CSE-534 - Project Report
HTTP/2 Server Push

Alex Merenstein and Utku Uckun

12/07/2018

Stony Brook University

1) Introduction

HTTP/2 Server Push allows servers to push objects to clients before the client requests the object, potentially speeding up page loads. For example, if the server sends a client a CSS stylesheet and it knows the client will need a font object referenced by the stylesheet, it can push the font object to the client while the client is still parsing the stylesheet. Once the client realizes it needs the font it will check for it in its caches, find it in its push cache, and not need to request it from the server - saving the client from sending a request to the server and waiting for the response. In this case, page load time is decreased by one RTT+time to download the font.

2) Problem Statement

Although Server Push is simple in concept, there are several details and nuances that complicate its use:

1. Pushing objects uses bandwidth that might be better utilized transmitting a higher priority object. Spending bandwidth on an unimportant object at the expense of an object required for rendering the page (e.g. a stylesheet) will actually increase page load time [1].
2. The server doesn't know what objects the client will need every time. Pushing objects the client wasn't going to request is a waste of bandwidth [1].
3. The server doesn't know what objects the client already has. Pushing objects that the client has cached is also a waste of bandwidth [1].
4. Only objects from the original domain (or from a domain that the original domain is authoritative for) can be pushed. Since websites typically pull resources from multiple domains, the actual number of objects that can be pushed may[3].

The fact that website structure and client handling of Server Push [2] both varies considerably means that devising a catch-all strategy for deciding what objects to push is difficult. These difficulties have meant that although HTTP/2 continues to gain traction, adoption Server Push has languished with just a few thousand sites making use of it [9].

3) Approach

There here have been numerous attempts at devising a good strategy for deciding what objects to push, e.g. [1, 3, 10, 11]. However, we did not find a strategy that took into account historical page load data except for strategies that looked for object dependencies or for frequently requested objects.

Our strategy looks at prior page loads, but does so in order to identify points during the page load which are likely to benefit from Server Push. We focus on the objects around these points and identify dependencies between objects that appear to be stalling the page load. The intuition is that a large gap of time (an “RTT gap”, since it will be at least one RTT in length) between object requests indicate places during the page load where the browser had requested an object, spent some time processing the object, and then requested another object as a result. By pushing the second object while the first is being processed, we can save the browser from having to request it from the server. We say that there is a dependency between these two objects, with the first object being the parent in the dependency and the second being the child.

In order to avoid pushing unnecessary objects (which can negatively impact page load times), our approach assumes that many page loads are being used to find dependencies, and includes checks to increase our confidence that the objects being pushed actually should be pushed. We also try to avoid pushing unimportant objects. As seen in Section 2, one of the difficulties in deploying Server Push is that wasting limited bandwidth to push less important objects (e.g. images) ahead of more critical objects (e.g. JavaScript or CSS) can hurt page load times. Thus, our algorithm only generates dependency pairs that include critical objects such as CSS, JS etc. objects.

4) Solution

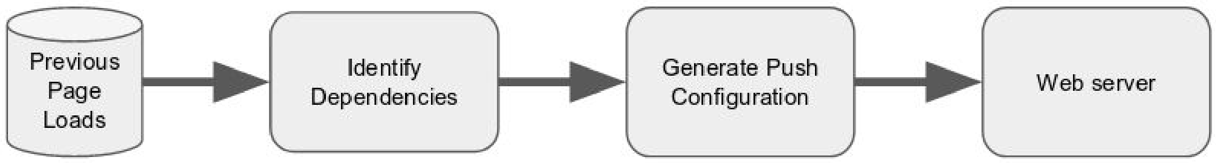

Figure 1: Push configuration generation

We implement our push strategy in a Python script. The script takes HAR [7] files as input, which contain lists of object requests and timings from multiple page loads. The HAR files are produced by the browser (Chromium). The script then identifies RTT gaps and dependencies, eliminates low confidence dependency pairs, and outputs a push configuration file. This process is depicted in Figure 1.

Since we have no way of knowing the actual RTT for the connections, we estimate it using the time between the first two requests. When looking for other RTT gaps we use ½ of this initial gap time length, to try and ensure we continue to find all RTT gaps even if the RTT decreases during the page load.

As mentioned in Section 3, we have checks that try to eliminate false dependencies and increase our confidence in the final push configuration. We require that the dependency exist in at least 80% of the previous page loads. Additionally, if the child of a dependency is requested before the parent in any of the previous page loads we eliminate that dependency pair. Finally, we remove any dependencies where the child is not an “important” filetype. The goal is to prevent less critical objects like images from blocking objects that are absolutely required to render the page, such as JavaScript or CSS objects.

