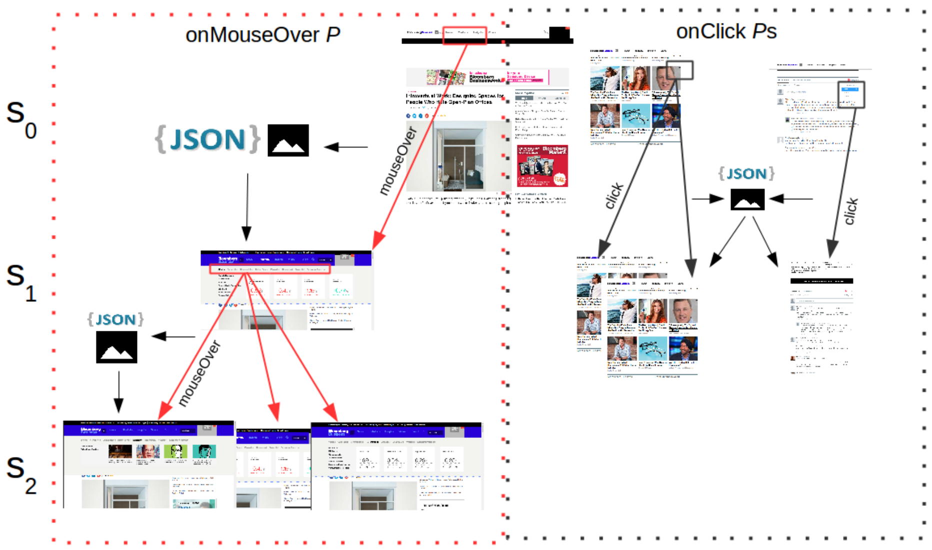

Figure 2: Example state tree of `http://www.bloomberg.com/bw/articles/2014-06-16/open-plan-offices-for-people-who-hate-open-plan-offices`. Mouseover events lead to multiple descendants at s_1 and further mouseover events lead to descendants at s_2, each requiring Ajax requests for JSON and image resources.

events and interactions can be executed on the client [10]. Our PhantomJS tool uses the list of events identified by VisualEvent to construct a set of interactions E that may lead to descendants. PhantomJS performs an exhaustive depth-first traversal of all possible combinations of interactions. Post-mortem, we perform state equivalence and identify the number of unique paths P, states s_n, and embedded resources RP that a crawler would have to visit in order to comprehensively archive the resources needed to provide full functionality in a memento.

We use the same 440 URI-R dataset from our prior investigation of crawling deferred representations [16]. We generated URI-Rs by randomly generating Bitly strings and identifying their redirection targets. We used PhantomJS to identify each URI-R as having a deferred or nondeferred representation and identify the number and type of descendants and interactions available on the representations of URI-Rs in this set, along with the descendants within the interaction tree and embedded resources required to build the descendant representations.

We consider a descendant that is a candidate to add to the tree identical to another descendant within the tree if the set of interactions to reach the descendant are identical. If we encounter a potential descendant that is reachable by the same interactions as another descendant within the tree, we do not add the descendant to the tree because the descendant already exists within the tree[5].

To crawl descendants, we begin by using PhantomJS to dereference a URI-R at s_0, and use VisualEvent to extract the interactive elements. We identify all possible combinations of interactions and use them as an interaction frontier, and iterate through the interaction frontier to crawl s_1. From s_1, we extract all possible interactions available and add them to the interaction frontier. We iterate through the interaction frontier until we have exhausted all possible combinations of interactions at each s_n. At the end of each s_n construction, we run state deduplication. We deem two interaction scripts as equivalent if they perform identical actions in identical order ($\{e_i, e_{i+1}, ..., ei + n\} = \{e_j, e_{j+1}, ..., ej + n\}$).

7 EDGE CASES

The approach that we identify in Section 6 is suitable for most of the deferred representations that a web user may encounter while browsing. However, deferred representations with certain conditions are not handled by our approach. Some representations use a DIV overlayed on the entire window area and identify interactions and events according to the pixel the user clicks. This creates an interaction frontier of (Width $\times$ Height)! or $2,073,600!$ for a screen size of 1920×1080 pixels. Due to this massive frontier size, we omit such interactions. Mapping (e.g., Google Maps) and similar applications that might have a near-infinite descendants are outside the scope of this work.

