Using CrowdLogger for In Situ Information Retrieval System Evaluation

Henry Feild
Computer Science Department
Endicott College
Beverly, Massachusetts, USA
hfeild@endicott.edu

James Allan
Center for Intelligent Information Retrieval
University of Massachusetts
Amherst, Massachusetts, USA
allan@cs.umass.edu

ABSTRACT

A major hurdle faced by many information retrieval researchers—especially in academia—is evaluating retrieval systems in the wild. Challenges include tapping into large user bases, collecting user behavior, and modifying a given retrieval system. We outline several options available to researchers to overcome these challenges along with their advantages and disadvantages. We then demonstrate how CrowdLogger, an open-source browser extension for Firefox and Google Chrome, can be used as an in situ evaluation platform.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process*

Keywords

User studies, in situ studies, browser extensions

1. INTRODUCTION

There are many challenges in evaluating retrieval systems in situ, and in this paper, we consider the following three: acquiring large participant pools, collecting user behavior and feedback, and modifying a given retrieval system. In this paper, we will demonstrate how an open source system called CrowdLogger[1] can be used for the in situ evaluation of information retrieval (IR) systems and specifically how CrowdLogger addresses these three challenges.

CrowdLogger is currently implemented as a browser extension for Firefox and Google Chrome and logs user behavior client-side. It allows extensions of its own—called CrowdLogger Remote Modules (CLRMs)—so CrowdLogger can be downloaded by a large pool of users and researchers can create and distribute CLRMs to those users. CLRMs have access to a rich API; the available functions include the ability to manipulate the content of pages viewed by participants and to upload data to a server privately [3]. We describe the system in more depth in section 3 and demonstrate how it can be used for the evaluation of retrieval systems in the wild in Section 4. First, we provide some background on the three challenges we consider, how CrowdLogger addresses those challenges, and discuss the difference between CrowdLogger and similar systems in Section 2.

[1]`https://code.google.com/p/crowdlogger/`

LivingLab'13, November 1, 2013, San Francisco, CA, USA.
ACM 978-1-4503-2412-0/13/10
http://dx.doi.org/10.1145/2513150.2513164 ...$15.00.

2. BACKGROUND

In this section, we first detail the three problems focused on in this work and how CrowdLogger addresses them. Then we look at existing research related to CrowdLogger.

Participant pools. For search companies with large user bases, recruiting users is as simple as redirecting some fraction of users to a *test bucket*, which exposes a different algorithm or interface to users (referred to as *bucket testing* or *A/B testing* [6, 7, 9]). Many companies also have browser toolbars at their disposal, e.g., Google,[2] Yahoo!,[3] and Bing,[4] which potentially track user behavior client-side [12, 13, 14]. These toolbars can be used in conjunction with bucket testing to collect richer information about a user's activities beyond the search engine result page (SERP). At least one company has leveraged their employee network to recruit for in situ studies where logging is performed client-side via a browser extension [4]. Some academic researchers have executed a similar strategy, but only for library computers at their institution, which is likely a niche search use case [5]. All of these techniques allow a large number of participants to be studied, but the scale is difficult to achieve for researchers at smaller companies or institutions. With CrowdLogger, we explore pooling participants among researchers, creating a large community resource.

Collecting user data. With respect to logging user behavior and feedback, there are client-side, server-side, and mixed approaches. Researchers working with a product like a web search engine can add JavaScript to track users' interactions with the SERP. Proxies are an alternative, and work by examining all HTTP traffic for a user. In addition to injecting JavaScript into SERPs served via HTTP, proxies can also track behavior on other pages visited by participants. Proxies are not tied to any special browser or operating system, making them more generalized and easier to maintain than client-side solutions. Among the downsides, JavaScript

[2]`http://www.google.com/toolbar/ie/index.html`
[3]`http://toolbar.yahoo.com`
[4]`http://www.bingtoolbar.com/`

injection is limited to HTTP traffic—HTTPS traffic is only decrypted once it reaches the browser. For example, it is not possible to track behavior on an HTTPS Google SERP with a proxy. Proxies also require storing all user behavior server-side, which could raise privacy concerns among participants and institutional review boards alike. Another approach, which we noted is used by many search companies in the form of toolbars, is to use browser extensions. These reside client-side, can track activity across pages displayed in multiple browser windows and tabs, and can either store data locally or send it directly to a server. Storing data locally allows the use of stronger privacy mechanisms and requires less trust to be placed in server-side processes [3]. However, extensions are different for each browser, making code base management difficult. CrowdLogger is currently implemented as a browser extension. The bulk of the code base is shared between two browsers with switch statements to handle browser-specific aspects.

