Proxy-based prefetching and pushing of web resources

Proxy-baserad prefetching och pushing av web resurser

Jacob Holm

Supervisor: Vengatanathan Krishnamoorthi (Linköping University)
Marcus Cromnow (Opera)
Examiner: Niklas Carlsson
Upphovsrätt


Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: [http://www.ep.liu.se/](http://www.ep.liu.se/)

© Jacob Holm
Abstract

The use of WWW is more prevalent now than ever. Latency has a significant impact on the WWW, with higher latencies causing longer loading time of webpages. On the other hand, if we can lower the latency, we will lower the loading time of a webpage. Latencies are often caused by data traveling long distances or through gateways that add additional processing delays to the forwarded packets. In this thesis we evaluate the latency benefits of different algorithms for prefetching and pushing of web resources, from a proxy when the client cache is known. We found that the most beneficial algorithm is a two sequence data mining technique. This algorithm is evaluated on a live system where we improve loading time by approximately 246 ms with only a 27% traffic increase on average. The results were measured by evaluating a large set of clients on Opera Turbo 2, a distributed proxy with knowledge of the client’s cache. We also concluded that by using a more conservative strategy we can push prefetched resources to the client, reducing the client requests by approximately 9.3% without any significant traffic increase between proxy and client.
I want to give my thanks to everyone at Opera for being great colleagues, providing a lot of helpful ideas and support, during my time at the office. A special thanks to my supervisor at Opera, Marcus Cromnow, for the extra attention given to me during this time and making the thesis possible. I also would like to thank Magnus Gasslander, from Opera, for being part of the startup process of this thesis. I have been given great comments, valuable discussions and a lot of guidance from my supervisor at the university, Vengatanathan Krishnamoorthi, I am really grateful for having you as my supervisor. My final thanks goes to my Niklas Carlsson for being a great examiner, providing a lot of insightful ideas and help throughout the work of this thesis.

Linköping, June 2016
Jacob Holm
Contents

Abstract iii
Acknowledgments iv
Contents v
List of Figures vii
List of Tables vii

1 Introduction 2
1.1 Motivation ................................................. 2
1.2 Aim .................................................. 4
1.3 Research questions ...................................... 4
1.4 Limitations ............................................. 5

2 Theory: Background and Related Work 7
2.1 HTTP .................................................. 7
2.2 Caching ............................................... 7
2.3 Prefetching ........................................... 8
2.4 Pushing ............................................... 15
2.5 Metrics ............................................... 15
2.6 Evaluation approaches .................................. 18

3 Methodology 21
3.1 System architecture .................................... 21
3.2 Prefetching techniques ................................ 22
3.3 Scenarios ............................................ 23
3.4 Data gathering ........................................ 23

4 Results 25
4.1 Implementation ........................................ 25
4.2 Data gathering ........................................ 28
4.3 Prefetch measurement results ........................ 30
4.4 Push measurement results ............................. 33

5 Discussion 36
5.1 Choosing a prefetching technique .................... 36
5.2 Performance results ................................... 38
5.3 Methodology .......................................... 42
5.4 The work in a wider context .......................... 44
5.5 Ethical and societal aspects ........................... 45

6 Conclusion 46
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Internal and external latency in a proxy environment. The figure is adopted from the similar figure by Kroeger et al. [14]</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>The system architecture of Opera Turbo 2</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>UML diagram of the two sequence implementation</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Latency Per Page during a 1 day run time</td>
<td>30</td>
</tr>
<tr>
<td>4.3</td>
<td>Traffic Increase of prefetching during a 1 day run time</td>
<td>31</td>
</tr>
<tr>
<td>4.4</td>
<td>Recall during a 1 day run time</td>
<td>31</td>
</tr>
<tr>
<td>4.5</td>
<td>Byte Recall during a 1 day run time</td>
<td>32</td>
</tr>
<tr>
<td>4.6</td>
<td>Server Load Ratio during a 1 day run time</td>
<td>33</td>
</tr>
<tr>
<td>4.7</td>
<td>Request Savings during a 1 day run time with 5 days of training data</td>
<td>34</td>
</tr>
<tr>
<td>4.8</td>
<td>Miss Rate Ratio during a 1 day run time with 5 days of training data</td>
<td>35</td>
</tr>
<tr>
<td>4.9</td>
<td>Traffic Increase of push during a 1 day run time with 5 days of training data</td>
<td>35</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Specification of the Opera Turbo 2 node used when performing the measurements</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Prefetching technique evaluation scenarios</td>
<td>23</td>
</tr>
<tr>
<td>3.3</td>
<td>Summary of required metrics for data gathering</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Trainer log mining attributes</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>Push strategies</td>
<td>34</td>
</tr>
</tbody>
</table>
1 Introduction

The use of World Wide Web (WWW) is more prevalent now than ever. Perhaps the biggest factor when measuring WWW browsing speed is latency. The latency occurs as data travels through long distances and gateway queuing delays. It is common that web documents have unpredictable and long delays [1]. Since the demand on the WWW is only getting higher and higher, the traffic increases. Although the impact of the increase in traffic on latency could be lowered by increasing the bandwidth, it is not a long term solution as it would encourage the users to create more network usage hungry applications [2]. Therefore, this can be considered a temporary fix since we will have to increase the bandwidth in the near future once again. A better approach would be to provide some improvements to the systems that are being used. Higher latency causes longer loading time for webpages. If we can lower the latency, we will also lower the loading time of a webpage. This is where prefetching [3] [4] and pushing [5] becomes relevant, since they try to help improve the latency on the WWW without requiring any additional infrastructure.

1.1 Motivation

The work described in this thesis has been conducted on and for Opera, a company that is mainly developing browsers. One of Opera’s products is Opera Turbo 2 which is a distributed proxy to which e.g. Opera browsers can forward all their requests. The main benefits of Opera Turbo 2 is that it reduces request latency and have great data compression. Opera Turbo 2 serves connections from million unique IP addresses every day. Due to the large amount of requests the proxy handles it would be beneficial to prefetch and push web resources, such as images, from the proxy to its clients whenever it is possible. This is
partly being done at the moment, by scanning the responses for dependencies such as links to CSS, JavaScripts and images. But there is many potential prefetches which is not identified by the response scanning and it is therefore desirable to complement the scanning with some other technique for prefetching and pushing of web resources.

Another great benefit of Opera Turbo 2 is that it has knowledge of the local cache of all the clients. This is a fairly uncommon feature that should impact prefetching in a good way, by lowering the number of bad prefetches.

During the request of a website the client is waiting for responses most of the time, meaning it is spending a lot of time being idle. When a response is received, additional requests might be needed in order to load the complete webpage. Web prefetching can be used to increase the amount of work the client will have while waiting for responses [6]. But it is hard to predict what to prefetch before knowing what is needed. Some prediction can be done if we have accessed the page before, knowing what was requested at that time. But chances are that it is already in the client’s cache. In the ideal case we would have a so called "perfect predictor" meaning that all the predictions would be correct. This is not how it works in the real world, where there is still a lot of improvements that can be done [7]. To have better prediction we can let a proxy predict the required web resources, which means the request is forwarded to a proxy that will handle it (forward to destination) and send the response to the client.

A proxy stores access logs from requests which can be used to collect access frequency of a lot of users [8], meaning the proxy will have a lot more knowledge about what resources is required by a request, than that of a single client, which can give a better prediction of what resources to prefetch [9]. The prediction and pre-collecting of resources is called prefetching which is, in a more general context, based on prediction of the cache. When prefetching is done on a proxy the resource will be loaded into the cache waiting for a client to request the resource [10]. Due to better prediction, a proxy will have less incorrectly prefetched resources than that of a single client. It might even be beneficial to push prefetched resources directly to the client [5], if predictions are accurate enough.

By pushing unnecessary data to the client we increase the required bandwidth. This can be costly since it increases the network traffic, with the costs and effects particularly noticeable in mobile networks [11]. The resources prefetched by the proxy should therefore not be pushed right away. If the prediction have high enough confidence, such that we are almost certain the resource is required
by the client, it will get pushed. This way most of the uncertain prefetched resources are filtered, lowering the amount of unnecessarily pushed resources to the client. There is also a possibility that the prediction is very good but the client already have the resource in its local cache. This is equally bad as pushing a wrongly predicted resource since it will not be needed by the client, meaning it will be an unnecessary pushed resource. This can be avoided by communication between the client and the proxy where the client sends information about its current state of the cache, meaning that the proxy will know how the client cache looks like at all times.

Caching is a historically great approach to reduce latency for static web documents. But nowadays the WWW is going towards more dynamic web documents, where the content changes over time [12]. This has negative impacts on caching since even though a popular document is cached it will be considered a cache miss because it is not considered being up to date [13]. This makes prefetching even more important since more dynamic documents will cause more cache misses and for every cache miss there could be a possible prefetch. In an old but still accurate study about latency reduction using prefetching and caching, Kroeger et al. claim that the latency reduction of a proxy with caching can be increased by a factor two when using prefetching [14] and have been evaluated in a real scenario [7]. The maximum cache hit rate that any caching algorithm can achieve is normally not more than 40-50%, which was concluded in a paper about caching proxy limitations [15]. Despite increased usage of dynamic contents these hit ratios have been relatively stable over the last decades [16] and across geographic locations [17]. By using prefetching on a proxy, latency reduction can be lowered by 67% which is more than when used on both the client (35%) and the server (54%) respectively, concluded in a paper about the limits of the web prefetching architecture by Domenech et al. [18].

1.2 Aim

In this thesis we aim to lower the total time it takes to load web pages by using prefetching and pushing. We will analyze the most suited algorithms for prediction of resources to prefetch and push in a proxy where we know the local cache of all the clients.

1.3 Research questions

From the explanation of what we aimed to do, we can extract some research questions. We want to know which existing techniques work best for prefetching and pushing of resources on a proxy where the client’s cache is known. This feature is quite unique since many systems do not allow the proxy to know any
of the clients’ caches. Therefore, research has to be done on what existing techniques there are and which of those that are theoretically suitable to be used on a proxy where the clients’ caches are known. We also need to consider that some of the resources are already being prefetched because being identified by the response scanner in Opera Turbo 2. Once we have found some theoretically suitable techniques we are also interested in which of these techniques that performs best in a live system, such as Opera Turbo 2. Our goal is to answer the following research questions.

