COS 561 Final Project: AccWeb – Improving Web Performance via Prefetching

Qizhe Cai Wei Hu Yueyang Qiu
{qizhec,huwei,yqiu}@cs.princeton.edu

Abstract

We present AccWeb (Accelerated Web), a web service that improves user experience of web browsing. AccWeb improves web performance by prefetching web resources when the user is typing in the address bar. In order to prefetch resources, AccWeb predicts the URL that the user wants to navigate. The prediction is based on the user's browsing history and typing behaviors. After a prediction is made, all demanded resources belonging to the predicted URL will be prefetched. Also, AccWeb gives users options to choose types of resources to prefetch – users can select the types of resources to prefetch based on their own preferences as well as real-time network condition. We build an AccWeb prototype, and our experiments show that the performance of browsers installing AccWeb is much better than the performance of browsers without AccWeb.

1 Introduction

Despite the prevalence of internet accesses on PCs and mobile devices, many users are complaining about their experiences of poor web performances. Current approaches to improving web performance mainly focus on designing new protocols (e.g. SPDY, HTTP/2) which make communication between servers and clients more efficient. For example, HTTP/2 reduces loading times by multiplexing multiple requests over a single TCP connection. However, with all these advancements, page loading times are still likely to exceed user tolerance limits for the foreseeable future, due to user expectations on lower loading times and richer web contents. In this project, instead of trying to further improve the speed of communications between servers and clients, we are interested in developing a user-oriented web service in order to speed up page loading time. In particular, we focus on the technique of prefetching web contents while the user is typing in the address bar.

Prefetching a web resource only makes sense when the resource is *static*, i.e., its URL pattern keeps unchanged for a long period of time. In order to evaluate whether prefetching is useful or not, we estimate the percentage of static web resources by testing the top 20 news websites on Alexa. Our result shows that around 70% of resources are static (see Fig. 1). Therefore, prefetching static web resources can indeed improve web performance. Moreover, the bandwidth and the number of TCP connections per domain of a browser for loading web resources are limited. If the static resources are prefetched, non-static resources can be loaded faster without the competitions from static ones.

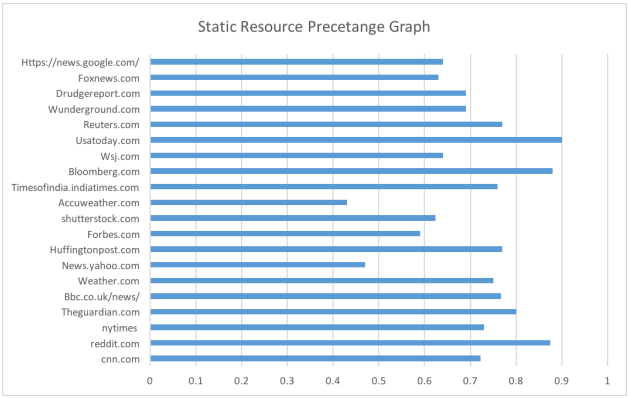

Figure 1: Percentages of static resources in Alexa's top 20 news websites.

In fact, some browsers such as Chrome already have built-in prefetching services based on URL prediction. However, we find that Chrome's prefetching mechanism falls short in the following aspects:

- URL Prediction is based on the user's browsing history. If the user clears browsing data, then

prediction cannot happen, which means no web resources can be preloaded.

- The set of URLs eligible to be prefetched is too conservative, compared with the URLs that the user has typed in the address bar before. Only a small number of websites that are visited most frequently can be prefetched by Chrome.

- When a URL is predicted, the majority of the web resources associated with that URL will be preloaded, regardless of the user's need. For that reason, the cost of prefetching unneeded web resources will be huge if Chrome predicts the URL incorrectly, especially when the network bandwidth is limited.

The goal of this project is to design a prefetching service with good performance while dealing with the above drawbacks of Chrome. We develop AccWeb, which runs in 3 separate stages. The first stage is analyzing. In this stage, AccWeb fetches and records all static web resources of URLs that the user has already visited in the past. The second stage is predicting. Based on the user's typing and browsing history, AccWeb predicts the URL that the user currently tries to navigate (if the prediction confidence is high enough) given the user's current input in the address bar. The third stage is prefetching. After receiving the predicted webpage, The URLs of the web resources on that page are sent through the HTTP link header field to the browser, and the browser starts to preload these resources.