Adapting systems. In lab studies, systems under investigation can rely on small, simple, and antiquated data sets. With in situ experiments, the goal is to observe users' interactions with a system to assist them with their own information needs. In today's environment, that likely means that systems need to be capable of returning current results from large collections like the web or a set of niche sites. There are a few likely system setups to provide current results. First, as we mentioned previously, a commercial system can be adapted, e.g., a web search company could modify the interface or the underlying algorithms. For researchers not connected with a large-scale product like web search, this can be difficult. Among their options are 1) provide a niche service, e.g., search over a smaller collection such as Wikipedia or a forum, and asking participants to use it when conducting appropriate searches, 2) use an API such as the Bing API[5] and ask participants to use it in place of their normal search engine, or 3) modify existing services client-side, e.g., a Google SERP. Option 1 gives researchers full control over the algorithm and interface, but may require significant back-end development; option 2 limits modifications to the interface, unless the terms of service allow manipulation of search result rankings; option 3 allows researchers to modify the interface and the content, but only to a limited degree. For instance, a popular tool called SurfCanyon[6] modifies SERPs for major search engines by surfacing as-yet unseen search results from deeper in the rankings every time the user clicks on a result link (the goal is for the surfaced results to be similar to the clicked result). CrowdLogger offers an API for option 3 and CLRMs can point users to web services that utilize the other options.

Related work. Besides commercial browser toolbars that may or may not track activity, many other tools exist for performing some of the functionalities encapsulated in CrowdLogger. We will briefly describe some of these tools and compare them with CrowdLogger. One project, called the Lemur Query Log Toolbar,[7] was developed as a passive logger, with the goal of collecting a large search log for the information retrieval research community. Implemented as an extension for Firefox and Internet Explorer, it logged many user interactions, including web search queries, viewed page text, copied and pasted text, and scrolls. This data was uploaded once a week to a centralized location. The primary purpose of the Lemur Toolbar was to collect data passively and it did not offer any mechanisms to interact with users, e.g., to gather feedback, or to interact with content pages—features CrowdLogger supports. The only notion of privacy in the Lemur Toolbar was that each week's worth of data was stored under a unique, anonymized user identifier, whereas CrowdLogger offers an API for uploading data using a secret sharing sharing scheme.

Another browser extension for logging user behavior is the HCI Browser, introduced by Capra [1]. The HCI Browser supports tracking in interactive, task-based IR studies. It logs user behavior similar to that logged by the Lemur Toolbar, but it was not made to support passive logging, as CrowdLogger allows. It could be adapted to handle such situations, but it still suffers from the same drawbacks as the Lemur Toolbar. Both of these loggers are open source and can be modified by researchers for other ends, but doing so requires understanding the internals, reworking them, and then releasing the software to a new batch of participants. In contrast, CrowdLogger allows researchers to develop specialized CLRMs, which have access to APIs for abstracting browser-specific tasks like logging, and can be installed, uninstalled, and updated on the fly. As a side effect, CrowdLogger can maintain a large pool of active participants that span multiple studies. Since CrowdLogger is open source, researchers can modify the underlying system to suit their particular needs (though they will lose access to the shared user base), just as with the Lemur Toolbar and the HCI Browser.

Lagun and Agichtein [8] introduced a web proxy called ViewSer that injects JavaScript into SERPs to blur out all search results that the mouse is not hovering over. The goal of ViewSer is to provide a mechanism for conducting remote user studies of SERP interactions where eye tracking hardware is not practical. Since the user must point the mouse at a result to unblur it, the software can track where the user is most likely looking at any given time. However, the tracking only occurs on SERPs. The functionality of ViewSer could be easily implemented as a CLRM and used with CrowdLogger, and would additionally give researchers access to post-SERP activity.