4) Evaluation

We use Mahimahi [3], a suite of tools from MIT which records HTTP requests and responses and then allows replaying future requests with the stored responses. Mahimahi runs a webserver (h2o) for each domain that had served content during the original page load, and uses Linux namespaces to isolate the webservers from regular Internet traffic. Each namespace is given the same IP address as the webserver has on the Internet and dnsmasq provides a local DNS server for the namespaces, so links to resources required by the web page (e.g. images or stylesheets) do not need to be changed.

Since the original version of Mahimahi does not support HTTP/2, we use an updated version from [3] which adds support for HTTP/2.

To evaluate our push strategy, we compare it with three other strategies: a baseline “no push” strategy, plus two strategies from [3]: “push all” (all possible objects pushed) and “push critical” (files the authors found to be important for displaying the “above the fold” portion of the web page). We use a testing script from [3] to run the tests, which uses Browsertime [6] to replay page loads using different push configurations. The recordings used for replaying page loads were retrieved from [5]. We use Chromium, version 70.0.3538.77, running on Ubuntu 16.04.

We use SpeedIndex [12] to evaluate the performance of different web pages instead of page load times. SpeedIndex attempts to measure when a person would believe that the page is loaded, which provides a more useful metric than other page load metrics such as when the onLoad browser event fires.

5) Results

You can find our results in Table 1 and in Figure 2 (lower scores better). From the results we can see that our algorithm was better than no push methodology for three websites, and in the two instances where it was worse it was worse by only a couple percent. When comparing to the “push critical” strategy, we found that although it out performed us in most cases, we were generally close in performance.

	no push	our strategy	% change
aliexpress	2260.0	2300.0	1.77
apple	2059.0	1713.0	-16.8
chase	3457.0	3535.0	2.3
craigslist	631.0	622.0	-1.42
wikipedia	3130.0	1285.0	-58.9

Table 1: SpeedIndex results from replaying websites without pushing and with pushing using our strategy

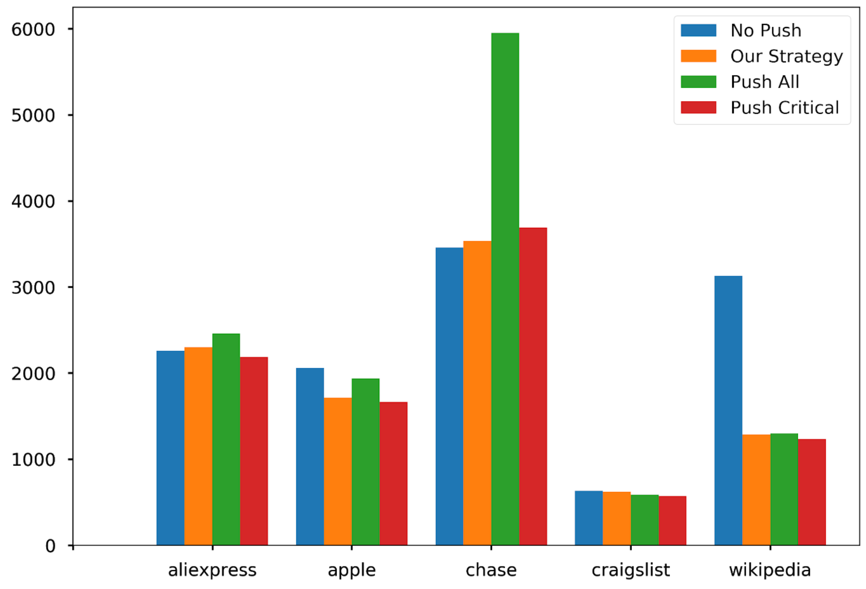

Figure 2: SpeedIndex results for given websites using different push methodologies

One of the interesting findings from these results is the big performance improvement pushing strategies provide to Wikipedia. Through trying different permutations of the push configurations, we found that there was one specific CSS file being pushed that was responsible for most of the 58% speedup. This is further evidence of the fickleness of Server Push

Another interesting case is Chase, since it illustrates the dangers of pushing too much. The “push all” configuration pushes 112 objects, compared to 16 for “push critical” and 29 for our strategy. Second interesting information we extract from these results is the case of web page chase. We can see that all push methodologies degrade the performance of page load time of the page. Again, we examine the HTML file of chase to understand the structure of the web page and we found out that chase web page has many script objects (CSS, JS etc.). We believe that this is the cause of worst performance with server pushing.