AccWeb eliminates (or at least alleviates) the abovementioned drawbacks of Chrome. First, the user's browsing history is stored in a browser extension called Predictor, so flushing browser history in the browser does not affect prefetching in AccWeb. Second, we design a prediction mechanism that is less conservative than Chrome's, while ensuring a high confidence level. Third, we allow the users to have control over which kinds of resources to be prefetched. Note that different users may have different preferences regarding web performance: some users like to load images as quickly as possible, while others care more about the loading time of texts. In addition, under poor network condition, users might want to limit the prefetching service to lower the cost of wrong predictions and to save bandwidth. We implement different modes (e.g. full mode, limited mode, image mode, etc.) that users can select based on their preferences and network condition. The demo of AccWeb can be found: COS561 final project.

2 Design

In this section, we describe the design of AccWeb. The architecture and the workflow of AccWeb is shown in Fig. 2. It is divided into three stages by their functional purposes: analyzing, predicting, and prefetching.

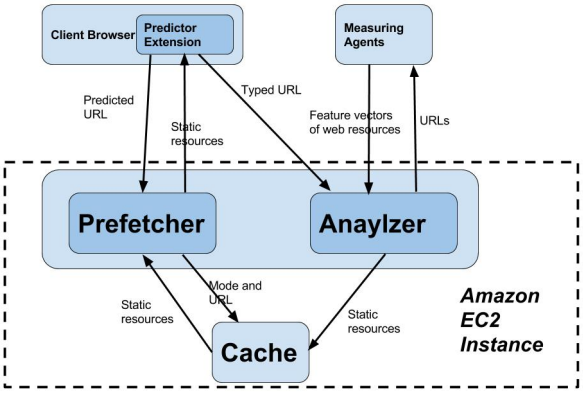

Figure 2: AccWeb architecture.

2.1 Analyzing

Analyzer is a Python server running on Amazon EC2 instance. Analyzer gets all URLs the user has visited and then sends these URLs to Measuring Agents. Measuring Agents navigate these websites, record all feature vectors of web resources, and send them back to Analyzer. A feature vector includes the type of resources, the time of sending requests, the time of getting responses, and so on. After analyzing data (which will be explained in Section 3), Analyzer forms potential responses for prefetching, and stores them in the cache.

2.2 Prediction

Predictor is a Chrome extension running on client browser and is responsible for predicting. Predictor records the user's typed texts and navigated URLs, and builds a prediction table. Utilizing this table,

we design an algorithm to predict the URL the user wants to navigate based on his/her current input in the address bar. Since Predictor stores all the data in the extension itself, when the user flushes the browsing history, it still can predict URL the user wants to navigate. Also, unlike Chrome, the set of URLs that will be prefetched in AccWeb is almost the same as the browsing history of the user, which allows much more flexibility. After Predictor sends the predicted URL to Prefetcher, Prefetcher will send back all resources that will be loaded. Details of our prediction algorithm will be explained in Section 3.

2.3 Prefetching

Prefetcher is a Python server running on Amazon EC2 instance. Depending on real-time network condition and the user's own preference, the user can select his/her favorite prefetching mode in the Chrome extension. The available modes are full mode, limit mode, image mode, script mode and DNS-prefetch mode. The details of these modes will be explained in Section 3. After receiving the predicted URL and the current mode from the extension, Prefetcher fetches corresponding resources from cache and sends them back to the extension.

3 Implementation Details

In this section, we present some implementation details of the three components of AccWeb.

3.1 Analyzing

There are three steps in Analyzer. First, Analyzer gets all static web resources, including the domain names for DNS prefetching and HTTP web requests for preloading. Second, Analyzer filters our static web requests that the the browser does not need to prefetch. Third, Analyzer forms link headers for each mode and stores headers in the cache.

In the first step, Analyzer gets all HTTP static requests. Measuring agents navigate the same URL multiple times, and send feature vectors of all HTTP web requests, such as types of web resource, time of sending requests, and time of getting responses, to Analyzer. These features will be used in different prefetching modes. For example, in the script mode, only resources whose type is script are prefetched. By checking the recurrence of web requests, Analyzer fetches all static HTTP web requests. Besides HTTP web requests, domain names are static resources as well, which can be used for DNS-prefetch.