Q1 Which previously proposed prefetching techniques are most beneficial for lowering the users’ perceived latency on a proxy?

It is very important to choose a prefetching technique with care since the wrong one could result in a lot of traffic increase and slow down the whole system. We therefore find some well-motivated techniques and examine the consequences of the technique when pushing resources from the proxy to the client’s. We assume the proxy know about the clients’ cached contents and consider the "most beneficial" techniques. While this information is beneficial, most existing prefetching techniques typically do not leverage this information.

Q2 How much does a prefetching technique in a live system lower the users’ perceived latency on a proxy, when the clients’ caches are known?

By evaluating the techniques from Q1 with real request data we determine which of these techniques that give the best improvement to the users’ perceived latency on a proxy, when the client’s caches are known.

Q3 Without affecting the network usage significantly, when should a prefetched resource be pushed from proxy to client?

With the best techniques from the evaluation in Q2 using prediction to know what resources to prefetch, we consider how the same type of prediction can be used for deciding which of the resources to push to the client, which made the initial request. To limit the amount of incorrect predictions, we make sure that only the ones that have a high probability are being pushed. To do this we introduce a confidence threshold for which we trust the prediction enough to push the resource. This threshold is to be set manually, requiring the best value to be determined.

1.4 Limitations

In this thesis work we have the unique opportunity to examine how well proxy based prefetching techniques perform when the client’s cache is known. In order to cover as much ground as possible in this thesis, we will not consider server based prefetching algorithms at all. Server based prefetching algorithms requires
1.4. Limitations

some additional support for prefetching on the target server itself, which is not widely supported by web servers. We will therefore cover a larger fraction of the WWW when not using server based prefetching algorithms.

Due to the complexity of the Opera Turbo 2 system we also limit this paper to only consider the prefetching techniques as a complement to the response scanner already implemented in the system.
2 Theory: Background and Related Work

2.1 HTTP

HTTP is short for HyperText Transfer Protocol and it is the most used application protocol in the WWW. It is used for web systems to seamlessly transfer hypertext and have been used by the WWW global information initiative since the early 90s. One important feature of HTTP is the headers included in each transfer, these consist of required information such as the destination and origin of the request, and optional information such as cache headers. HTTP is described more in detail in RFC2616 [19].

There are multiple cache headers in HTTP but the most common ones are max-age and etag. Max-age defines the lifetime of a cached response for which it is valid, when the max-age is exceeded the request is invalidated and must be requested from the origin server once again. Etag is normally a hash of the request which is compared with the hash of the local cached response, if they do not match the cached response is invalidated and must be requested from the origin server once again.

2.2 Caching

The most common technique for reducing the users’ perceived latency is caching, which more specifically it reduces the overall network traffic of systems [20]. The main purpose of this technique is to store the responses of recurring requests closer to the client. Closer can mean that the response is stored on the same machine (the client) which is called browser caching. It can also mean that the request is stored on a proxy from which the client’s request have to pass, this
2.3 Prefetching

is called proxy caching. It can even mean that the request is stored on the web server after it was requested, such that the web server can reuse that cached response and will not have to process the request again. This is called origin server caching. These three type of caches are described more in detail and together with their consequences in Charu et al.’s paper about caching in the WWW [21]. An interesting approach of distributing objects in Content Distribution Networks (CDN) cache hierarchies is introduced in a paper by Narayanan et al. [22] to reduce latency of web pages. The cached responses are prioritized and the most important are kept updated by prefetching new responses before they expire. This differs from typical prefetching, where future requests are preloaded into the cache, instead of keeping the cached responses up to date.

2.2.1 Cache replacement policies

Since the cache is limited there exists a problem of what to keep and what to replace when the cache is full, such policies are called cache replacement policies. The cache replacement policy plays a great role of the cache efficiency [23]. The policies that have been given most attention in research and usage are:

**LRU** Least recently used, where the least recently used objects are replaced.

**LFU** Least frequently used, where the least frequent objects are replaced.

**SIZE** The objects with the largest size are replaced.

**GDS** Greedy dual size, where the objects in the cache are assigned a priority, based on multiple parameters, and the objects with the lowest priorities are replaced with the new ones.

**GDSF** Greedy dual size frequency, which works similar to GDS but also consider the visit frequency together with the priority when objects are replaced.

2.3 Prefetching

Prefetching techniques are used in multiple different areas to reduce latency such as in storage systems [24], hierarchies of CPU memory [25] and systems of the WWW. In this paper we are only focusing on the latter.

Latency is the time it takes from when a request is sent, until a response is replied. When using a proxy we can divide the latency in two different latencies, internal and external latency, which Kroeger et al. [14] did in his paper about web caching and prefetching. Internal latency being the latency caused by the client - proxy communication. External latency being the latency caused by the proxy - web server communication.
2.3. Prefetching

The goal of prefetching is to reduce the User’s Perceived Latency (UPL) in the WWW which is the delay of the client when requesting a web site. It is important to notice that by reducing the UPL we do not necessarily have to lower the latency or the network traffic. Prefetching will generally affect the system resources used and prediction accuracy. A "perfect" prediction accuracy is normally discussed, which is basically that all the predictions are accurate, meaning that maximum system resources will be used. Same way that low prediction accuracy usually means lower amount of system resources are being used [26]. By using prefetching the network traffic is usually increased since we will always make some unnecessary requests, not needed by the client (assuming we will not achieve "perfect" predictions at all time). Prefetching has more advantages than improved latency, it also prevents bandwidth underutilization, which is when we have bandwidth to use but does not use it [2].

Prefetching techniques are classified into three types: client-, proxy- and server-based [27].

Client based

Prefetching is done on the client. It has the advantage of knowing a lot about the current user, based that specific user’s surfing behavior [28]. A big drawback is that the prefetched requests cannot be shared with other clients and requires a lot of bandwidth, which usually limited on a client.

Proxy based

The prefetching is done on a proxy. The main goal of a proxy based prefetching algorithm is to lower the latency between the client and the server. This is done

![Fig. 2.1: Internal and external latency in a proxy environment. The figure is adopted from the similar figure by Kroeger et al. [14]](image-url)
by lowering, or in the best case completely removing, the latency between the proxy and the server [14]. There is two main advantages to use proxy based prefetching [26]. The first being that the bandwidth of a proxy is generally much higher than that of a client. A client does not want the original request to have to compete with the prefetching requests. Having more bandwidth does not completely remove this issue but certainly make it less hard to deal with. The second advantage is that more users will be able to use the successfully predicted resources that were prefetched. Nowadays proxy-based prefetching is given more research than the others [29] [30].

**Server based**

Prediction is done on the server, which will send hints with the initial request of which resources the client should prefetch. Meaning the prefetching is actually done from the client. It is important to notice that the client in this case can be a proxy forwarding the request of an actual client. An advantage is that you get good predictions on the web site due to maximum information about the system and its incoming requests [27]. A disadvantage is that the prefetching is limited to the web servers implementing this, considering real browsing behavior this will only improve the overall user’s perceived latency by a small portion.

### 2.3.1 Prefetching techniques

Prefetching techniques can be classified into two general categories which is proposed in the survey paper about caching and prefetching by Ali et al. [31]:

- **Content-based** where the body of the response is analyzed for dependencies required by the client to fully load the site or resource. Most common is to parse a requested html file for JavaScript-, CSS- and image-links.

- **History-based** where previously made requests are analyzed, by parsing logs, to predict future requests. The different prefetching techniques being used and researched today can be divided into four sub categories. The proposed sub categories were Markov Models, Dependency Graph, Cost Function and Data Mining [31].

Opera Turbo 2 is doing content-based prefetching by lexically analyzing the responses before serving them to the requestor. This way dependencies such as CSS, JavaScripts and images are identified and prefetched. A similar content-based prefetching technique is used in a paper by Xu and I. Ibrahim [32], where they are also searching for links to prefetch by reading through the responses. A perfect content-based prefetching technique would identify the full dependency graph of the requested site. However, there is limitations to nowadays content-based prefetching in terms of revealing the full dependency graph. In a paper by Netravali et al. [33] they conclude that the dependency graph created by lexical analysis of relationships between web resources is very limited.
History-based prefetching has been given a lot of attention in both research and business. The subcategories are presented in the subsections below.

2.3.2 Prefetching based on Markov Model

Markov Models are mostly used in probability theory to model a system whose state changes randomly. The users’ history of access patterns are modelled in a Markov Model. In a k-th order Markov Model each access is modelled as a state where each state depends on K previous states. E.g., in a first-order (1-st order) the next state (access) \( X_k \) depends on the current state \( X_{k-1} \) and in a second-order (2-nd order) the next state (access) \( X_k \) depends on both the current state \( X_{k-1} \) and the previous state \( X_{k-2} \).

One very popular way of using Markov Models in prefetching techniques is by doing Prediction by Partial Matching (PPM). This is done by using high-order Markov models as a prediction tree where all the user’s access history have been modelled. One major issue with PPM, and many other techniques based on high-order Markov models, is that for each accessed page the model grows linearly in size. This is a big issue when using it as a proxy based prefetching technique since it handles a lot of requests. However, if the size of the model is handled, using some method, then PPM should be able to be used as a proxy based prefetching technique. Depending on how good the method that manages the size of the model is at keeping the relevant access information, the technique will be better or worse. There is no characteristics that would suggest that techniques using Markov models should be better or worse when the client cache is known at the proxy.

In a paper by Shi et al. [34] they propose a technique called Integrated Web Prefetching and Caching Model (IWPCM). IWPCM is using first-order PPM, which requires a lot of data to model on a proxy but still less than using high-order PPM. To have a manageable model on a proxy, modifications to the prefetching technique to conserve the data of the model would have to be done.