Tretter and Golovchinsky [11] released a Google Chrome browser extension called SearchPanel that displays a panel on SERPs indicating which results a user has visited before. By default, the extension uploads and stores the MD5 hash of pages visited and search queries submitted for research purposes. In a limited way, this model of providing users with a service (in this case, the SearchPanel) in exchange for data is very similar to commercial toolbars. We designed CrowdLogger with this in mind. CLRMs are not restricted to remote studies; 'apps' are also supported (the key difference being that apps are not offered with research in mind). For example, the Search Task Assistant [2] CLRM[8] displays a panel that automatically organizes a user's search history into tasks. By making available a number of useful apps like this to CrowdLogger users,[9] we hope to entice users to install CrowdLogger and participate in studies.

[5]https://datamarket.azure.com/dataset/5BA839F1-12CE-4CCE-BF57-A49D98D29A44

[6]http://www.surfcanyon.com/

[7]http://www.lemurproject.org/querylogtoolbar/

[8]https://code.google.com/p/crowdlogger/source/browse/#git/unconfigured/clrms/search-task-assistant

[9]Currently, we only have the one app. We hope that the IR community can help develop additional apps.

3. CROWDLOGGER

CrowdLogger serves as a client-side platform, providing a rich API for tracking user behavior, interacting with web pages and users, communicating with servers, and uploading data privately. CrowdLogger supports application and study modules (CLRMs)—extensions engineered by researchers to run within CrowdLogger. To carry out an in situ study, a researcher must develop a study module and upload it to the CrowdLogger web server. The study module will then show up in CrowdLogger's version of an 'App Store', where participants can install and participate in the study. Since CrowdLogger is open source, anyone can become their own CrowdLogger maintainer. However, one of our goals is to establish a large pool of shared users to allow easy recruiting on a larger than conventional scale for academic studies. Thus, we recommend that the community distribute their application and study modules through one common CrowdLogger maintainer. For example, we run a CrowdLogger instances at the University of Massachusetts.[10]

The API currently exposed by CrowdLogger includes the following classes: user activity, local storage, server communication, privacy, and user interface. These are all implemented features. We briefly describe them below.

• **User activity.** This class is broken into two parts: user history and real-time events. CrowdLogger tracks certain interactions in the browser, namely page loads, focuses, and blurs, link clicks, searches and result sets on SERPs (currently only Google, Yahoo!, and Bing pages are supported), tab adds, selects, and removals, and logging status changes. All data is logged to a database on the client (currently using HTML5 IndexedDB), shared among all CLRMs. The user history interface provides CLRMs access to this database and all of its contents. In addition, when these interactions are first detected, CrowdLogger sends out alerts to all CLRMs that have registered to listen for such events; registration for these events is provided via the real-time events interface. The user activity API prevents multiple researchers from having to programming the same logging functionality; this is particularly useful when considering complex technologies such Google Instant, where the SERP changes with every keystroke and the query terms associated with the results shown are not present in the search URL—managing this can be difficult.

• **Local storage.** The local storage interface gives users access to a simple-object storage database, allowing operations such as adding, removing, clearing, and listing stores; adding, reading, removing, and updating data; and removing the database. It also provides a sub-interface specifically for getting and setting preferences. This interface serves as an abstraction to the underlying database technology and variations in implementations across browsers, in this case, IndexedDB.

• **Server communication.** Researchers may need to communicate to servers for a variety of reasons, e.g., to query a data set or to perform computation that is too intense for a client machine. The server communication API allows this kind of communication.

• **Privacy.** Privacy was one of the major motivations for the original CrowdLogger [3]. However, not every study requires that privacy be maintained to the level we investigated in our previous work, e.g., using frequency thresholding or differential privacy techniques. These techniques encrypt data before uploading it to the server in such a way that individual pieces of data, e.g., a query, can only be decrypted if it is common to at least k participants. The privacy API exposes a function for client-side encryption (specifically, we implement Shamir's Secret Sharing Scheme [10]). The server-side code to decrypt the messages is also included in the CrowdLogger code base. This API allows researchers to provide this kind of privacy if they so desire.

