An Effective Pre-refresh Mechanism for Mobile Handheld Device's Embedded Web Browser

Hua Qiang Li [0], Young Hak Kim
Department of Computer Engineering,
Kumoh National Institute of Technology
huaqiang@cespc1.kumoh.ac.kr [0], yhkim@cespc1.kumoh.ac.kr

ABSTRACT

Mobile handheld devices such as PDA(Personal Digital Assistant) and Cellular Phone are more and more popular for personal web surfing. But today, most mobile handheld devices have relatively poor web browsing capability because of the small bandwidth of wireless connection, the small speed of CPU, the small capacity of RAM and Flash ROM, the small display size of LCD and the short lifecycle of battery. Users have to endure bigger communication latency than Desk PC's one because of these factors. In this paper, we propose an effective pre-refresh mechanism for handheld device's embedded web browser so as to make user's web surfing faster. The experiment was done using the simulator designed by ourselves and the experiment result demonstrates the proposed mechanism has a good performance to make web surfing faster.

1. Introduction

Recent years, World-Wide Web technologies and mobile handheld devices have had a very fast development. Now, most mobile handheld devices have been integrated with embedded web browser for surfing the internet, especially PDA products. So users can access the internet from everywhere at any time.

But now, most mobile handheld devices have small wireless bandwidth, typically the connection speed is anywhere from 9600bps(bits per second) to 28.8 kbps[1]. At these low rates it can take up to 3.5 minutes to download a 250KB web page. If the link is very lossy, this time can be significantly extended. Processors used in most handheld devices do not have the power of those found in PCs. Taking PDA as an example, most current PDAs' CPU speed is from 160MHz to 206MHz. Web pages with large amounts of graphical content or a lot of script files need to be handled by the browser decoding and interpreter engines. This can overload the processor and cause the device to become unresponsive, and in turn, unusable. Adding handheld device's small memory, small display size and short battery lifecycle, all these factors bring big communication latency for users to access the internet.

In the following, we first summarize the existing technologies for reducing client latency, indicate their advantages and disadvantages, then introduce our proposed mechanism that makes the internet surfing faster for mobile handheld device users.

2. Related Works

Ever since World-Wide Web emerged, reducing client latency has been one of the primary concerns of the Internet R&D community. Many techniques were proposed for reducing client latency.

- Prefetching between caching proxies and browsers[2] is a well-known technique for reducing latency for modem users. Because the low modem bandwidth is a primary contributor to client latency, this approach relies on the proxy to predict which cached documents a user might reference next, and takes advantage of the idle time between user requests to push or pull the documents to the user. This approach can reduce user perceived latency up to 23.4% as the author said. But this approach is not supported by most current ISPs and LAN Proxies, because it is very difficulty to realize and imposing a big process burden for proxies.
- Large Browser Cache is another technique to reduce client latency[2]. It increases the hit ratio and reduces network traffic. Infinite browser cache(almost no replacement occurs) is supposed in the experiment[2]. The result shows it reduces 4.1% client latency. But this approach is absolutely not fit for mobile handheld devices. Taking PDA as an example, general deskpc's browser cache has the a default size of 5-8MB[2]. But most current PDA products have the system memory from 8MB to 32MB and Flash ROM from 2MB to 32MB. It is impossible to allocate a big browser cache to PDA.
- Delta Compression technique[3] only transfer modified web pages between the proxy and client. That is, if an old copy of the modified page exists in the browser cache, the proxy only sends the difference between the latest version and the old version. This approach can eliminate 88% of bytes in transfers of modified objects, according to the results in[3]. But until now, this technique is not a official standard for extension of HTTP[6], most current web servers, ISPs, Proxies and Clients don't support it. So mobile handheld device which uses wireless modem for web surfing can't get benefits from Delta Compression Technique.
- Application-level compression to HTML documents was investigated in the studies[3,4]. It has suggested that HTML texts can be first compressed, and then transferred from one end to another. HTTP/1.1 supports application-level compression via the "transfer-encoding" tag[5]. This technique can eliminate 25% of bytes in transfers of HTML documents, according to the average number reported[3]. But this technique needs CPU overhead for compression and decompression. It is not practical for slow speed CPU used by mobile handheld device. It will aggravate the burden of CPU and memory of mobile handheld device during web surfing and may result in more latency for users.

3. Proposed Mechanism

Because the techniques mentioned above are not fit for *reducing client latency for mobile handheld devices*, we propose "the effective pre-refresh mechanism", and this mechanism also works well in Desk PC's browser.

3.1 Overview

Through plentiful observing, we find mobile handheld device's user has his habit in web browsing. It means every user has his some favorable web addresses. For example, somebody likes yahoo, google, daum, microsoft etc. From the browsing beginning, user always opens one of his favorable web pages, and finds the information which he wants. In this case, after a long period, objects saved in mobile-handheld device's browser's cache are these which user always browses. So most user's favorable web objects are saved in mobile-handheld device browser's cache.