In another paper by Chen and Xiaodong [35] they propose a popularity-based method for keeping a low size on the Markov model. The key is to assign long branches to popular URLs and smaller branches to the less popular URL’s. This way, together with some other optimizations described in their paper, the model size is well manageable while still getting good enough predictions using PPM. They report that the hit-rate is up to 90% while traffic increase is at most 21%, this is based on logs from year 1998 which means that the results can vary a lot today. Due to the smaller size of the Markov model, this technique should be suitable for proxy based prefetching.
Lastly, in the paper by Fan et al. [36] they propose a simple method for managing the size of the model with a latency reduction of approximately 23% under the assumption that the proxy knows the client’s cache, although this is based on logs from year 1999. The size of the model is limited by ignoring URL’s which has been visited less than T times, limiting the number of past accesses that is used to predict and how many accesses ahead to predict. In the paper they discuss in detail about appropriate values for each of these limitations.

2.3.3 Prefetching based on Dependency Graph

The user’s access information is modelled as Dependency Graphs (DG), where the nodes represent requests and the arcs represent the access that was made, pointing to the next request. Based on how many times a request sequence has been done the arcs are assigned a probability weight value for that request to happen. DG has been used for a long time in the context of prefetching.

In an early paper about predictive prefetching by Padmanabhan and Mogul [37], DG proven to have good capabilities for prefetching when used in a prediction engine, on a separate daemon from where the client can ask for predictions.

In another early paper by Schechter et al. [38], they manage to get 80% of the predictions correct using DG. However, they do not measure the traffic increase at all which is most likely huge. Since the dependency graph only keep track of one access ahead of each request the model does not get very large and therefore storage is not an issue even when used on a major proxy.

DG, being an older technique, can be modified to a Double Dependency Graph (DDG) which is done in the paper by Marquez et al. [11]. Instead of having one type of dependency it is divided into two types. One type for requests within the same web page and the other type for the requests done on other web pages. The results show latency reduction up to 20.20% when evaluated in a mobile environment and with the technique used for proxy based prefetching.

2.3.4 Prefetching based on Cost Function

This is a category for techniques who primarily do prediction based on the popularity of the request. How this is done varies for each technique.

One technique that has influenced a lot of prefetching techniques is the technique proposed in the paper by Markatos et al. [27]. In this paper they propose a top-ten popularity prefetching technique which focuses on keeping an up-to-date cache of a maximum of ten requests on each web server. This is not a general prediction algorithm since it will not be able to predict future requests based
on a single request but rather the access of a web server. The results show that 60% of all future requests are prefetched with a 20% increase in network traffic.

Another popularity based technique is proposed in a paper by Bouras et al. [26]. They analyze the access logs to determine the $N$ most popular requests to be requested. Basically, each request will have a popularity list stored for direct access during prediction. In the evaluation, in the paper, it receives a 73% hit ratio with only 2% increase in network traffic, though it is compared with PPM which gets a hit ratio of 85% with 4% increase in network traffic. While the PPM technique is considered better, the complexity of the $n$ most popular is really low. Lower complexity means it will be easier to implement compared to PPM.

### 2.3.5 Prefetching based on Data Mining

Prefetching based on data mining is divided by two subcategories, which is data mining based on association rules and data mining based on clustering [31]. These are handled in their own section, respectively, below.

#### Association rules

There are two important factors relevant to data mining based on association rules, those are support and confidence. Support is basically the discovery of popular requests while confidence refers to the discovery of association between popular requests.

A technique used for proxy based prefetching is proposed in a paper by Huang and Hsu [39]. By mining for popular two sequence accesses and storing them as probabilities in a rule table, the technique manages to make great predictions. In the paper they get a hit ratio of approximately 70% where 95% of the predictions are correct.

A technique called Ordered web mining ($WM_o$) is proposed by Alexandros et al. [40] [2]. The special about $WM_o$ is that it applies pruning and dynamic adjustment in the order of rules. The results show that hit ratios are increased by approximately 35% with low increase in network traffic, though this is based on a client-server prefetching. When used on a proxy the results will most likely loose hit rate accuracy and increase the network traffic.

In an early paper by Jiang and Kleinrock [41] they propose a technique with focus on how many items to prefetch based on available resources. Nothing more than regular data mining on association rules is used where a rule table with probabilities is created. With this threshold for how many items to prefetch they conclude from the results that up to 70% of the predictions are correct.
Comparison of three data mining techniques were done by Géry and Haddad [42] in their paper about evaluation of web usage mining approaches for request prediction: Association rules (AR), frequent sequences (FS) and frequent generalised sequences (FGS). Each of these mining techniques were used to predict the next sequence of pages. The results show that AR and FGS have the best coverage and accuracy in finding the correct next sequence of pages. This is only done in the context of next page prediction, the prefetching capabilities have not been considered; however, both of the techniques could potentially work well for prefetching from a proxy or server.

N-grams are items a continuous sequence, for the context of this paper, you can consider an item to be something that has been requested on the web (gif, html, etc.). The order which items are received in will form n-grams. E.g. sequences $S_1$, $S_2$, $S_3$ would form the n-grams $S_1 \rightarrow S_2$ and $S_2 \rightarrow S_3$. A technique using access sequence mining with n-grams is proposed by Qiang et al. [8]. They construct a framework using this for prefetching on a server and report results of up to 90% hit ratio at the cost of 70% increased network traffic. Since a proxy will not have access to as much information about a web server as the web server itself we expect the hit ratio to be lower when this framework is used for proxy-based prefetching.

Another technique using n-grams is proposed in a paper by Su et al. [5] where they use a similar approach but this one concludes that n equal to 3 and 4 are reasonable good. But n equal to 3+ is best when a lot of training data is used (which is the case when used on proxy servers). This is based on logs that are at least 16 years old though and a lot have happened to the web in that time.

We get a mobile perspective in the paper by Song and Cao [43] where a technique of association rule based data mining is proposed, called Cache Miss Initiated Prefetch (CIMP). The technique is based on client prefetching and evaluated on mobile devices. Modifications to the technique will have to be done if used on a proxy since mobile specific factors are considered, such as power consumption. The results of CIMP show hit ratio of approximately 80% with almost no network traffic increase.

**Clustering**

ClustPref is a clustering based data mining technique proposed in a paper by George et al. [44] which shows great promise with hit rates of approximately 3-22% higher when used together with respectively cache replacement technique used. The technique uses a web navigational graph which is partitioned by using different association rule mining techniques. This is a proxy-based technique.
A newer technique is to use vector space models to clustered user navigational behavior used in the paper by Wan et al. [45]. In this paper they propose a technique using random indexing (RI) which shows great promise since it is able to find user groups that many other techniques fail to find. When using RI in the context of prefetching results show great promise with precision between approximately 75-80% and recall of approximately 46-53%. This is a fairly new technique in the context of prefetching, which means that not too much study has been done yet. The results are based on server-based prefetching but should work equally well for proxy-based prefetching since both have logs containing a lot of user profiles.

2.4 Pushing

Pushing in the client-proxy context refers the proxy initiated communication between the proxy and the client. The proxy push some information to the client, such as predicted web resources the client would potentially request in the future. When the proxy push the resource it will not have received any request for that specific resource from the client yet nor might ever get it. For push to work, some kind of communication protocol need to be setup that enables the proxy to send data to the client. The most used communication protocol that support push is Google’s SPDY protocol [46]. The primary objective of the SPDY protocol is to improve the speed of the web. This is done by extending the HTTP protocol with three core features described below.

**Header compression** The request and response headers can be compressed which can reduce the bandwidth usage.

**Multiplexing** Requests can be multiplexed over a single TCP connection. This reduce the amount of connections, together with the associated overhead, and also fixes the head-of-line problem where a slow request would block other requests until completion.

**Server push** Described as pushing in this section, the primary goal of pushing is to allow the server to better utilize the bandwidth.

2.5 Metrics

Performance metrics are important to capture the system tradeoffs when using the techniques considered in this thesis. The metrics considered here are based on the findings in a survey by Domènech et al. [47], where they went through the most popular metrics for prefetching and evaluated which of them to use.
2.5. Metrics

2.5.1 Prefetching

Domènech et al. \cite{47} conclude in their survey that studies on performance of prefetching techniques should at least include a latency related metric. To measure the user’s perceived latency it is suggested to use Latency Per Page ($L_p$) as a measurement for latency. We look at the page request time $T_{req}$, when a page request was sent from the client until all the responses of the required resources were completely received. We then look at the time it takes from when the responses were completely received to when they were fully displayed to the user to get a $T_{rend}$ render time. $L_p$ is calculated by adding $T_{req}$ with $T_{rend}$, getting the formula:

$$L_p = T_{req} + T_{rend},$$

where $T_{req}$ is the page request time and $T_{rend}$ is the render time.

Latency is not be the only interesting aspect when evaluating the performance of prefetching techniques. By recommendation from the metric survey by Domènech et al. \cite{47} a Traffic Increase ($TI_{pref}$) metric should be used. This metric is collected to be able to analyze the impact of bad prefetches. By looking at the size of the bad prefetches we can calculate how much bandwidth we "waste". An important factor, which many papers fail to include, is the network overhead related to the additional network traffic caused by the specific prefetching technique used. Some techniques have network overhead, such as sending additional information from the server, and some does not have any network overhead at all. By summarizing the size of all the bad requests caused by prefetching together with the size of the network overhead and size of all requests not caused by prefetching, we can normalize this value using the summation of the size of all requests not caused by prefetching. This is given by the formula:

$$TI_{pref} = \frac{Not_b + OH_b + Req_b}{Req_b},$$

where $Not_b$ is the size of the objects not used, $OH_b$ is the size of the network overhead and $Req_b$ is the size of the object requests not caused by prefetching.