• **User interface.** Finally, the user interface API gives researchers access to common functions, such as opening a window to interact with a user (e.g., display a feedback form), alert the user to something that requires their attention, or to inject custom scripts into content pages (e.g., to modify the page content or track an interaction event like mouse movements). With respect to something as seemingly trivial as opening browser windows, Chrome and Firefox behave differently, so this interface prevents researchers from worrying about which browser is being used. The function to inject JavaScript into content pages is crucial to CrowdLogger's ability to meet the second and third challenges we discussed earlier, namely, tracking user behavior and adapting IR systems for experimentation. This allows researchers to expand CrowdLogger significantly beyond its stand-alone capabilities.

In addition to the API, CrowdLogger has built in support to allow users to manually inspect and remove data in their history database and to export their data.

4. EVALUATING IR SYSTEMS IN SITU

In this section we first describe the general procedure to evaluate an IR system in the wild using CrowdLogger. To provide a clearer picture of how CrowdLogger works, we follow with a specific example.

4.1 General procedure

First, a researcher must program a CLRM. This involves three components: a set of core JavaScript files that run in the background, a set of HTML/CSS/JavaScript resource files (e.g., for displaying dialogs), and a JSON metadata description file.[11] Given these three, a researcher can assemble the CLRM using the `clrm-package.rb` script provided in the CrowdLogger code base. Once the CLRM is ready, it needs to be uploaded to the CrowdLogger web server being used for the study, e.g., the UMass CrowdLogger server—this is how the CLRM shows up in users' CLRM libraries. We are planning to add an option to allow users to specify additional CLRM repositories. Once available in the CLRM library, a user can install the study module and begin using it. Currently, the study CLRM is responsible for administering informed consent forms when appropriate.

4.2 Example

Suppose that a researcher, Bob, wants to develop a CLRM that measures the effects on result click-through rates of a SERP interface that indicates, for each result on the SERP, how frequently a user visits that domain. Bob decides to use a between-subjects design (similar to a bucket test); when a participants installs the CLRM, that user is either assigned to the treatment group (domain visitation frequencies indicated) or the control group (un-modified SERP interface).

[10]http://crowdlogger.org

[11]E.g., https://code.google.com/p/crowdlogger/source/browse/#git/unconfigured/clrms/demo

The code to assign a user to a group is placed in the core component of the CLRM. The code can use the server communication API to request the assignment from the server (e.g., if a Latin square design is being used), or just randomly pick. Either way, in the core component, Bob adds code to store the user's group using the local storage preferences API. He also adds code that registers a script that alters a SERP to include the domain visitation indicators to be injected on SERPs, but only if the user has been assigned to the treatment group. Bob stores the code for the injection script in a file in the resource directory and then referenced from the core code.

Bob is not interested in past user behavior, so he ignores the user history API. However, to track a user's click interactions with SERPs, he does register for SERP page loads and click events using the real-time events API. He updates the core code to include a function that saves information about clicks and non-clicks to his CLRM's database, e.g., the tuples *[treatment group, anonymized user id, MD5 hash of query, search engine, rank, MD5 hash of domain, domain visitation frequency, clicked?]*. Bob sets up a polling function that, once a day, uploads the data to a server using the server communication API.

Bob programs the CLRM to stop after four weeks of logging. When the fourth week is entered, a message is sent to the user via the user interface notification API to send an alert to the participant informing them that the study has concluded and they can uninstall the CLRM.

Bob compiles the CLRM and then uploads the CLRM to the UMass CrowdLogger server to make it available to all UMass CrowdLogger users, thus reducing the amount of manual recruiting he must perform.

5. MOVING FORWARD

There are two major directions we are considering in order to make CrowdLogger more useful and easier to develop for: a CLRM builder for researchers and a new client-side configuration to more easily support additional browsers as well as providing richer data access facilities, like exporting data in a particular format. The goal of the CLRM builder is to save researchers from programming common patterns, such as launching a feedback form or logging activity from a particular page. One possible implementation would be to create a web interface that also provides text areas where researchers can add in JavaScript for the parts that cannot be automatically built. The easier CrowdLogger is to develop for, the more likely researchers will use it.