For these style of deferred representations, a *canned* set of interactions (e.g., pan once, zoom twice, pan again) would be more useful [10]. With enough of these canned interactions, a sizable portion of the descendants can be identified by a crawler over time, with coverage scaling with the number of executions performed. This is the archival equivalent of the Halting Problem – it is difficult to recognize when the crawler

[5]Note that this refers to equivalency of interaction scripts, meaning the crawler should not visit this state, rather than two states that are reached with different interactions but have the same sets of embedded resources (Section 5).

Event Type	Deferred		Nondeferred	
	Average	s	Average	s
Depth	0.47	0.5	0	0
Breadth	36.16	97.15	0.53	2.62
Descendants	38.5	780.72	0.62	2.81

Table 1: The average distribution of descendants within the deferred representation URI-R set.

# States	Deferred	Nondeferred
Min	0	0
Max	7,308	13
Median	1 (occurrences 17)	0 (occurrences 119)

Table 2: The range of descendants varies greatly among the deferred representations.

has captured *enough* of the embedded resources, when it should stop, or when it has captured everything.

8 DESCENDANT STATES

During our crawl of the 440 URI-Rs, we classified each as having a deferred or nondeferred representation. As previously discussed, URI-Rs with deferred representations will have an event that causes the JavaScript and Ajax in the representation to request additional resources. We dereferenced each of our 440 URI-Rs and identified 137 URI-Rs with nondeferred representations and 303 URI-Rs with deferred representations.

8.1 Dataset Differences

The nondeferred URI-Rs have a much smaller graph of descendants, and therefore we expect them to be easier to crawl. The nondeferred representation set of URI-Rs had $\overline{|S_{descendants}|} = 0.62$ per URI-R ($s = 2.81$, $M = 101$) as shown in Table 1. Nondeferred representations have a depth (i.e., max length of the P) of 0 (after state deduplication) since there are no new states reached as a result of the first round of interactions (that is, the set of interactions available in the initial representation does not grow as a result of subsequent interactions). Nondeferred representations have descendants at s_1 but the descendants do not result in additional embedded resources. However, there are 0.53 interactions or events in s_0 that, without our *a priori* knowledge of the dataset, may lead to new states or event-triggered requests for new embedded resources.

Deferred representations are much more complex, with $\overline{|S_{descendants}|} = 38.5$ per URI-R ($s = 780.72$). The standard deviation of the sample is quite large, with the number of states varying greatly from resource to resource. For example, the maximum number of descendants for a URI-R is 7,308 (Table 2). Further, there are many interactions available in deferred representations (36.16 per URI-R). Surprisingly, deferred representations are relatively *shallow*, with an average depth of 0.47 levels beyond the first set of interactions per URI-R ($s = 0.5$) and a maximum path depth of 2. This is counter to our initial intuition that deferred representations would have large, deep trees of interactions to traverse to retrieve all of the possible embedded resources for all descendants[6].

Event Type	Percent of URI-Rs		Contribution to RP_{new}
	Deferred	Nondeferred	
click	62.11%	4.29%	63.2%
mouseover	25.26%	3.00%	4.7%
mousedown	16.84%	1.72%	2.8%
blur	14.74%	0.86%	9.8%
change	11.58%	2.14%	0.0%
mouseout	8.42%	0.00%	0.8%
submit	6.32%	0.00%	0.0%
unload	5.26%	0.00%	1.2%
keydown	4.21%	0.00%	0.2%
focus	4.21%	0.00%	0.0%
keypress	2.11%	0.00%	5.5%
focusout	1.05%	0.00%	0.0%
dblclick	1.05%	0.00%	0.0%
submit	0.10%	0.43%	0.9%
mouseup	0.00%	0.86%	0.0%
focus	0.00%	0.43%	0.0%
other	29.47%	0.86%	11.0%

Table 3: Breakdown of the URI-Rs with various events attached to their DOMs and the percent of all new embedded resources contributed by the events.