To measure the usefulness of a prefetching technique Domènech et al. \cite{47} propose the metric Recall (Rc). This metric determine the usefulness by looking how much the prefetching contributes over all requests. Calculated by dividing all the good predictions (prefetched responses that were used) over all the responses to the client. This gives us the formula:

$$Rc = \frac{GP}{Resp},$$

where $GP$ are all the good predictions and $Resp$ is all the responses received by a client.
Another metric to measure the usefulness of a prefetching technique is Byte Recall (\(R_{Cb}\)) which was also proposed by Domènech et al. [47] but as a complement to the Recall metric. Why do we want two metrics for measuring the usefulness? Recall is simply a fraction of how much the amount Round Trip Times (RTTs) are lowered which affects the users’ perceived latency. But another factor in users’ perceived latency is the amount of data to be transferred and therefore it is relevant to look on the usefulness in bytes of the prefetching technique. The metric is similar to \(Rc\) but instead of looking at the number predictions and responses we look at the size of them, respectively, more precisely we divide the size of all the good predictions (prefetched responses that were used) over the size of all the responses. This gives us the formula:

\[
R_{Cb} = \frac{GP_b}{Resp_b},
\]

where \(GP_b\) is the size of all the good predictions and \(Resp_b\) is the size of the responses received by a client.

By doing prediction of what resources to prefetch, the load is increased on the server. A common way of measuring load ratio is Server Load Ratio (\(SLR\)), which is proposed by Domènech et al. [47]. The Server Load (\(SL\)) is a measurement of how many requests are handled by the server during a period of time. Server Load Ratio is calculated by dividing the Server Load of when prefetching was used with the Server load of when prefetching was not used, for the same period of time. This gives us the formula:

\[
SLR = \frac{SL_{p_t}}{SL_t},
\]

where \(SL_{p_t}\) is Server Load when prefetching was used during a time period \(t\) and \(SL_t\) is Server Load when prefetching was not used during a time period \(t\).

### 2.5.2 Pushing

Domènech et al. [47] propose a Request Savings (\(RS\)) metric to be used when evaluating the efficiency of push between proxy and client. The goal of this metric is to determine the accuracy of the pushes being sent to the client. The metric is simply a measurement where all cache hits on a client is divided by all the requests from a client. Given by the formula:

\[
RS = \frac{Resp_c}{Req},
\]

where \(Resp_c\) is the number of cached responses used (cache hits) and \(Req\) is the number of requests from a client. In order to determine the accuracy this metric it has to be compared by a reference value. The reference value is normally
calculated from the RS formula but when measured on a system where push is disabled, or a system that logs the amount of "good" pushes such that it can simply be subtracted from the number of cached responses when calculating.

While the RS is a great way to determine the efficiency of push, it does not take the size of the cached responses into consideration. The size have a major impact on the users’ perceived latency, it is therefore relevant to have a metric for measuring the efficiency of push in terms of size. For this purpose Domènech et al. [47] propose that the Miss Rate Ratio (MRR) should be used. This is a metric where you look at the Miss Rate (MR\(_b\)) of when the prefetching is used and divide it with the miss rate when the prefetching is not used. The Miss Rate and Miss Rate Ratio are given by the formulas:

\[
MR_b = \frac{M_b}{R_b},
\]

and

\[
MRR = \frac{MRp_b}{MRn_b},
\]

where \(M_b\) is the size of the responses that were not obtained from the cache, \(R_b\) is the size of all the responses, \(MRp_b\) is the miss rate when the prefetching is used, and \(MRn_b\) is the miss rate when prefetching is not used.

With the same reasoning about Traffic Increase (\(TI_{pref}\)) being a good metric when evaluating prefetch (see section 2.5.1), it is also a good metric for evaluating push. The formula have to be slightly modified to fit the push scenario. We can completely ignore the network overhead since there is no additional traffic between the proxy and the client. Instead of using the size of objects not used that were prefetched to the proxy we use the size of objects not used that were pushed to the client. Which gives us the formula:

\[
TI_{push} = \frac{Not_b + Req_b}{Req_b},
\]

where \(TI_{push}\) is the traffic increase caused by push to the client, \(Not_b\) is the size of the objects that were pushed to the client but not used and \(Req_b\) is the size of the received objects on the client not caused by push.

### 2.6 Evaluation approaches

#### 2.6.1 Trace-driven simulation

The most common way of measuring how well a prefetching technique is performing is by doing trace-driven simulation, often used in both newer (e.g. [7]) and older (e.g. [5]) papers about prefetching. This type of measurement normally takes real logs as input and simulates the requests by feeding from the
logs, you can think of it as a replay of the requests done to a server at a certain time period. If the prefetching technique requires some training/pre-processing, an initial part of the logs can be used. In a paper about prefetching based on N-grams [5], the first 4/5 of the log is used for training and the rest 1/5 for testing. The benefit of performing this type of measurement is that it is relatively easy to do since it only requires access to some relevant access logs and a simulation tool. The simulation tool is either developed for the specific requirements of the experiments or a simulation framework is used, such as the simulator proposed in a paper by Marquez et al. [48], which is a performance evaluation of caching and prefetching techniques. However, the results are still considered theoretical since we will only analyze how well a certain prefetching algorithm would perform based on those access logs. In practice the access logs may look different due to effects of the system such as performance (requests per second) and timing, for which the requests are served and the responses are handled (it basically means that the requests might happen at different times). Abdullah Balamash et al. [49] concludes that by using logs we will not be able to control the traffic parameters which could be of value when studying certain scenarios. This is also observed by Barford and Crovella [50] in a paper where they present SURGE, a tool for generating web workload. The quick summary of this tool is that it creates a model based on request statistics and from this model simulates the run time of a web server, creating a synthetic log. This means that traffic can be controlled by generating synthetic access logs which was done in a paper by Nanopoulos et al. [40] where a model is created using the SURGE tool. If no logs are available Stochastic Petri Nets can be used for performance evaluation of prefetching techniques, which was done in a paper by Shi et al. [34]. A Petri Net is a graph where nodes represent transitions or places and the vertices represents the preconditions to reach transitions or places. Stochastic Petri Nets simply introduces a new precondition, a delay set by a probability based random value [51].

2.6.2 Measurement in a live system

The best possible measurement of a prefetching technique is by evaluation in a live system. The system is being analyzed two times for a specific time period $t$ each, once when the prefetching is turned on and once when the prefetching is turned off. The analysis can be done live, or by parsing some pre-fetch log that was dumped by the system during run-time, on the metrics described in Section 2.5. This would provide information about how well the prefetching algorithm actually performs on the targeted system, which is normally the goal of the prefetch implementation. There still exists issues with this since the measurement will be based on completely separate set of requests, due to the two separate measurements, might impact the results. However, if the time period $p$
is chosen carefully and with high enough time period on the two measurements, both should converge towards accurate average metric values.
A literature survey of different prefetching techniques was conducted, looking for the most beneficial prefetching technique in a proxy-based environment and that could benefit of the proxy’s knowledge about the local cache of its clients. The techniques identified are based on data mining and one that stood out was the two sequence prefetching techniques proposed by Huang and Hsu [39]. This is mainly because it shows great results in evaluation and that the evaluation was performed in a proxy-based environment. In Section 5.1 there is a more detailed discussion of why those techniques were chosen over the others. These techniques use data mining based on association rules to create a rule table which is used to predict the next requests. The measurements have been done on one of the nodes (see Table 3.1 for specification) in Opera Turbo 2 used in production. Meaning the evaluations are be based on live measurements with real network and run-time conditions.

### 3.1 System architecture

The system architecture, illustrated in Figure 3.1, consist of two major parts. The clients which are many different Opera browsers and the servers which are

<table>
<thead>
<tr>
<th>CPU</th>
<th>16 cores (32 with Hyper-Threading) Intel(R) Xeon(R) CPU E5-2650L 1.80GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>66 GB DDR3 RAM</td>
</tr>
<tr>
<td>Operating system</td>
<td>Linux Debian v7.9</td>
</tr>
</tbody>
</table>

Table 3.1: Specification of the Opera Turbo 2 node used when performing the measurements
3.2 Prefetching techniques

To evaluate Opera Turbo 2 using the two sequence prefetching technique they first had to be implemented. The requirement was that they had to be written in the programming language Pike [52], a very uncommon object-oriented language, to be compatible with Opera Turbo 2. There is no Pike library already implementing the two sequence data mining techniques. Therefore, the two sequence mining were implemented from scratch.

The database available for storing prefetching information in Opera Turbo 2 is MySQL [53], which is one of the most used relational databases in the world. Pike have a connector module which could be used in order to communicate with the database, meaning this would not have to be implemented from scratch. The prefetching were also supposed to be non-blocking such that it did not put requests in a blocking state waiting for the database query to finish.
To determine how well a technique performs it should be compared using the same type of evaluation but when the technique is disabled. Evaluation without any prefetching technique is a good point of reference for comparison.

### 3.3 Scenarios

In the paper by Huang and Hsu [39], where the two sequence prefetching technique was proposed, they used three days of training data. However, the proxy which provided the log file handles approximately 14 million requests per week (according to the one week log used in the paper). This means that an Opera Turbo 2 node handles approximately 23 million more requests each day. Because the proxies handles such a big difference in load, measurement is instead done on the set of scenarios listed in Table 3.2. To make sure measurement is done in a fair manner the scenarios did not consider any run time below one day and due to the time limit of this thesis nor any higher. With one day of run time we catch the overall surfing usage, without the risk of only measuring at a bad time during the day, such as night time. Two different sizes of training data is used, one with one day of training data and one with five days of training data. This provides an idea on how sensitive the prefetching technique is to the training data size. To get a good behavior of the prefetching technique, the logs being used should be as recent as possible. E.g. for scenario 1 we would use the logs of the last 24 hours as training data.

<table>
<thead>
<tr>
<th>No.</th>
<th>Run time</th>
<th>Days of training data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 day</td>
<td>1 day</td>
</tr>
<tr>
<td>2</td>
<td>1 day</td>
<td>5 days</td>
</tr>
</tbody>
</table>

Table 3.2: Prefetching technique evaluation scenarios

Additional scenarios for technique specific parameters related to push are added if such parameters exists. These are however specific to the prefetching technique and will not be evaluated on systems not running the prefetching technique. The purpose of the additional scenarios is to make sure that the technique, in terms of push, is evaluated in a fair manner and not performing bad because of poorly chosen parameters.