In addition, during user's reading web documents, there is much free time wasted. We should use the free time to pre-refresh the objects in cache. Through research, we find when user begins his browsing, almost all the HTML documents in the cache expired. For every request, a browser mainly executes below steps[7], seeing Figure 1:

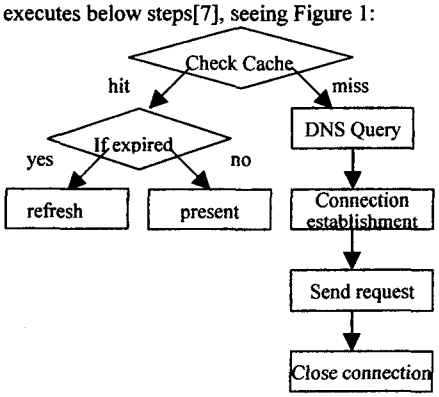

Figure 1: Steps for every request in browser

So we should make browser pre-refresh the expired web objects in the cache during idle time. In this case, after users read documents, if users request the web objects which are already pre-refreshed in the cache, the web objects will be very fast showed to user. User will like this fast speed. Furthemore this mechanism will fully use CPU process ability and network bandwidth during idle time, consequently reduce latency and save money for user during web browsing.

3.2 Implement scheme

The basic assumptions behind "pre-refresh mechanism" are:

- Users have idle times between requests, because users often read some parts of one document before jumping to the next one.
- The browser can predict which web pages a user will access in the near future based on number of the reference for every cached object observed from the past period.
- The browser has a cache that holds user's favorable web pages. Furthermore, because most mobile handheld device has small Flash ROM, it can't allocate much capacity for browser cache, and most commercial web sites make their web pages so big, for example, the full content of yahoo main page is 128KB including HTML documents, GIF, JPG, Javascript, Flash etc. It is impossible for embedded browser to support and cache all the web objects. So we assume only HTML documents are cached. It is reasonable because the text information in the HTML documents are most important for users.

The cumulative distribution of user idle time in the UCB, DEC and Pisa traces[2] shows 40% of the requests are preceded by 2 to 128 seconds of idle time, indicating plenty of pre-refresh opportunities.

The basic pre-refresh mechanism are:

1. In the embedded browser cache, we allocate an attribute named "reference number" to the cached HTML document object. This attribute means the number of reference for every cached HTML document in past period. We will use this attribute as primary index to sort the cached HTML documents. If a HTML document object is first cached, we assign 1 as the value of this attribute. Later, if user requests this HTML document object again and this HTML document object is always in the cache, we will increase the attribute's value with 1.
2. We use the attribute "reference number" as the primary index and the attribute "size" as the second index to sort all cached HTML document objects. The attribute "size" means the size of the cached HTML document. The attribute "size" is a default attribute in most browser cache. For example, Microsoft Internet Explorer's cache has 8 attributes "size", "name", "internet address", "type", "expiration date", "last modified date", "last accessed date" "last validated date". During sorting, we use "reference number" to arrange cached HTML documents in descending order, when "reference number" is same, the one which has relatively small "size" value will be sorted ahead. For every request, we update the sort.
3. We use Least Frequently Used(LFU) removal policy with the attributes "reference number" and "size" to remove HTML documents when the cache is saturated. In this case, the HTML documents with low "reference number" value and big "size" value will be removed continuously until the cache has enough room for the coming HTML document.
4. We provide a background process to pre-refresh cached HTML documents during idle time. This process runs from browser's open to close. According to experience, "idle time" judge condition is appointed to 10 seconds. It means if user has no request during 10 seconds, this process will pre-refresh cached HTML documents continuously according to Step 2's sort. During pre-refreshing, if user has request, pre-refresh will be stopped unless the request is for the object that is being pre-refreshed.

3.3 Performance Metrics

We are mainly interested in the following performance metrics:

- **Request Savings**: the number of times that a user request hits the pre-refreshed HTML documents in the browser cache, in percentage of the total number of user requests.

- **Wasted Bandwidth**: the number of times that the pre-refreshed HTML documents are not hitted by user requests, in percentage of the total number of user requests.

"Request Savings" is the primary goal of pre-refreshed mechanism. It will prove if the pre-refreshed mechanism has a good performance for reducing latency for users. "Wasted Bandwidth" can be tolerated, because it only uses idle time's network bandwidth and CPU process ability, and if there is no pre-refreshed mechanism supported, these network bandwidth and CPU process ability are also wasted.