The types of events on the client also vary depending on the event that is executed to create the new s_n. For example, onclick events are prevalent in URI-Rs with deferred representations, with 62.11% of all URI-Rs containing an onclick event (Table 3). Even in the nondeferred set of URI-Rs, 4.29% of the URI-Rs have an onclick event attached to their DOM. While other events occur with relative frequency, clicks dominate the initiated requests for additional embedded resources in deferred representations (Table 3), with onclick events being responsible for initiating the requests for 63.2% of new embedded resources (and, by definition, 0% in the nondeferred representation set). Recall that Rosenthal et al. are using only click interactions to interact with pages. Table 3 suggests that their approach is effective considering most events are onclick events and the highest value target event (i.e., the most embedded resources are discovered through onclick events).

8.2 Traversing Paths

As we discussed in Section 5, P identifies a unique navigation through descendants to uncover the URI-Rs of new embedded resources. In our dataset, we uncovered 8,691 descendants (8,519 for the deferred set, 172 for the nondeferred

[6]Our dataset and toolset omits the edge cases in Section 7.

set) as a result of client-side events, which is 19.7 descendants per URI-R. However, we only identified 2,080 paths through these descendants to uncover all of the new embedded resources, which is 4.7 paths per URI-R.

Nondeferred representations have more embedded resources (R_0= 31.02 per URI-R) than their deferred counterparts (R_0= 25.39 per URI-R) at their initial s_0. The paths P through the descendants are responsible for uncovering 54,378 new embedded resources (out of 66,320 total). That is, $|R_0|$=11,942 (7,692 from the deferred representations and 4,350 from the nondeferred representations), $|R_0+R_1|$=56,957, and $|R_0+R_1+R_2|=|RP|$=66,320. This shows that traversing P_n to reach s_1 and s_2 will significantly increase the crawl frontier beyond the base case of s_0, but crawling s_1 provides larger contributions to the frontier than both s_0 and s_2. As we mentioned in Section 8.1, the depth of the deferred representations was shallow (max($|P|$)=2). However, the majority of the URI-Rs added to the crawl frontier were identified by exploring the paths P of the descendants.

Note that out of the total 8,691 total descendants, the nondeferred representations have only 138 occurrences of s_0 and 34 occurrences of s_1. The deferred representations have 6,051 occurrences of s_1 and 2,468 occurrences of s_2. Following P through each s_1 adds R_1=53,706 URI-Rs to the crawl frontier, or 8.88 URI-Rs per descendant. This shows that deferred representations have many more descendants than nondeferred representations. Since R_2=10,208, we add, on average, 4.14 new URI-Rs to the frontier per s_2 followed in P. According to these averages, crawling s_1 provides the largest benefit to the crawl frontier. If we consider only the 2,080 paths that lead to new embedded resources, we would add 30.73 URI-Rs to the crawl frontier per P.

8.3 Impact on Crawl Time

Our prior work [16] measured crawl times for Heritrix and PhantomJS, including deferred and nondeferred representations. We measured that Heritrix is 12.15 times faster than PhantomJS (2.065 URIs/second versus 0.170 URIs/second, respectively). Using these results and the set of states S that PhantomJS can uncover and visit, we calculate the expected frontier size and crawl time that we can expect during a crawl of our 440 URI-Rs. Using these metrics, we calculated the s_0 crawl time for Heritrix-only crawls, PhantomJS crawls of only the URI-Rs with deferred representations, and s_1 and s_2 uncovered by our PhantomJS utility.

As we note in Table 4, s_1 has the greatest addition to the crawl frontier. As shown in Table 4, crawling s_0, s_1, and s_2 will lead to a crawl time 38.9 times longer than using only Heritrix to perform the crawl, but will also discover and add to the crawl frontier 15.60 times more URI-Rs. Alternatively, crawling only s_0 and s_1 will have a crawl time 27.04 times longer than Heritrix-only crawls, but will add 13.40 times more URI-Rs to the frontier for an improved frontier size return on the time-based crawling investment.