Second, not all static HTTP requests need to be prefetched. Web developers set some web resources' expire date the same date as or even earlier than the current date in HTTP headers. So these web resources must be fetched every time when the browser loads webpages, even if resources have already been in the cache. In this case, Analyzer discards these resources from the cache. This step can reduce the number of prefetching resources, which is an important step for the limit mode. In the limit mode, AccWeb only selects six HTTP web requests to prefetch per domain . If some web resources becomes expired immediately after prefetched, then the limit mode will be ineffective since there are few prefetched resources that can be used during loading a web page.

After storing all static web resources into cache, Analyzer forms link HTTP headers for different modes and stores headers to cache as well. This step can save the time of forming headers when the browser preloads static resources in a web page. When Predictor sends predicted URL to Prefetcher, Prefetcher gets responses directly from cache rather than spends extra time on forming link headers. Reducing the total time of fetching static resources improves the web performance of loading websites.

3.2 Prediction

The goal of Predictor is to accurately predict the URL that the user wants to visit based on his/her current input in the address bar. We first need to build a prediction table to store the user's typing and browsing history, which can be updated when there is a new browsing record, and then use this table to make predictions.

Each entry in the prediction table records the user's browsing information about a (string, URL) pair. For every such pair in the table, we store 3 attributes: hit count, miss count, and confidence. We construct the table as follows. When the user types the string str1 in the address bar and visits the URL url1, then for all prefix pfx of str1:

- The hit count of (pfx, url1) will increase by 1. If the pair (pfx, url1) is not in the table, it will

be added to the table with hit count 1 and miss count 0.

- For any other URL url2 such that (pfx, url2) is in the table, the miss count of (pfx, url2) will increase by 1.

For each pair, the confidence is always equal to $\frac{\text{hit count}}{\text{hit count}+\text{miss count}}$.

Our rule of prediction is: when the user types a string str in the address bar, find the URL url that maximizes the confidence of (str, url) in the table. Additionally, this pair (str, url) has to satisfy two conditions: (i) the confidence is sufficiently large (say, at least 0.9); (ii) the hit count is not too small (say, at least 5). When making a prediction for prefetching, we really want the prediction to be correct otherwise the prefetching will be a waste of resources, so we impose these two restrictions. If either of these conditions is not satisfied, we will not make any prediction.

We also propose a variant that extends the above prediction method. By the end of every day, we multiply all the hit counts and miss counts in the table by a factor less than 1 (say, 0.9). This explicitly attaches more importance to more recent browsing activities, which makes sense since the user's interests and browsing habits might change over time.

3.3 Prefetching

All prefetching actions are done through adding a link header field in the response to Predictor. The link header field is an HTTP header that allows the server to point an interested client to another resource containing metadata about the requested resource. The basic format of a link header is: `Link: <meta.rdf>; rel=meta,`.

The string between "<" and ">" indicates the address of the web resource, and the value of `rel` specifies the relationship of the web resource to clients or the actions that the browser should do immediately or in the future. Available actions related to prefetching include dns-prefetch, preconnect, prefetch, preload and prerender. However, in AccWeb, we only use dns-prefetch and preload. After a browser sends any HTTP requests to a server, by adding link header field in the HTTP response header, a server can require the browser to get IP address of server (dns-prefetch), set up a TCP connection (preconnect), and load web resources (prefetch or preload).

Compared with sending requests directly by extensions on client browsers, retrieving web resources through link header field has several advantages. First, it is more convenient. The browser extension does not have to send requests directly. Second, types of web resources that can be prefetched are more flexible. The Chrome extension can only send HTTP requests and HTTP responses. Using link headers, AccWeb can ask the browser to send DNS queries, pre-establish TCP connections and send HTTP requests. This can give users more options to prefetch web resources. Third, requests are actually formed and sent by browsers when using link header fields. If web resources are in the cache of browsers, requests will not be formed. If a Chrome extension sends a web request, the browser will first check whether the cache of the browser contains the web resource and then decide whether it should send the request to servers. This process may take about 10 ms. Although the time is very short, we want to save as much time as possible.