4. Simulation and Performance Analysis

Because it is very difficult and time consuming to modify the open source embedded web browsers in order to implement our proposed mechanism and do performance analysis, we designed the simulation tool with Visual Basic6.0 for getting the result of performance metrics mentioned above. Also, we run the simulator in the Notebook Computer with CPU 133MHz and university's LAN with simulator's cache size 512KB. Because this Notebook Computer CPU speed and network speed is very close to the performance of mobile handheld devices (especially close to PDA), it ensures our experiment result is very close to the real environment.

Steps for implementing the "Pre-Refresh Mechanism" and "Performance Metrics" with our simulator are:

1. We use Visual Basic's "Web Browser Object" and "Internet Transfer Object" to make a simple web browser. We use it to simulate the embedded web browser. It is reasonable because our work mainly concentrates on the browser cache and pre-refresh function. It doesn't interfere to the vision effect of the small size screen in most mobile handheld devices.
2. We use IE's cache directory "Temporary Internet Files" and a ACCESS Database to simulate the cache for Step 1's simple web browser. In the access table, we mainly create four fields "Reference Number", "Size", "Hit" and "Miss". The meaning of "Reference Number" and "Size" is already explained in "3.2 Implement scheme". "Hit" means the number of times of user request hits the pre-refreshed HTML documents. The sum of total records "Hit" field's value divided by total request number is the value of performance metric "Request Savings". "Miss" means the number of times that the pre-refreshed HTML documents are not hitted by user requests. The sum of total records "Miss" field's value divided by total request number is the value of performance metric "Wasted Bandwidth".
3. We simulate 512KB for cache capacity and use LFU removal algorithm with "Reference Number" and "Size" fields to remove cached HTML documents when cache is saturated.
4. We use "Timer Object" to do "pre-refresh mechanism" during idle time and assume the idle time judge condition is 10 seconds.

The experiment result is "Request Savings" has the value 23.16%, "Wasted Bandwidth" has the value 47.35%. So our proposed "pre-refresh mechanism" makes web surfing faster than there is no this mechanism for mobile handheld device users. Though "Wasted Bandwidth" is high, but as we explain in "3.3 Performance Metrics", if there is no pre-refreshed mechanism supported, these network bandwidth and CPU process ability are also wasted during idle time.

Furthermore, we run the simulator with Desk PC with CPU PIII 800MHz and simulator' cache 119MB, the experiment result shows our proposed mechanism gets greatly better performance .

5. Conclusions and Future Work

Today, more and more internet population use mobile handheld device(such as PDA, Cellular Phone etc.) to access the World Wide Web. But users spend a lot of time impatiently waiting for web pages to come up on screen. In this paper, we summarized the existing techniques to reduce client latency, indicating their strength and weakness, then we proposed our "An effective pre-refresh mechanism for mobile handheld device's embedded web browser". Our proposed mechanism uses the idle time of user reading web pages to pre-refresh cached HTML documents in the embedded web browser. Through our simulation, it greatly reduces the latency for users during web surfing.

Although this mechanism has been aimed primarily at mobile handheld device's embedded web browser, the author believes that it may also be appropriate for Desk PC's Browser. A number of future work will be done.

- We will practically implement this mechanism in the open source embedded web browser in order to evaluate it's performance in the real environment.
- We will design more performance metrics and use more traces to more accurately evaluate the performance of our proposed mechanism.
- We will merge other techniques with our proposed mechanism to more reduce latency for users.

References

[1] Anthony Massa, "9 Tips for Improving Embedded Web Browser Design", http://www.commsdesign.com, Feb.5, 2002.

[2] Li Fan, Pei Cao, and Quinn Jacobson, "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Perfromance", International ACM Conference on Measurement and Modeling of Computer System, PP. 178-187, 1999.

[3] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy, "Potential benefits of delta encoding and data compression for http", In Proceedings of ACM SIGCOMM'97, August 1997, Available from http://www.research.att.com/~douglis/

[4] Henrik Frystyk Nielsen, Jim Gettys, Anselm BairdSmith, Eric Prudhommeaux, Hakon Wium Lie, and Chris Lilley, "Network performance effects of http/1.1", In Proceedings of ACM SIGCOMM'97, August 1997, Available from http://www.w3.org/Protocols/HTTP/Performance/Pipeline

[5] R.Fielding, J.Gettys, J.Mogul, H.Frystyk, and T.Berners-Lee, "Hypertext Transfer Protocol-HTTP/1.1 RFC 2068, Jan 1997.

[6] Jeffrey C. Mogul, "What is HTTP Delta Encoding", http://webreference.com/internet/software/servers/http/deltaencoding/intro/, 2002.

[7] Md. Ahsan Habib, Marc Abrams, "Analysis of Sources of Latency in Downloading Web Pages", WebNet 2000, October 30 – November 4, 2000 San Antonio, Texas, USA