	H-only	s_0	s_1	s_2
Time (s)	1,035	8,452	27,990	40,258
Size (URI-Rs)	4,250	11,942	56,957	66,320
Time Increase	-	8.12x	27.04x	38.90x
Size Increase	-	2.81x	13.40x	15.60x
Growth rate per added unit time	-	0.35x	0.50x	0.40x

Table 4: The increases in run time and frontier size relative to the Heritrix-only (H-only) run.

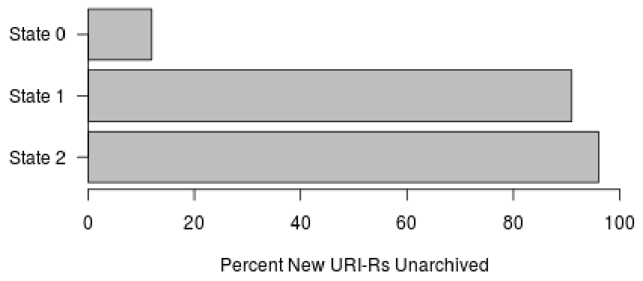

Figure 3: Embedded resources discovered in s_1 and s_2 are much more frequently unarchived (92% and 96%, respectively) than s_0 (12% unarchived).

9 ARCHIVAL COVERAGE

While the increases in frontier size as presented in Section 8 appear impressive, we can only identify the impact on the archives' holdings by identifying which embedded resources have mementos in today's archives (i.e., are shown to have been previously discovered). We used Memento to retrieve the TimeMap of each embedded resource's URI-R to determine whether the embedded resource had any mementos (i.e., has been archived before) or if the resource identified by the URI-R has not been previously archived.

The embedded resources from our entire set of 440 URI-Rs are very well archived – only 12% of the set of embedded resources in s_0 do not have a memento. This is consistent with the archival rates of resources from our prior studies [2, 14]. We only consider the *new* embedded resources in the descendants in s_1 and s_2. That is, we only consider the embedded resources added to the descendant that were not present in the previous state, or the set of resources R_{n+1} not a subset of the previous state's R_n. More formally, we define new embedded resources R_{new} in Equation 3.

$$R_{\text{new}} = \forall r \in (R_{n+1} - R_n), n \geq 0 \qquad (3)$$

However, the archival coverage of s_1 and s_2 is much lower, with 92% of R_{new} in s_1 missing from the archives (i.e., the URI-R of the embedded resource does not have a memento), and 96% of R_{new} in s_2 missing from the archives. This demonstrates that the embedded resources required to

URI-R	Occurrences
ads.pubmatic.com/AdServer/js/showad.js #PIX&kdntuid=1&p=52041&s=undefined&a=undefined&it=0	1782
edge.quantserve.com/quant.js	1656
www.benzinga.com/ajax-cache/market-overview/index-update	1629
ads.pubmatic.com/AdServer/js/showad.js	1503
www.google-analytics.com/analytics.js	1330
b.scorecardresearch.com/beacon.js	1291
www.google-analytics.com/ga.js	1208
www.google.com/pagead/drt/ui	1151
js.moatads.com/advancedigital402839074273/moatad.js	1112
a.postrelease.com/serve/load.js?async=true	907
Total	12,239

Table 5: The top 10 URI-Rs that appear as embedded resources in descendants make up 22.4% of all resources added to the crawl frontier.

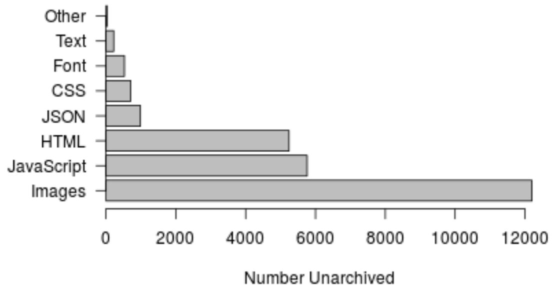

Figure 4: Images, JavaScript, and HTML are the most frequently occurring unarchived resources in deferred representations, with some unarchived images being quite large.

construct descendants are not well archived. Because s_0 is highly visible to crawlers such as Heritrix and archival services like Archive.is [4], it is archived at a much higher rate than the descendants (Figure 3).