### 3.4 Data gathering

By logging requests and responses on a proxy, these can be analyzed afterwards. To make sure we only consider requests not being identified by the already implemented response scanner we made sure that this was logged per request. In Section 2.5 the evaluation metrics relevant to prefetching and pushing are presented. For prefetching the interesting evaluation metrics are Latency Per
3.4. Data gathering

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request timestamp</td>
<td>The time when a request was made</td>
</tr>
<tr>
<td>Request time</td>
<td>The total time it took to receive all the resources from a requested site</td>
</tr>
<tr>
<td>Render time</td>
<td>The time it took from all the resources of a site was received, to the time when the site was fully displayed to the user</td>
</tr>
<tr>
<td>Size prefetches not used</td>
<td>The size of all the requests that were prefetched to the proxy but never requested by the client</td>
</tr>
<tr>
<td>Technique network overhead</td>
<td>The network overhead of the prefetching technique</td>
</tr>
<tr>
<td>Size not prefetched</td>
<td>Size of the responses to the proxy not caused by prefetching</td>
</tr>
<tr>
<td>Good predictions</td>
<td>Number of prefetches that were used</td>
</tr>
<tr>
<td>Client responses</td>
<td>Number of responses that were sent to the client</td>
</tr>
<tr>
<td>Size good predictions</td>
<td>The size of all prefetches that were used</td>
</tr>
<tr>
<td>Cache hits</td>
<td>Number of times cached responses were used on a client</td>
</tr>
<tr>
<td>Client requests</td>
<td>Number of requests that were sent from a client to the proxy</td>
</tr>
<tr>
<td>Size cache miss</td>
<td>Size of the responses that were not obtained from the cache of a client</td>
</tr>
<tr>
<td>Size all client responses</td>
<td>Size of all responses to the clients including the ones that were obtained from the clients’ caches</td>
</tr>
<tr>
<td>Size push not used</td>
<td>Size of all the pushes that were never used by a client</td>
</tr>
<tr>
<td>Size requests</td>
<td>Size of all the responses that were received by a client but not caused by push</td>
</tr>
</tbody>
</table>

Table 3.3: Summary of required metrics for data gathering

Page (\(L_p\)), Traffic Increase caused by prefetching (\(TI_{pref}\)), Recall (\(Rc\)) and Byte Recall (\(Rc_b\)). For push the interesting evaluation metrics are Request Savings (\(RS\)), Miss Rate Ratio (\(MMR\)) and Traffic Increase caused by push (\(TI_{push}\)). A complete list of all the metrics required to calculate these evaluation metrics is given in Table 3.3. After the two sequence technique is implemented, the creation of a logging module from where we could report all of the required metrics is done. The logs should be saved to file from which metrics can be extracted after running the scenarios in Table 3.2. To extract the metrics from log files some analysis tool can be created to obtain the metrics easier and input them to a spreadsheet. In the spreadsheet the results can easily be analyzed by creating graphs for comparison, from which the results are produced.
4 Results

4.1 Implementation

A prefetching package was implemented to be able to switch between prefetching implementations with minimal changes. The class hierarchy can be found in Figure 4.1.

Prefetcher

An interface for which different types of prefetches may be implemented. Such as a prefetcher with pre-processing requirement. The only requirement of this interface is that the "getPrefetches" method should be implemented. Which basically takes a URL as input and output what URLs to prefetch.

PrefetchPreprocess

A class implementing the Prefetcher interface. It does so by using a predictor to get the Prediction results. To not flood the servers we only allow a maximum of 10 predictions which is provided as an argument to the "getPrefetches" method. The prediction results are processed and the ones above a certain push confidence threshold are marked for push to the client. Meaning once received by the proxy they are directly pushed to the client. The URL of all the predictions are returned by the "getPrefetches" method. We also have a method to perform pre-processing, the idea is that only prefetching techniques that require pre-processing will use this class. This method simply forwards the file names to the train method of the assigned PrefetchTrainer class.
4.1. Implementation

PrefetchTrainer

An interface for which different types of trainers may be implemented. The only method required is train() which gets the filename of the input files to be used for training, such as logs.

TwoSequencePrefetchTrainer

A class implementing the PrefetchTrainer interface. It stores a storage instance which can be used for storage communication. The "train" method required by the PrefetchTrainer interface is implemented such that it is mining the logs the files provided and extract values for the attributes provided in Table 4.1. The user id is required to identify a user request sequence, as well as the main request URL and the requested URL. Mobile and language is encoded on both of the urls, since we only want to link the requests related to same language and display format (mobile/desktop). If we do not do this, imagine English speaking desktop users getting mobile content in Chinese. That would most likely not be appreciated. When data extraction from the provided log files has finished, two sequences of requests are identified. All unique two sequences are discarded which is also done by Huang and Hsu [39] in their evaluation where the support (minimum occurrences of a two sequence) is set to two. The two sequences

Figure 4.1: UML diagram of the two sequence implementation
are translated into rules and their local and global confidence are calculated. The local confidence is simply the confidence that the rule is accurate given a specific user id (it is local to the user). The global confidence is the overall confidence of the rule not provided a specific user id. In the paper by Huang and Hsu [39] they suggest that for best results a minimum confidence threshold of 0.01 should be used. The rules with global confidence below 0.01 are therefore discarded. The remaining rules are inserted into storage and are later used by the TwoSequencePrefetchPredictor to predict what URLs to prefetch.

**PrefetchPredictor**

An interface for which different types of predictors may be implemented. The only method required is "predict", which takes the URL to predict prefetches and the maximum number of predictions allowed as arguments.

**TwoSequencePrefetchPredictor**

A class implementing the PrefetchPredictor interface. It stores a storage instance which can be used for storage communication. This is the class that does the two sequence prediction described in the paper by Huang and Hsu [39]. It uses the storage instance to query for the rules associated to a main request URL. The results are processed and the rules with highest confidence are transformed into predictions. The maximum predictions returned is specified by the provided "max predictions" argument.

**Storage**

An interface for which different types of storages may be implemented. The methods are "addRule" "findRule" and "clear" which are all required for the two sequence technique to work properly.

**MySQLStorage**

A class implementing the Storage interface. To add a rule to the SQL database "addRule" method is used, to find the rules with "firstURL" set to a specific URL in the SQL database "findRules" method is used and to clear all rules from the database the clear method is used.

**Rule**

A class containing information to be stored in a storage. Rules are created by a trainer and inserted into the storage and can later be queried by a predictor.

**Prediction**

A class containing information about a prediction and its confidence. The confidence is used to determine if the response, after being prefetched, should be pushed or not.
4.2 Data gathering

The data was gathered by logging to two log files. One where loading time of sites were stored and the other for rest of the data.

4.2.1 Loading time log

The loading time is calculated from when the site request was sent from the client to when the site was fully displayed. That means from the time when all of the requested resources for that site have been delivered and fully processed e.g. by rendering. This is done on the client and notice that this is actually the users’ perceived latency being measured. This means that it represents the evaluation metric Latency Per Page ($L_p$), since it is equal to the request time plus the render time for an entire site. The loading time logs are stored on the proxy. Each client communicates to the proxy when a site has been fully loaded, allowing the proxy to log the loading times of all the clients.

4.2.2 Main log

In the main log all of the metrics from Table 3.3 are logged, except for the request time and the render time since those are already taken care of in the loading time log.

Request timestamp

When a request was received the timestamp was logged.

Size prefetches not used

If the prefetch was not used after one minute the size of that response was logged.

Technique network overhead

No additional network overhead were caused by the two sequence technique, therefore this metric was not logged.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User id</td>
<td>string</td>
<td>The unique id of a user</td>
</tr>
<tr>
<td>Main request URL</td>
<td>string</td>
<td>Url of the main request, also known as referer</td>
</tr>
<tr>
<td>Requested URL</td>
<td>string</td>
<td>The URL that was requested</td>
</tr>
<tr>
<td>Mobile</td>
<td>bool</td>
<td>True if the request was done from a mobile, else false</td>
</tr>
<tr>
<td>Language</td>
<td>string</td>
<td>The language header of the response</td>
</tr>
<tr>
<td>Status</td>
<td>int</td>
<td>Status code of the response</td>
</tr>
</tbody>
</table>

Table 4.1: Trainer log mining attributes
4.2. Data gathering

Size not prefetched
When a response was sent and not marked as a prefetch, this metric was logged.

Good predictions
When the proxy received a request that was a cache hit which had been prefetched or when a pushed request was used on the client (if push had been turned off the client would have requested the resource). The client reports any invalid pushes to the proxy, which means we can simply subtract all pushes with all invalid pushes to get the amount of good pushes.

Client responses
When a response was sent this metric was logged.

Size good predictions
Same as for Good predictions but instead of looking at the amount of occurrences we simply looked at the size of the response.

Cache hits
When a request was already in the client’s cache this was communicated to the proxy for logging.

Client requests
When the request was received on the proxy this was logged.

Size cache miss
When a response was sent to a client from the proxy, the size of the response was logged. This would automatically only include client cache misses since it is observed from the proxy.

Size all client responses
The size cache miss metric was already being logged, meaning we only had to supply information to the proxy about the size of cache hit responses. This was communicated to the proxy every time a client cache hit occurred.

Size push not used
If a push was received by a client, it was marked as push and a timeout for five minutes was set. If it have not been used within that timeout period it was marked as invalid and the size was communicated to the proxy.
4.3 Prefetch measurement results

In this section we present the results from a full day when using one out of three different implementations.

- **CBP** The Content-Based Prefetching used in Opera Turbo 2, by scanning responses for dependencies.
- **CBP+TS (1DT)** The Two Sequence prefetching technique used in this paper together with CBP, when used with a single day of training data.
- **CBP+TS (5DT)** Same as CBP+TS (1DT), but with five days of training data.