One of the limitations of CrowdLogger is that it is only available for Google Chrome and Firefox. While this covers a large share of the browser market,[12] it does not cover all of it. One way we are considering approaching this is a hybrid system where CrowdLogger is installed as a desktop application with light-weight extensions for any browser that allows them. The desktop application takes care of CLRM management, saving data to the database, exporting data, etc. The light-weight browser extensions communicate with the desktop application to obtain the JavaScript to inject into content pages (only those not opened in Private Browsing or Incognito mode). The injected JavaScript will communicate with the desktop application via message passing with a small web server component of the desktop application. We are in the early stages of planning the system, but it should make CrowdLogger more useful and robust.

[12] https://en.wikipedia.org/wiki/Usage_share_of_web_browsers

Additionally, we must develop policies and controls over how multiple CLRMs are handled within a single CrowdLogger installation. Concerns include computational and storage loads on the client and the interaction of CLRMs that adapt the user experience. High computation/storage loads may negatively impact users' experience both inside and outside of the browser, causing them to uninstall CrowdLogger. Poorly managed CLRM interaction could cause uncertainty and unreliability in collected data. We are working towards policies to address these issues.

6. SUMMARY

We described several options available for overcoming three challenges faced when evaluating IR systems in situ: acquiring large participant pools, collecting user behavior and feedback, and modifying a given retrieval system. We then described how CrowdLogger, an open source extension for Google Chrome and Firefox, faces these challenges. We included an overview of CrowdLogger and described the steps a researcher must take to use CrowdLogger for an in situ study. Finally, we considered future directions we are considering to make CrowdLogger a more useful and friendly platform for conducting in situ research.

References

[1] R. Capra. HCI Browser: A tool for administration and data collection for studies of web search behaviors. In A. Marcus, editor, *Design, User Experience, and Usability*, volume 6770 of *Lecture Notes in Computer Science*, pages 259–268. Springer Berlin Heidelberg, 2011.

[2] H. Feild and J. Allan. Task-aware search assistant. In *SIGIR*, page 1015, 2012.

[3] H. Feild, J. Allan, and J. Glatt. CrowdLogging: Distributed, private, and anonymous search logging. In *SIGIR*, page . ACM, 2011.

[4] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *TOIS*, 23(2):147–168, 2005.

[5] Q. Guo and E. Agichtein. Ready to buy or just browsing?: detecting web searcher goals from interaction data. In *SIGIR*, pages 130–137. ACM, 2010.

[6] K. Haas, P. Mika, P. Tarjan, and R. Blanco. Enhanced results for web search. In *SIGIR*, SIGIR '11, pages 725–734. ACM, 2011.

[7] D. Hillard, S. Schroedl, E. Manavoglu, H. Raghavan, and C. Leggetter. Improving ad relevance in sponsored search. In *CIKM*, pages 361–370. ACM, 2010.

[8] D. Lagun and E. Agichtein. Viewser: enabling large-scale remote user studies of web search examination and interaction. In *SIGIR*, volume 11, pages 365–374, 2011.

[9] Y. Lu, F. Peng, X. Wei, and B. Dumoulin. Personalize web search results with user's location. In *SIGIR*, pages 763–764. ACM, 2010.

[10] A. Shamir. How to share a secret. *CACM*, 22(11):612–613, 1979.

[11] S. Tretter and G. Golovchinsky. SearchPanel, 2013. URL `http://searchpanel.wordpress.com/`.

[12] R. W. White and S. M. Drucker. Investigating behavioral variability in web search. In *WWW*, pages 21–30. ACM, 2007.

[13] R. W. White and D. Morris. Investigating the querying and browsing behavior of advanced search engine users. In *SIGIR*, pages 255–262. ACM, 2007.

[14] R. W. White, P. Bailey, and L. Chen. Predicting user interests from contextual information. In *SIGIR*, pages 363–370. ACM, 2009.