In deferred representations, the unarchived embedded resources are most frequently images (Figure 4), with additional JavaScript files edging out HTML as the second most frequently unarchived MIME-type. The unarchived images specific to deferred representations vary in size between near 0B to 4.6MB, and average 3.5KB. The majority of images are small in size but several are quite large and presumably important (according to our importance metric D_m which evaluates the relative importance of embedded resources [13]).

We also observe a large amount of overlap between the embedded resources among descendants. For example, the top 10 embedded resources and their occurrence counts are

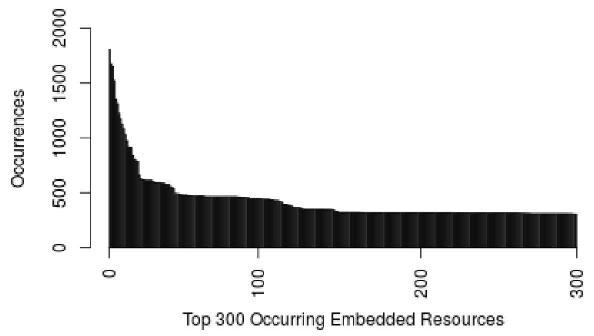

Figure 5: The occurrence of embedded resources loaded into deferred representation descendants.

provided in Table 5 (we trim the session-specific portions of the URI-Rs for comparison purposes). In all, just the top 10 occurring embedded resources account for 22.4% of R_{new} discovered by traversing through the paths. In theory, if we can archive these embedded resources once, they should be available for their peer mementos while in the archives.

The resources in Table 5 are mostly ad servers and data services such as Google Analytics [7]. The top 300 occurring embedded resources in our entire crawl frontier are graphed – in order of most frequent to least frequently occurring – in Figure 5. We also measured the number of R_{new} and the deduplicated crawl frontier if we crawl all descendants with deferred representations per URI-R. The largest 10% of our frontier contributes 91% of RP; that is, a large portion of the discovered crawl frontier is shared by our seed list.

10 STORING DESCENDANTS

The International Internet Preservation Consortium proposed [20, 21] an additional set of JSON metadata to better represent deferred representations and descendants in WARCs [43]. We adapt the metadata to describe descendants and include the interactions, state transitions, rendered content, and interactive elements (Table 6).

We present a summary of the storage requirements for descendants in Table 7. If we write out the JSON describing the states, transitions, rendered content, and other information, it would add, on average, 16.45 KB per descendant or memento. With 8,691 descendants, a total of 143 MB of storage space will be required just for the metadata, along with the storage space for the representations of the 54,378 new embedded resources.

The embedded resources discovered in our crawl average 2.5 KB in size. The embedded resources at s_0 were 2.6 KB on average, and the newly discovered embedded resources, as a result of deferred representations, were 2.4 KB in size on average. We estimate that nondeferred representations, which have 31.02 embedded resources on average, would require 80.7 KB per URI-R, or 11.1 MB of storage for the 137

[7]We have previously demonstrated the value of archiving advertisements and other data services [11].

Field Name	Data within field
startedDateTime	Timestamp of interactions (no change from WARC Spec)
id	ID of s_n represented by these interactions and resulting R_n.
title	URI-R of the descendant
pageTimings	Script of interactions (as CSV) to reach s_n from s_0. E.g., click button A, click button B, then double click image C.
comment	Additional information
renderedContent	The resulting DOM of s_n.
renderedElements	RP from s_0 to s_n.
map	The set of interactions available from s_n that will transition to s_{n+1}.

Table 6: JSON object with the IIPC-proposed metadata fields representing s_n stored as the deferred representation metadata of a WARC.

Storage Target	Size
JSON Metadata per descendant/memento	16.5 KB
JSON Metadata of all descendants	143 MB
Nondeferred (137 URIs)	
Average Embedded Resource	2.5 KB
Embedded Resources per URI	80.7 KB
Total embedded resource storage	11.1 MB
Total with JSON Metadata	13.4 MB
Deferred (303 URIs)	
Average Embedded Resource	2.6 KB
Embedded Resources per URI	70.0 KB
Embedded resource storage s_0	21.21 MB
Embedded resource storage s_1	108.0 MB
Embedded resource storage s_2	22.5 MB
Total with JSON Metadata	151.71 MB

Table 7: The storage impact of deferred representations and their descendants is 5.12 times higher per URI-R than archiving nondeferred representations.