DNS-prefetch mode

In the DNS-prefetch mode, AccWeb only prefetches IP addresses of domain names. This mode can be used when the network is in congestion. Prefetching IP addresses does not occupy much bandwidth, while AccWeb can help clients avoid DNS lookup time. The DNS-prefetch format of link header is: `Link: "<" + hostname + ">; rel=dns-prefetch,"`.

Full load mode

In the full load mode, AccWeb prefetches all static resources. This mode can be used when the network condition is pretty good and the user wants to load all the resources quickly. The link header looks like: `Link: "<" + url + ">; rel=preload,"`. This link header is also used in all the modes to be introduced below.

Limit mode

In order to reduce the cost of misprediction, save the bandwidth and still improve the web performance, we design a limit mode. In this mode, AccWeb

will prefetch only six HTTP web requests per domain. Prefetched HTTP requests are selected randomly. Also, Prefetching six resources per domain reduces the stalled time of web requests when loading webpages. Chrome only allows six TCP connections per domain in the HTTP/1.1 protocol, and the requests that cannot be fetched immediately are stalled in a queue. As some websites have more than thirty web requests in the same domain, the stalled time of web requests may be very long.

Image mode & script mode

The image mode and the script mode are for users who mainly concern about the loading time of image or script resources. Using these modes can save the bandwidth for loading other types of resources.

4 Evaluation

We perform some preliminary experiments to evaluate the effectiveness of AccWeb.

4.1 Evaluation setup

Our experiments are conducted using Chrome browser on a Macbook Pro laptop, running macOS Sierra, as the client. This client connects to a WiFi hotspot exported by a TP-LINK router. For simplicity, the client is both measuring agents for analyzing web resources and the user for browsing web pages. We load the full version of web pages using Google Chrome Version 55.0.2883.95 (64-bit) for Mac. We host Prefetcher and Analyzer in a small instance VM in Amazon EC2's US West Region. Predictor is a Google Chrome extension running on the browser of the client. Different experiments have small changes on the setup, which will be explained in the subsections below.

4.2 Page Loading Time

We evaluate the improvement in page loading times enabled by the full mode and the limit mode of AccWeb, compared with the loading times without AccWeb. We have two assumptions: first, we assume that our prediction is correct; second, we assume that before the user loads a web page, all static resources required to be prefetched are already loaded - when the browser starts to load web pages, the prefetched resources can be obtained from the cache. We select 50 websites from Alexa's top 150 websites in China, load 10 times for each web page, and record loading time of web pages for the full load mode, the limit mode, and non-prefetching mode. When testing non-prefetching mode, we disable Chrome's default prefetching service.

We plot the CDFs (across websites) of average loading times (Fig. 3) and CDFs of median loading times (Fig. 4) of the websites in the 3 different modes. As we can see from the plots, there are improvements in loading times when using AccWeb. For example, in the full mode, around 75% of web pages' average loading times are less than or equal to 2 seconds, while in the non-prefetching mode this fraction is 64%. This improvement is also evident in median loading times.

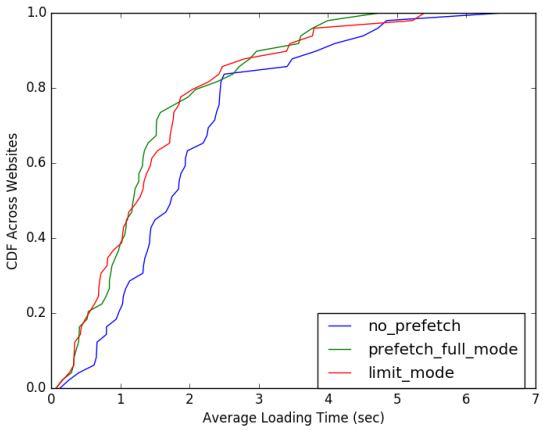

Figure 3: CDFs of average loading times of 50 websites in full mode, limit mode, and non-prefetching mode.