Figure 4.2 shows the average Latency Per Page ($L_p$), when using each of the three techniques. Note that the Latency Per Page ($L_p$) of CBP with TS is lowered by 246 ms, using five days of training data. We also see that the $L_p$ of CBO with TS is increased by approximately 99 ms, when using 1 day of training data. To ensure fair measurement of $L_p$ based on prefetching, push was disabled since this would have affected the results.

Figure 4.3 shows 27% increase in traffic for TS together with CBP, both when using one day and five days of training data. We also see that the Traffic
4.3. Prefetch measurement results

Increase of the content-based prefetching (CBP) is close to nothing, to be exact the traffic is increased by only 0.005%. Since the traffic increase is close to nothing we know that there is not a lot of bad prefetches being done. This suggest that CBP is very accurate or do very limited amount of prefetches, such that it do not affect the traffic significantly.
4.3. Prefetch measurement results

Figure 4.4 shows 0.12 Recall for TS together with CBP, both when using one day and five days of training data. We also see that when only using CBP the Recall is close to zero, to be exact 0.001, which means that the usefulness of CBP is very limited. However, the usefulness of the predictions were approximately 12% better when using CBP with TS compared to CBP without TS, both when using one day and five days of training data.

Figure 4.5 shows 0.43 Byte Recall for CBP with five days of training data and 0.4 for one day of training data. We also see that when only using CBP the Byte Recall is close to zero, to be exact 0.0001, which means that the usefulness of CBP very limited. We can see that the Byte Recall is increased by approximately 0.43 when using CBT with TS (five days of training data), compared to using CBT without TS. TS with one day of training data have a slightly lower increase of approximately 0.4, compared to using CBT without TS.

Figure 4.6 shows a Server Load Ratio of 1.0 when using CBT which is expected since this is our point of reference. We also see Server Load Ratio’s of 0.43 and 0.24 when using CBT with TS using one respectively five days of training. This means that the amount of requests handled is decreased by more than half when using TS with one day of training. When using TS with five days of training approximately a quarter of the requests are being handled compared to CBT without TS. Based on these results we conclude that the prediction of the two sequence prefetching technique impacts the server load significantly. We see that by using the two sequence prefetching technique we decrease the
amount of requests handled by the proxy by approximately 57% and 76% when using one day and five days of training data respectively.

### 4.4 Push measurement results

For using push with the two sequence prefetching technique we introduced four push strategies shown in Table 4.2, with different values of minimum push confidence, where $0 \leq \text{Push Confidence} \leq 1$. The different values for minimum push confidence were observed by looking at the global confidence of the rule table, created by the trainer of the two sequence prefetching technique. The average global confidence after the rule table had been generated was approximately 0.04. While the optimal minimum push confidence might not be any of these values, it was our belief that the values would tell us in which range the optimal minimum push confidence value within, meaning it would give us a lower and upper bound. We measured and got results for the lower and upper bound, this gave us a range for the results of the optimal minimum push confidence value.

Each of the strategies were run for one day and with five days of training data for the two sequence prefetching technique.

The Request Savings metric provide information about the efficiency of push between proxy and client, where a high value is more efficient than a low value. Figure 4.7 shows 0.086 Request Savings for the None strategy, 0.093 for the VCon strategy, 0.088 for the Con strategy, 0.88 for the Med strategy and 0.90 for the Agg strategy. We can see that the None strategy has lowest Request Savings, while all of the other strategies have higher Request Savings by a mag-
4.4. Push measurement results

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Minimum push confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1.00</td>
</tr>
<tr>
<td>Very Conservative (VCon)</td>
<td>0.8</td>
</tr>
<tr>
<td>Conservative (Con)</td>
<td>0.10</td>
</tr>
<tr>
<td>Medium (Med)</td>
<td>0.04</td>
</tr>
<tr>
<td>Aggressive (Agg)</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 4.2: Push strategies

Figure 4.7: Request Savings during a 1 day run time with 5 days of training data

...since the None strategy is when we do not push any resources that was prefetched by the two sequence prefetching technique, we see that by allowing push we can increase the Request Savings by 0.007. The Very Conservative strategy have the highest Request Savings of 0.093, which means we have a push efficiency maximum of approximately 9.3%. We can also observe that the lowest Request Savings, when push is enabled, is approximately 0.088 when using the Conservative strategy. Making the observed efficiency range, when push was enabled, between 8.8% and 9.3%.

Figure 4.8 shows 1.0000 Miss Rate Ratio for the None strategy, 0.9984 for the VCon strategy, 1.0058 for the Con strategy, 1.0060 for the Med strategy and 1.0031 for the Agg strategy. The Miss Rate Ratio of the None strategy is expected since this is the point of reference. Miss Rate Ratio complements the Request Savings metric, instead of looking at the number of good pushes it consider the size of the bad pushes. A low value of the Miss Rate Ratio is more efficient than a high value. We observe that the lowest Miss Rate Ratio is approximately 0.998 when using Very Conservative strategy. This is the only strategy that improves the Miss Rate Ratio since the rest are larger than one, with a maximum of 1.0060 using Medium strategy.
Figure 4.8: Miss Rate Ratio during a 1 day run time with 5 days of training data

Figure 4.9: Traffic Increase of push during a 1 day run time with 5 days of training data

Figure 4.9 shows 1.0033 Traffic Increase for None Strategy, 1.0031 for VCon strategy, 1.0025 for Con strategy, 1.0030 for Med strategy and 1.0027 for the Agg strategy. We observe that difference in the traffic increase between the different strategies is low. The largest traffic increase is when using None strategy, with an increase of 0.33%, while the lowest is when using Conservative strategy with an increase of 0.25%. Giving us a traffic increase range between 0.25% and 0.33%, which is very low.
5 Discussion

5.1 Choosing a prefetching technique

After reading the method, you might still wonder why the two sequence prefetching technique were chosen. Like in many papers, a literature review was done to identify the different prefetching techniques that is used and researched. These were described in the theory chapter in Section 2.3.1. By looking at the plain numbers of the techniques we got a good understanding of what the general benefit of the techniques in a category and which of the individual techniques that seem most promising. However, not all prefetching techniques are suitable to be used in a proxy environment, such as the technique proposed by Narayanan et al. [22] which is used in content distribution networks to prefetch already cached responses before they expire. Also, not all prefetching techniques can benefit from the proxy’s knowledge of the clients’ caches. Therefore, the techniques of each prefetching category is discussed below with the suitability in mind.

5.1.1 Markov models

We can conclude that what the best markov models has to offer is a technique based on PPM, due to the popularity and great results in evaluation from several papers. The hard thing is to keep a manageable size of the Markov model which is done in a similar manner for two of the techniques proposed. Both techniques use popularity in some way to limit the model. The most promising shows hit rate of 90% while the traffic increase is at most 21% [35]. The other one [36] seem promising with a latency reduction of 23% but it does not report how much traffic is increased. Making the results of the latter one more confident.
In general the prediction accuracy of PPM seem really good and is used for comparison in a lot of literature.

5.1.2 Dependency graph

The technique which proposed path profiles have great prediction capabilities but we do not know anything about how much the hit rate or the traffic increase is affected. We can only assume that the traffic increase will be really high since DG is used. By using a DDG latency reduction of 20.20% is achieved in the last technique which was also used on a proxy, we are therefore confident it would be suitable as a proxy-based prefetching technique. The best alternative using dependency graph is therefore this technique.

5.1.3 Popularity

Since we were interested in prediction of future requests given an initial main request, the top-10 technique could not be used. This left us with the n most popular technique which show good results on hit ratio and network traffic, although they do conclude that PPM is better. It does however requires less of an effort to implement, which is also a factor to consider.

5.1.4 Data mining

Based on the amount of work that have been done in this category, it is fair to say that it is the most popular one. The most promising techniques, by looking at the results, using association rules is the proxy-based technique by mining two sequence accesses, CIMP, and the server based framework. While the latter gets higher hit ratio than rest of the techniques, it does so at the cost of a lot of increased network traffic. We do not know how well it would perform as a proxy-based framework either. CIMP have the second highest hit ratio but is a client-based technique specialized for mobile devices. One could argue that the mobile devices have more constraining factors than a proxy, such as power consumption, and if removed it would improve the results for the technique. There is still an uncertainty regarding this technique which leaves us with the technique by mining two sequence accesses. This technique is already proxy-based and get great results, even though the results on hit ratios are worse compared to CIMP and the server based framework. With 95% of the predictions being correct, the effect on the network traffic will be very small. Therefore, there is most confidence that this technique would perform well for proxy-based prefetching.

The two sequence prefetching technique have also been recognized in newer papers, published in well known journals and transactions, suggesting that the technique is still relevant. One might have expected that some other techniques, due to the increased dynamic content, would be more relevant. But dispite
5.2. Performance results

Which previously proposed prefetching techniques are most beneficial for lowering the users' perceived latency on a proxy?

Due to the popularity in research and the results shown in the presented papers we think it is fair to say that data mining techniques are most beneficial for lowering the users’ perceived latency on a proxy. We argue that the two sequence technique based on association rule mining provides more confidence than the clustering data mining technique by random indexing. However, this is mainly because of it being a more explored area and also that it is proven to work well for proxy-based prefetching. We therefore conclude that the two sequence based on association rule mining should be used in proxy-based prefetching to lower the users’ perceived latency. But with the note that random indexing can prove to be even better, or at least equally good, in the future.

5.2 Performance results

5.2.1 Prefetching

The measurements were done using Opera Turbo 2 together with its content-based prefetching technique, which identify dependencies of responses, with and without the two sequence technique that was introduced by Huang and Hsu [39].

From the results in Figure 4.2 we see that the $L_P$, which is the users’ perceived latency, is lowered by 246 ms, when five days of training data is used. An improvement was expected since the two sequence prefetching technique showed great results on prefetch hit ratio and prefetch byte hit ratio in the paper by Huang and Hsu [39]. Prefetch hit ratio and prefetch byte hit ratio gives an implication of how much the prefetching technique could improve the users’ perceived latency. We can however not say that certain prefetch hit ratio and prefetch byte hit ratio will improve the users’ perceived latency by a fixed value since many factors is part of the users’ perceived latency. A lowering of 246 ms in users’ perceived latency is considered a lot. According to Hoff [54] in

dynamic content increase the cache hits ratios have been proven to be relatively stable over the last decades [16], which is closely coupled with how well a prefetching technique is performing.