URI-Rs in the collection. The storage requirement increases to 13.4 MB with the additional metadata.

Deferred representations have 25.4 embedded resources at s_0, or 70.0 KB per URI-R. For the 303 URI-Rs in the collection, s_0 would require 21.21 MB of storage. In s_1, the crawl discovered 45,015 embedded resources which requires 108.0 MB of additional storage, and 9,363 embedded resources at s_2, or an additional 22.5 MB of storage. In all, deferred representations require 151.71 MB of storage for the entire collection and all crawl levels. This is 11.3 times more storage than is required for the nondeferred representations, or 5.12 times more storage per URI-R crawled.

If we consider the July 2015 Common Crawl [29] as representative of what an archive might be able to crawl in one month (145 TB, 1.81 billion URIs), an archive would require 597.4 TB of additional storage (for a total of 742.4 TB) to house descendants and metadata. If we assume that the July crawl is a representative sample, an archive would need 7.17 PB of additional storage per year. Alternatively, can also say that an archive will miss 7.17 PB of data per year because of deferred representations.

11 CONCLUSIONS

In this paper, we present a model for crawling deferred representations by identifying interactive portions of pages and discovering descendants. We adapt prior work by Dincturk et al. and present a FSM to describe descendants and estimate storage requirements for the descendants.

We show that the deferred representations from our 440 URI-R sample set have 38.5 descendants per URI-R, and are surprisingly shallow, only reaching a depth of two levels. This means that these deferred representations are shallower than originally anticipated (but also very broad) and therefore it is more feasible to completely archive deferred representations using automated methods than previously thought. Archives that do not execute JavaScript during archiving are incomplete; 69% of URIs have descendants and 96% of the embedded resources in those descendants are not archived.

Crawling all descendants is 38.9 times slower than crawling with only Heritrix, but adds 15.60 times more URI-Rs to the crawl frontier than Heritrix alone.

Crawling all descendants in the sample is 38.9 times slower than crawling with only Heritrix, but adds 15.60 times more URI-Rs to the crawl frontier than Heritrix alone. However, most of R_{new} (newly discovered by traversing the paths) are unarchived (92% unarchived, and assumed to be undiscovered, at s_1 and 96% at s_2). However, 22.4% of the newly discovered URI-Rs match one of the top 10 occurring URI-Rs, indicating a high amount of overlap within RP; mostly, these are ad servers and data-services like Google Analytics.

In the future, we will work to incorporate PhantomJS into a web crawler to measure the actual benefits and increased archival coverage realized when crawling deferred representations. We will also work to develop an approach to solve our current edge cases (Section 7), including a way to handle applications like mapping applications using our automated approach along with an approach using "canned interactions". Our goal is to understand how many executions of canned interactions are necessary to uncover an acceptable threshold of embedded resources (e.g., how many pans and zooms are needed to get all Google Maps tiles for all of Norfolk, VA, USA?). We will also investigate filling out forms similar to Rosenthal et al. [40].

Our work presented in this paper provides measurements for a well-known phenomenon as it occurs in a small sample of URI-Rs, establishes an understanding of how much web archives and crawlers are missing by not accurately crawling deferred representations, and presents a process for better archiving descendants. We demonstrate that archiving deferred representation is a less daunting task with regards to crawl time than previously thought, with fewer levels of interactions required to discover all descendants. The increased frontier size and associated metadata will introduce storage challenges with deferred representations requiring 5.12 times more storage.

12 ACKNOWLEDGMENTS

This work was supported in part by NSF 1526700 and NEH HK-50181. The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author. Approved for Public Release; Distribution Unlimited. Case Number 16-1039.

REFERENCES

[1] S. Ainsworth. Web Archiving in Popular Media. http://ws-dl.blogspot.com/2016/09/web-archiving-in-popular- media.html, 2016.