4.3 Number of HTTP Reqeusts Prefetched

We use the same 50 websites from the last experiment to test the number of HTTP requests prefetched in the full mode and the limit mode. For each website, we record the numbers of web requests that are prefetched in both modes. The resulting CDFs are plotted in Fig. 5. We can see that the number of requests prefetched in the full mode significantly exceeds the number of requests in the limit mode. In the limit mode, the maximum number of web requests is less than 125. For about 80% of websites,

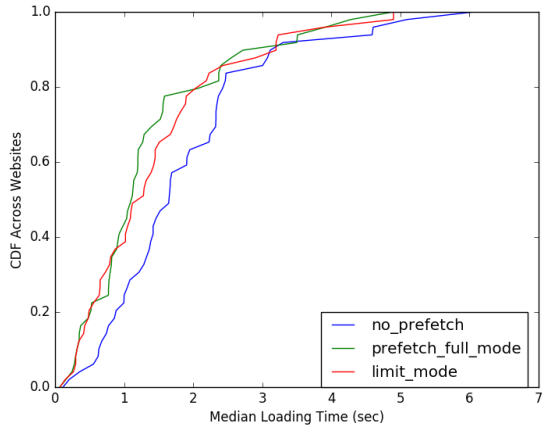

Figure 4: CDFs of median loading times of 50 websites in full mode, limit mode, and non-prefetching mode.

the number of web requests prefetched in the limit mode is less than 50; in the full mode, this fraction is only 40%.

Our finding implies that the cost of mispredictions in the full mode can be much higher than the cost in the limit mode. While the loading times in the limit mode shown in Fig. 3 and Fig. 4 are not significantly less than the times in the full mode, the difference in misprediction costs can be vital. Hence, when the network bandwidth is limited, we suggest using the limit mode instead of the full mode.

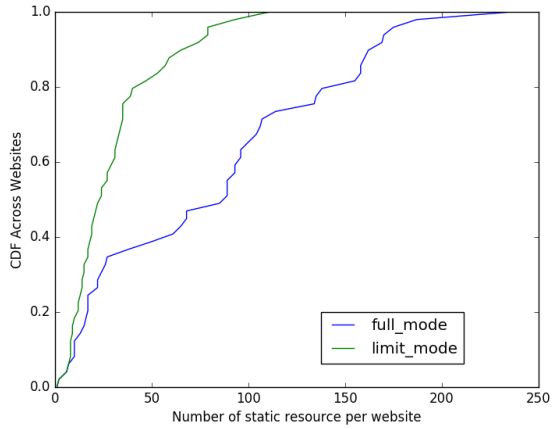

Figure 5: CDFs of the numers of prefetched resources of 50 websites in full mode and limit mode.

5 Related Work

Prefetching content [3, 5] and predicting accurately what contents should be prefetched [1, 4, 6] were previous studied. However, they did not consider overheads of misprediction or limited bandwidth in real time. Some approaches (e.g. [2]) utilizes proxies to maximize low-bandwidth users' utility. One problem is that if proxies are down, then users cannot connect to the internet.

6 Conclusion and Future Work

We develop AccWeb, a web service that performs prefetching to improve user experience of web browsing. AccWeb has a mechanism to predict the URL the user wants to visit when he/she is typing in the browser's address bar, and prefetches the static resources associated with this URL. The user is allowed to select the types of resources to prefetch based on personal preference and network condition. Preliminary experimental results show the effectiveness of AccWeb in reducing page loading times.

There are several directions for further investigation, which we are unable to study due to time limitation. First, it would be great if AccWeb can adjust to network condition automatically, so that it will switch to limit mode or suggest the user to do so when the network bandwidth is limited. Second, designing an accurate prediction algorithm is an interested problem in its own. One might borrow ideas from machine learning, and take into consideration many kinds of user behaviors in the prediction process. Moreover, how to store the data (user browsing history, potential URLs to be prefetched, etc.) efficiently is a problem worth being studied.

References

[1] X. Chen and X. Zhang. A popularity-based prediction model for web prefetching. *Computer*, 36(3):63–70, 2003.

[2] L. Fan, P. Cao, W. Lin, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: potential and performance. In *ACM SIGMETRICS Performance Evaluation Review*, volume 27, pages 178–187. ACM, 1999.

[3] Z. Jiang and L. Kleinrock. Web prefetching in a mobile environment. *IEEE Personal Communications*, 5 (5):25–34, 1998.

[4] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *IEEE transactions on knowledge and data engineering*, 15(5):1155–1169, 2003.

[5] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review*, 26(3):22–36, 1996.

[6] T. Palpanas and A. Mendelzon. *Web prefetching using partial match prediction*. Citeseer, 1998.