For the clustering based data mining techniques random indexing is the clear winner by looking at the results. The other technique, using web navigational graph [44], is more explored and is a good technique for prefetching. RI is a quite new technique, which means it is more experimental, but results were given with great confidence from the paper. Therefore, we find that RI is the most promising clustering based data mining technique.

Which previously proposed prefetching techniques are most beneficial for lowering the users' perceived latency on a proxy?
his blog post about latency, Amazon loses 1% in sales for every 100ms latency. Making a 245 ms an approximate 2.45% less loss in sales for Amazon. We also observe that one day of training data is insufficient since the \( L_p \) is increased by approximately 99 ms. This is of course really bad since the goal is not to increase the \( L_p \), but to decrease it.

We concluded that one day of training data was insufficient due to an increase in \( L_p \) which can be further supported with the approximate 0.24 degradation in Server Load Ratio. When using five days of training data we can lower the \( L_p \) by 246 ms, but we are able to handle approximately a quarter of the load. This decrease in Server Load suggests that we need four nodes to handle the same load that was handled by a single node when not using the two sequence prefetching technique.

The most common negative effect of a prefetching technique is traffic increase. The content-based prefetching technique had close to no traffic increase at all, which is really good. There is two major reasons for this. First, it is good at detecting certain dependencies such that almost all the dependencies are resources that is required by the requested site. Second, Opera Turbo 2 knows about the local cache of its clients, which means that traffic increase related to unnecessary prefetches that are already in the client’s cache can be completely avoided. Worth noticing is that traffic increase is not always a bad thing, only when it increases the users’ perceived latency. A proxy normally have many periods when its bandwidth is underutilized and a certain traffic increase would not impact the users’ perceived latency significantly. The two sequence prefetching technique does however increase the traffic by approximately 27% (for both one day and five days of training data), which can be seen in Figure 4.3. The traffic increase is related to bad prefetches, which was never requested by the client.

The benefit of prefetching can be determined by the metrics Recall and Byte Recall. From Figures 4.4 and 4.5, we observe that the content-based prefetching have Recall and Byte Recall close to zero. This means that the usefulness of the content-based prefetching technique is almost none. The reason for this is that while it performs well at identifying good dependencies to prefetch (according to the low traffic increase), the majority of the dependencies are not identified. But when the two sequence prefetching technique is added we see improvement in both Recall and Byte Recall. While it is not considered good that the content-based prefetching misses the majority of dependencies, it makes it easier for us to compare, when using the two sequence technique, with the results provided by Huang and Hsu [39]. They report prefetch hit ratio of approximately 95% and prefetch byte hit ratio of approximately 95%. These
5.2. Performance results

metrics are measurements on the accuracy of the prefetching technique, while Recall and Byte Recall measures the usefulness of the prefetching technique. High accuracy would lead to high usefulness if a lot of prefetching is done. They do not measure the hit ratio when no prefetching was used for the system used by Huang and Hsu [39], but they report that the worst prefetching technique measured approximately 45% in cache hit ratio and 45% in cache byte hit ratio. By comparing the cache hit ratio and cache byte hit ratio of these bad performing prefetching techniques we get a minimum of how much the cache hit ratio and cache byte hit ratio were increased by using the two sequence prefetching technique. This can be considered a measurement on the usefulness of a prefetching technique. We can therefore, by looking at the results from the paper by Huang and Hsu [39], assume that the cache hit ratio was increased by at least 20% and the cache byte hit ratio was increased by at least 15% using the two sequence prefetching technique. Based on these observations we can conclude that the Recall of the two sequence prefetching technique is less than expected. We got a usefulness of approximately 12% while Huang and Hsu [39] got at least 20%, by only looking at amount of good prefetches. However, when looking at the usefulness when considering the size of the good prefetched responses we get at least 40% while Huang and Hsu [39] got at least 15%. This is very interesting results because that Huang and Hsu [39] report less usefulness looking at the amount of good prefetched bytes compared to the amount of good prefetches. While we got the opposite with more usefulness looking at the amount of good prefetched bytes compared to the amount of good prefetches. We also got a magnitude in difference compared to Huang and Hsu [39]. This means that even though the two sequence technique did a lot of bad predictions for us, the correct ones were mostly large responses. Otherwise we would not have such a big difference in Recall and Byte Recall. Huang and Hsu [39] did their evaluation of the two sequence prefetching technique using trace based simulation and we evaluated the same technique in a live system, this can have a huge impact on the results. We also do not know when the logs used by Huang and Hsu [39] were actually collected, since the paper is from 2008 we know that they are at least 8 years older than our logs. The WWW has changed a lot in 8 years, not only is the content more dynamic but according to a survey [54] the amount of websites has increased by 690 million in the last 8 years. By looking at the two sequence prefetching technique with one day and five days of training data, we can see that the difference is approximately 0% and 3% in Recall and Byte Recall respectively. This is a very small difference, which implicates that the amount of training data does not significantly affect Recall and Byte Recall when one or more days of training data is being used.
How much does a prefetching technique in a live system lower the users’ perceived latency on a proxy, when the clients’ caches are known?

The two sequence prefetching technique were considered most promising on a proxy-based system when the client’s caches are known. Evaluations in a live system show that we can, by an average, lower the users’ perceived latency by approximately 246 ms. An effect of this is that the traffic is increased by approximately 27% which together with the prediction time lowers the amount of requests being handled on the proxy to approximately one fourth.

5.2.2 Pushing

The measurement of push were used on Opera Turbo 2 together with its content-based prefetching technique, which identify dependencies of responses, and with the two sequence technique introduced by Huang and Hsu [39].

From the results in Figure 4.7, we can see that by using a Very Conservative strategy we maximize the Request Savings. The expected results here would be that as the minimum push confidence is decreased, the Request Savings are increased due to more push being done to the client. Except for the results of Very Conservative, this expectation is satisfied. However, when using a Very Conservative strategy something interesting happens, the request savings is at approximately 0.093 which is an increase of approximately 0.003 compared with the Aggressive strategy. Our explanation to this is that due to more push being sent to the client, the client cache reaches its limit faster causing cached objects to be replaced. This behaviour can be observed from Figure 4.8 as well, where the only strategy improving the Miss Rate Ratio is Very Conservative. This means that more requests are served from the client cache when using Very Conservative compared the other strategies, supporting our theory about the client cache being flooded causing less cache hits. We can conclude that somewhere between 0.1 and 0.8, in minimum push threshold, the Request Savings starts to improve even though the client cache is being flooded. This is supported by the Request Savings trend seen for Conservative, Medium and Aggressive where the Requests Savings are increased when the minimum push confidence is decreased.

The negative impact of push is the traffic increase between client and proxy. We can see in Figure 4.9 that none of the strategies increases the traffic with more than 0.33%. This is not a very large traffic increase at all. Our expectations would be that as the minimum push confidence is decreased, the amount of traffic is increased. This is not what our data show, e.g. by looking at the None strategy where we get the highest traffic increase, but we are expecting it to be the lowest. However, by looking at the traffic increase range which is 0.25% and 0.33% gives us the very small variation of 0.08%. We consider this variation negligible and therefore conclude that the traffic increase is not
affected by the different push strategies. But this conclusion still does not follow our expectations. The explanation for this is simply that mostly good pushes are sent to the clients, which is an effect of the proxy’s notion of the clients’ caches that reduces the amount of bad pushes.

**Without affecting the network usage significantly, when should a prefetched resource be pushed from proxy to client?** By using the two sequence prefetching technique and measuring with different values on the minimum push confidence we could determine when a prefetched resource should be pushed. The measurement were done in a live system and we conclude that by using a minimum push confidence of 0.8 we can reduce the amount of client requests by approximately 9.3%. This is done with approximately 0.3% traffic increase and does therefore not affect the network usage significantly.

### 5.3 Methodology

#### 5.3.1 Implementation

The implementation of the two sequence prefetching technique was done in the language Pike. The language choice did not have too much impact on the result since it does not perform any computational heavy procedures when predicting prefetches, the most of the time is in idle waiting for the database to return results. By instead looking at the preprocessing which is a really computationally heavy process, because of the huge amount of data handled and the complexity of $O(N^2)$ to create the rule table. This is however not done during the measurement, but before the system is started and therefore it does not impact the performance while running the system with prefetch enabled.

#### 5.3.2 Evaluation metrics

The metrics used for evaluation of prefetching were Latency Per Page ($L_p$), Traffic Increase caused by prefetching ($TI_{pref}$), Recall ($R_c$) and Byte Recall ($R_{cb}$). The most important metric here is $L_p$ since it is the users’ perceived latency and gives us an overall view of how good the system is performing. But traffic increase is expensive, since a more powerful backend is required this can therefore be an unwanted effect if the prefetching technique is performing poorly but still requires a lot of extra bandwidth. To evaluate how well the prefetching technique is performing we used Recall and Byte Recall. This gives us a measurement on how much help the system gets from prefetching. By looking at the Server Load Ratio ($SLR$), which is a metric on how much requests per hour are served, we evaluate the system performance. The measurement and prefetching technique was setup on a single node, getting served requests by a load balancer. The load balancer automatically sends fewer requests to a node with high workload. That means if the prefetching technique is computationally
heavy, such as fewer requests are served, Server Load Ratio will catch this performance degradation. But one may argue that hardware is cheap and since the goal was to lower the users’ perceived latency we can simply add more nodes that serves less requests but faster.

Recall and Byte Recall was used to evaluate how well the prefetching technique performed. These metrics only consider the good predictions, not the bad ones. Meaning we can have a lot of bad requests and a lot of good requests. Simply looking at the Recall and the Byte Recall in this given case might look very well, but in reality approximately half of the prefetches are bad. However, this will be captured by the traffic increase between the proxy and web servers. But the traffic increase might also be related to other effects of the system. To see how well the actual prediction is working the metrics Precision and Byte Precision should be used, which was concluded by Domènech et al. [47]. Precision and Byte Precision are both metrics which measures ratio of good predictions versus the total number of predictions. This could later be used to verify that a certain traffic increase was directly related to bad prefetches.