[2] S. Ainsworth, A. Alsum, H. SalahEldeen, M. C. Weigle, and M. L. Nelson. How much of the Web is archived? In *Proceedings of the JCDL 2011*, pages 133–136, 2011.

[3] S. Ainsworth, M. L. Nelson, and H. Van de Sompel. A framework for evaluation of composite memento temporal coherence. Technical Report arXiv:1402.0928, arXiv, 2014.

[4] Archive.is. Archive.is. http://archive.is/, 2013.

[5] V. Banos and Y. Manolopoulos. A Quantitative Approach to Evaluate Website Archivability Using the CLEAR+ Method. *IJDL*, 17(2), pages 119–141, 2015.

[6] M. Ben Saad and S. Gançarski. Archiving the web using page changes patterns: A case study. In *Proceedings of the JCDL 2011*, pages 113–122, 2011.

[7] A. Bright. Web evidence points to pro-Russia rebels in downing of MH17. http://www.csmonitor.com/World/Europe/2014/0717/Web- evidence-points-to-pro-Russia-rebels -in-downing-of-MH17-video, 2014.

[8] J. F. Brunelle. Replaying the SOPA Protest. http://ws-dl.blogspot.com/2013/11/2013-11-28-replaying- sopa-protest.html, November 2013.

[9] J. F. Brunelle. Google and JavaScript. http://ws-dl.blogspot.com/2014/06/2014-06-18-google-and- javascript.html, 2014.

[10] J. F. Brunelle. PhantomJS+VisualEvent or Selenium for Web Archiving? http://ws-dl.blogspot.com/2015/06/2015-06-26-phantomjsvisualevent-or.html, 2015.

[11] J. F. Brunelle. Scripts in a Frame: A Framework for Archiving Deferred Representations *PhD Dissertation*, 2016.

[12] J. F. Brunelle, M. Kelly, H. SalahEldeen, M. C. Weigle, and M. L. Nelson. Not All Mementos Are Created Equal: Measuring The Impact Of Missing Resources. In *Proceedings of JCDL 2014*, pages 321 – 330, 2014.

[13] J. F. Brunelle, M. Kelly, H. SalahEldeen, M. C. Weigle, and M. L. Nelson. Not All Mementos Are Created Equal: Measuring The Impact Of Missing Resources. *IJDL*, 16(3), pages 283–301, 2015.

[14] J. F. Brunelle, M. Kelly, M. C. Weigle, and M. L. Nelson. The Impact of JavaScript on Archivability. *IJDL*, 17(2), pages 95–117, 2015.

[15] J. F. Brunelle and M. L. Nelson. Zombies in the archives. http://ws-dl.blogspot.com/2012/10/2012-10-10-zombies-in-archives.html, 2012.

[16] J. F. Brunelle, M. C. Weigle, and M. L. Nelson. Archiving Deferred Representations Using a Two-Tiered Crawling Approach. In *Proceedings of iPRES 2015*, 2015.

[17] M. E. Dincturk, G.-V. Jourdan, G. V. Bochmann, and I. V. Onut. A Model-Based Approach for Crawling Rich Internet Applications. *ACM Transactions on the Web*, 8(3):19:1–19:39, July 2014.

[18] D. A. Fahrenthold. SOPA protests shut down Web sites. http://www.washingtonpost.com/politics/sopa-protests-to-shut-down-web-sites/2012/01/17/gIQA4WYl6P_story.html, January 2012.

[19] G. Gray and S. Martin. Choosing a sustainable web archiving method: A comparison of capture quality. *D-Lib Magazine*, 19(5), May 2013.

[20] IIPC. Minutes of the WARC revision workshop. http://iipc.github.io/warc-specifications/specifications/ warc-format/meetings/2015-05-01-IIPC-GA-WARC- Meeting-Minutes/, 2015.

[21] IIPC. Proposal for Standardizing the Recording Rendered Targets. http://nlevitt.github.io/warc-specifications/specifications/warc-rendered-targets/recording-screenshots.html, 2015.

[22] Internet Archive. Brozzler. https://github.com/internetarchive/brozzler, 2017.