6) Future Work

Improvements to Our Push Strategy

Although our push strategy works well in some cases, there are some aspects that should still be investigated:

- Currently we find the gap between the first object (HTML file) and second object and use half of this value as our gap lower threshold. If the time difference between any two requests is larger than this threshold we treat it as an RTT gap. However, this value may be overly conservative (leading to missed opportunities to identify and push a dependency) or overly permissive (causing false dependencies to be identified and potentially pushed, wasting bandwidth). We believe it is worth investigating to determine a more systematic method of estimating the RTT gap length.
- We currently do not push any images. However, there may be cases where it is appropriate to push images, for example if the image is displayed prominently above the fold of the page and can be displayed before the browser finishes processing all of the CSS and JavaScript objects. We think it would be worthwhile to devise a more fine grained approach for deciding what objects are important enough to be pushed.
- The first “Rule of Thumb”[8] for Server Push is “Push just enough resources to fill idle network time, and no more.” Currently we don’t take into consideration the

size of the objects being pushed or how many bytes would be appropriate to push.
- Currently we use HAR files from the browser to know the ordering of object requests. Since the HAR file is built by the browser it represents the “ground truth” of what requests were made by the browser, and when. Using HAR files as input to our push strategy is problematic though if we want to be able to run the algorithm on the server with real, live page loads. We would like to investigate whether we can use the object requests as seen by the server as an input to our push strategy. One challenge that we forese is that the browser may make requests to third party sites that would be unseen by the original server and could lead to the server falsely identifying RTT gaps.

Client Specific Pushing

To the best of our knowledge, all current push strategies push the same objects to all clients. Moreover, most prior research uses Chrome/Chromium exclusively for testing their push strategy. We suspect that differences in how browsers construct the DOM and execute JavaScript will lead to differences in object request ordering and therefore the optimal push ordering.

Learning Push Configurations

Despite the large amount of work on the topic, there is still no singular push strategy that yields a good improvement for all websites. We suspect that due to the diversity of websites such a strategy will not exist.

Instead, we concur with the suggestion in [3] that perhaps servers could learn better push strategies. We believe that servers could find better push configurations by adjusting on the fly to live traffic, trying to push different objects and testing different permutations of push orders. This approach would enable the server to find the best push configuration for that specific site. An additional benefit would be that the server could automatically adjust its push configuration if the site changed and a new configuration became better.

7) Conclusion

In summary, we have developed a new push strategy, based on identifying “RTT gaps” in object lists from previous page loads. Unlike other strategies, ours focuses on portions of the page load which we believe present the most opportunity for speedup with Server Push. We implemented our strategy and used a record and replay framework (Mahimahi) to test our push strategy compared with a baseline (no pushing) and two strategies from a recently presented paper [3]. We found that our strategy outperformed the baseline in three out of five test cases and performed comparably to the other two strategies. Our implementation of our push strategy is available online[1].

Our conclusion from this research is that using object request time information to extract object dependency information is a reasonable strategy. Furthermore, our results suggest that pushing less is a good rule-of-thumb in order to not worsen the page load times compared to no push strategy. Therefore, we believe that our minimalistic solution is a step in the correct direction.

[1] https://github.com/pzoxiuv/http2-conext-push

REFERENCES

[1] Kyriakos Zarifis, Mark Holland, Manish Jain, Ethan Katz-Bassett, and Ramesh Govindan. 2017. *Making Effective Use of HTTP/2 Server Push in Content Delivery Networks*. Technical Report. University of Southern California

[2] Jake Archibald. HTTP/2 push is tougher than I thought. https://jakearchibald.com/2017/h2-push-tougher-than-i-thought/. Online 05/30/2017

[3] Torsten Zimmermann, Benedikt Wolters, Oliver Hohlfeld, Klaus Wehrle. 2018. Is the Web ready for HTTP/2 Server Push?.

[4] Ravi Netravali, Anirudh Sivaraman, Greg D. Hill, and Keith Winstein. Mahimahi. http://mahimahi.mit.edu/. Online 12/15/2015

[5] COMSYS. https://github.com/COMSYS/http2-conext-push. Online 12/1/2018

[6] Browsertime. https://www.sitespeed.io/documentation/browsertime/. Online 12/5/2018

[7] Jan Odvarko. http://www.softwareishard.com/blog/har-12-spec/. Online 12/7/2018.

[8] Tom Bergan, Simon Pelchat, Michael Buettner. 2016. *Rules of Thumb for HTTP/2 Push.* https://docs.google.com/document/d/1K0NykTXBbbbTlv60t5MyJvXjqKGsCVNYHyLEXIxYMv0/edit#

[9] COMSYS. https://http2.netray.io/stats.html. Online 12/7/2018

[10] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2014. How Speedy is SPDY?. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, Berkeley, CA, USA, 387-399.

[11] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V. Madhyastha, and Vyas Sekar. 2015. *KLOTSKI: Reprioritizing Web Content to Improve User Experience on Mobile Devices.* University of California, Riverside.

[12] Google. https://developers.google.com/web/tools/lighthouse/audits/speed-index. Online 8/21/2018