The metrics used for evaluation of push were Request Savings (RS), Miss Rate Ratio (MMR) and Traffic Increase caused by push ($TI_{push}$). The request savings gives a general view on how much push helps the clients to improve their cache hits. The miss rate ratio also gives a general view on how much cache misses are improved by push. However, it does not consider how well the actual push is working. With the same reasoning above for prefetching metrics, Precision and Byte Precision could be used to verify that a certain traffic increase between the client and the proxy were caused by push.

Measurement of Latency Per Page is hard to achieve. Opera Turbo 2 together with the Opera browser support this feature where the browser will report back to the proxy when the site has finished loading. However, it is proven to be somewhat hard to verify the validity of the stats on dynamic sites. You can ask yourself the question of when a site is fully loaded. A completely static site has finished loading when all of the requests of the resources have been served and rendered. But many sites make sure that the required resources are loaded as soon as possible such that the user can interact with the site without all of the resources loaded. It might not even be noticed by the user if e.g. some JavaScripts are missing. When all requests have been processed and rendered in the browser we know that the site has been fully loaded, but it might have been fully displayed to the user before that. From this we can say that $T_{loaded} \geq T_{displayed}$. 

5.4 The work in a wider context

5.4.1 Adaptive streaming

Prefetching techniques such as those discussed here are also valuable in many other contexts, including for video streaming. For example, in adaptive streaming we can use prefetching to improve the users’ perceived Quality of Experience (QoE). One use case is to preload multiple videos that the user might navigate to in a near future. To do this some kind of recommendation system have to be used to predict which videos to preload. This is closely coupled with prefetching of resources to a website since the recommendation system can be based on users access patterns. In this case the two sequence prefetching technique can be used in a similar manner to predict the N next videos that should be preloaded. The second part in preloading multiple videos is to do the actual preloading of video chunks, in this case the two sequence prefetching technique evaluated in this paper is not useful. Instead a policy can be used, this is evaluated by Krishnamoorthi et al. [55] where they conclude that a token-based adaptive-quality policy is most promising. However, this on a client which means we will not be able to benefit from push, which was evaluated in our paper. In another paper by Krishnamoorthi et al. [56] they evaluate proxy assisted adaptive streaming and conclude that the prefetching techniques, used in the evaluation, does not perform well when the link between a proxy and a server is the bottleneck. These prefetching techniques only consider which chunks to preload. Another prefetching approach would be to change the main objective of the prefetching technique to maximize the amount of cache hits on the proxy, and not do next request predictions. This is not what is being evaluated in our paper but in theory we should be able to use the two sequence prefetching technique as a recommender system for which streams are most likely to be accessed and prefetch N of these chunks to the cache. Although the results may vary a lot since this is a completely different use case than what is studied in our paper.

5.4.2 Live environment

While the two sequence prefetching technique performs well, more techniques should be evaluated in a live environment since this actually catches the current state of the WWW. In a paper by de la Ossa et al. [57] they also see the need for evaluation in live environment. They use Delfos, an environment where prefetching can be evaluated on real system, which was introduced in a paper by the same authors [58]. But they still want to keep the workload consistent to have a fair comparison. This is done by a program they called mod-trainer which gets fed web server logs to reproduce the real user requests. To say this is in a live system is according to us inaccurate, because in a live system you get effects that are not captured by simply repeating the requests done from a web server log. Consider the case when a request is received faster because of prefetching,
then the browser will identify dependencies and requests those. The proxy will now receive the requests of those dependencies at a different time, potentially affecting the results by changing the state of the cache and the workload of the proxy. While this should affect the results, we do not know by how much. That makes it hard to say that a live system is better. Evaluation of how much a system being fed web access logs differs from a live system, would be required to determine which is better, or if they are equally good.

5.5 Ethical and societal aspects

5.5.1 Integrity of the user

The database of rules created by the two sequence prefetching technique stores the id of a browser session together with first URL and second URL. This means that the browser session id is bound to a computer and most likely a specific user. This id is part of the session handshake between the browser and the proxy which is done securely. There is no way to identify the person by simply looking at the id. But security always changes and something that is secure today might be insecure tomorrow. This means that, in theory, someone could obtain the id of the session (even though this is most unlikely). But without the rule database the user id will not give the attacker any useful information. Only after an attacker would gain access to the database, the integrity of the browser client would be broken because of the session id to URL mapping being exposed. To gain access to the database, the whole Opera Turbo 2 security system would have to be compromised which makes this a fairly unlikely scenario, but there still exists a small possibility. Due to a very small chance of this happening it is almost fair to say that the integrity is not compromised by the rule database and the same goes for the logs being stored on the proxy.
6 Conclusion

When a site is requested, prefetching of web resources required by that site is a good way of reducing the users’ perceived latency. In this thesis various prefetching techniques are considered to be used in a system with a proxy when the clients’ caches are known. The performance of the top most promising prefetching techniques are discussed in a proxy environment where the clients’ caches are known, and the most promising were chosen for evaluation. A prefetching technique based on data mining of popular two sequence accesses, and storing them as probabilities in a rule table, were concluded most promising in the given proxy context. The two sequence prefetching technique was implemented in Opera Turbo 2, a proxy system which knows about the clients’ caches, used and developed by Opera. This system already did some content-type prefetching which was complemented by the two sequence prefetching technique. Measurement was done on live requests on one of the nodes used in production by Opera Turbo 2, both with and without the two sequence prefetching technique. Since the two sequence prefetching technique required training based on previous access logs, both a shorter (one day) and a longer period (five days) of training logs were considered. We conclude that the two sequence prefetching technique can lower the users’ perceived latency by approximately 246 ms, with a traffic increase of only 27% but with one fourth of the requests handled. This improvement is only caused by the two sequence prefetching technique and not the content-type prefetching technique already being used in Opera Turbo 2. We also performed measurement to determine when it is best to push the resources, prefetched by the two sequence prefetching technique. This was done by evaluating five different values on minimum push confidence, which is a value provided by the two sequence prefetching technique. We conclude that by using a mini-
mum push confidence of 0.8, we can reduce the client requests by approximately 9.3% with a traffic increase of approximately 0.3%.

6.1 Further research

This paper performs an evaluation of a two sequence prefetching technique based on data mining. Measurements are done in a live environment with requests from real users, where we also consider when to push prefetched resources to users before it is requested. By doing this we lowered the users’ perceived latency, which is of great importance on a proxy-based system.

6.1.1 Additional evaluations

The training part of the two sequence prefetching technique, used in this paper, does only consider a minimum confidence threshold of 0.01. We did not consider other values on the threshold since this had already been evaluated by Huang and Hsu [39], where they conclude that a threshold of 0.01 should be used to obtain best results. However, it would be good to verify this by performing evaluation on the same threshold values and see if it would produce similar results when measured in a live environment, such as Opera Turbo 2. The expected results is that as the minimum confidence threshold is increased from 0.01, the prefetch hit ratios are lowered.

When evaluating the prefetching techniques we considered two scenarios: one using one day of training data, and one using five days of training data. We used one day of logs to get a smaller set of requests and five days to get a larger set of requests, when generating the rules for the training part of the two sequence prefetching technique. An additional scenario using three days of training data would have provided more information about the different trends of the evaluation metrics, such that we would get a clearer view of how the result curve looks like for each of the different evaluation metrics. That way we might have been able to draw more conclusions about how many days, or hours, of request logs that should be used to achieve optimal conditions for the two sequence prefetching technique.

6.1.2 Cache replacement policies

One part that we did not focus on in this paper was which cache replacement policy that would work best together the two sequence prefetching technique, on the proxy, and the provided push strategies, on the clients. This could have had a great impact on the results and would therefore be something very relevant to do further studies about.
6.1.3 More prefetching techniques

By doing a literature study we found the most influential and relevant papers to prefetching. But due to the limited time period of a master’s thesis we only had time to evaluate one prefetching technique. It would be great to evaluate more prefetching techniques on the following:

1. Determine how well they perform in a live proxy-based environment
2. If they would benefit from that the proxy knows about the local cache of the clients
3. When it is beneficial to push the prefetched resources to the clients

There is very limited studies being done on all of these subjects, if any. While we think that the two sequence prefetching technique performs well, we also believe that there exists better ones that is more suitable for Opera Turbo 2, and other similar systems, such as a prefetching technique based on random indexing.

6.1.4 Database optimizations

The prediction of the two sequence prefetching technique required a significant amount of system resources. This has mainly to do with the size of the rule table, which can be observed from Figure 1.6, where one day of training data allowed for approximately 20% more requests to be handled compared to five days of training data. The big difference here is the amount of rules in the database that have to be searched through when doing the prediction. MySQL was used as a database to store the rules and query for them in this paper. This can have a great impact on the system load and there might be better suited databases, preferably NoSQL database that is very different from MySQL, such as MongoDB.

6.1.5 Additional evaluation

We perform an evaluation on the two sequence prefetching technique introduced by Huang and Hsu [39]. In their evaluation of the two sequence prefetching technique they conclude the best value for minimum confidence, to determine if the two sequence should be added to the rule table, is 1%. They evaluate using minimum confidence of 1%, 5%, 10% and 20%, where you can clearly see that as the minimum confidence grows larger the prefetch hit ratio and prefetch byte hit ratio are lowered. Additional evaluation on the two sequence prefetching technique can be done to verify that these minimum confidence values produce similar results on a live system.
Bibliography


[34] Lei Shi, Ying-Jie Han, Xiao-Guang Ding, Lin Wei, and Zhi-Min Gu. “An SPN-based integrated model for Web prefetching and caching”. In: Journal of Computer Science and Technology 21.4 (2006), pp. 482–489.


[52] Pike. May 2016. URL: http://pike.lysator.liu.se/