[23] P. Jack. Extractorhtml extract-javascript. https://webarchive.jira.com/wiki/display/Heritrix/ExtractorHTML+extract-javascript, 2014.

[24] M. Kelly, J. F. Brunelle, M. C. Weigle, and M. L. Nelson. On the Change in Archivability of Websites Over Time. In *Proceedings of TPDL 2013*, pages 35–47, 2013.

[25] I. Kreymer. Browsertrix: Browser-Based On-Demand Web Archiving Automation. https://github.com/ikreymer/browsertrix, 2015.

[26] I. Kreymer. Webrecorder.io. https://webrecorder.io/, 2015.

[27] A. LaFrance. Raiders of the Lost Web. http://www.theatlantic.com/technology/archive/2015/10/ raiders-of-the-lost-web/409210/, 2015.

[28] J. Lepore. The Cobweb: Can the Internet be Archived? *The New Yorker*, January 26, 2015.

[29] S. Merity. July 2015 Crawl Archive Available. http://blog.commoncrawl.org/2015/08/ july-2015-crawl-archive-available/, 2015.

[30] A. Mesbah, E. Bozdag, and A. van Deursen. Crawling Ajax by inferring user interface state changes. In *Proceedings of ICWE 2008*, pages 122 –134, 2008.

[31] A. Mesbah and A. van Deursen. Invariant-based automatic testing of Ajax user interfaces. In *Proceedings of CSMR 2009*, pages 210–220, 2009.

[32] A. Mesbah, A. van Deursen, and S. Lenselink. Crawling Ajax-Based Web Applications Through Dynamic Analysis of User Interface State Changes. *ACM Transactions on the Web*, 6(1):3:1–3:30, Mar. 2012.

[33] G. Mohr, M. Kimpton, M. Stack, and I. Ranitovic. Introduction to Heritrix, an archival quality web crawler. In *Proceedings of IWAW 2004*, 2004.

[34] K. C. Negulescu. Web Archiving @ the Internet Archive. Presentation at the 2010 Digital Preservation Partners Meeting, 2010.

[35] NetPreserve.org. IIPC Future of the Web Workshop – Introduction & Overview, 2012.

[36] PhantomJS. http://phantomjs.org/, 2013.

[37] N. Potter. Wikipedia Blackout: Websites Wikipedia, Reddit, Others Go Dark Wednesday to Protest SOPA, PIPA. http://abcnews.go.com/Technology/wikipedia-blackout-websites -wikipedia-reddit-dark-wednesday- protest/story?id=15373251, January 2012.

[38] S. Reed. Introduction to Umbra. https://webarchive.jira.com/wiki/display/ARIH/ Introduction+to+Umbra, 2014.

[39] D. S. H. Rosenthal. Talk on Harvesting the Future Web at IIPC2013. http://blog.dshr.org/2013/04/talk-on-harvesting-future-web-at.html, 2013.

[40] D. S. H. Rosenthal, D. L. Vargas, T. A. Lipkis, and C. T. Griffin. Enhancing the LOCKSS Digital Preservation Technology. *D-Lib Magazine*, 21(9/10), September/October 2015.

[41] K. Sigurðsson. Incremental crawling with Heritrix. In *Proceedings of IWAW 2005*, 2005.

[42] P. Singer, D. Helic, A. Hotho, and M. Strohmaier. HypTrails: A Bayesian Approach for Comparing Hypotheses About Human Trails on the Web. In *Proceedings of WWW 2015*, pages 1003–1013, 2015.

[43] Technical Committee ISO/TC 46. The WARC File Format (ISO 28500). http://bibnum.bnf.fr/warc/ WARC_ISO_28500_version1_latestdraft.pdf, 2008.

[44] B. Tofel. 'Wayback' for Accessing Web Archives. In *Proceedings of IWAW 2007*, 2007.

[45] H. Van de Sompel, M. L. Nelson, R. Sanderson, L. L. Balakireva, S. Ainsworth, and H. Shankar. Memento: Time Travel for the Web. Technical Report arXiv:0911.1112, Los Alamos National Laboratory, 2009.