Processor Architectural concepts in the web and the web browser

Chan Kit Wai and Somasundaram Meiyappan

1. Introduction

Pipelining is a general concept that was borrowed from product manufacturing lines and is the concept of performing the various steps needed towards product completion in parallel; but the operations in those steps/stages would be performed on a different piece of the same product. This thus enhances the throughput of the line.

Computer architecture scientists used this concept in the processors for increased throughput. We also have applications and similarities to not just pipelining; but to the various other concepts in computer architecture in the web - especially web browsing.

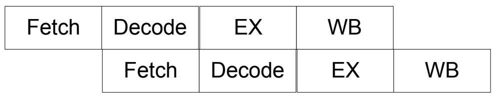
2. Basic concept of pipelining in the web

The application of pipelining concepts is found in HTTP request, the protocol behind the world wide web. HTTP 1.1 supports pipelining by issuing multiple HTTP requests to be written out to a socket together without waiting for the corresponding responses [Moz]. Instead of sending a request and waiting for a reply to return before sending the next request, It stores all of their requests on the local web server, and then sends them all at once. Pipelining is a design choice that would cut down significantly on three things: message creation overhead in the client, network traffic, and server-side congestion. A single pipe could encapsulate several page requests, making it faster to build than several independent page request messages.

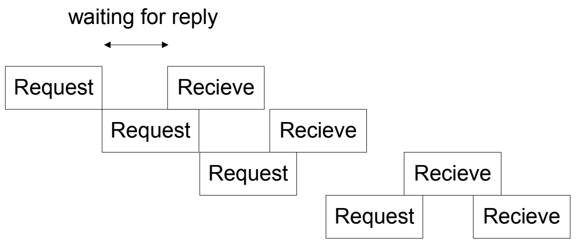
In addition, the end user can also get greater efficiency in another way. By remembering the actual IP address of the server it is requesting files from. It reduces the overhead in order to translate the URL into an IP address, hence saving computational and communication overhead in a big way when the traffic load is high.

The HTTP pipelining concept is similar to the instruction level pipelining in some of the areas. The process of loading a web page consists of sending a request packet to the destination server, receiving packets from it, decoding the packet, assembling them and translating into contentto be displayed in the browser. From a non-pipelined processor point of view, each instruction is started only after the previous instruction has completed. Similarly in non-pipelined HTTP 1.0, each new request is sent only after the previous request is acknowledged with received data packets. For both scenarios, the processor and the browser have a low efficiency in terms of instruction throughput and the downloading time of a webpage respectively. Interestingly, this concept has also trickled down to the connection protocol TCP. A new implementation called FastTCP can send consequtive packets to the client after a certain time even if the first packet has not been acknowldeged. More information can be obtained from *http://netlab.caltech.edu/FAST/*.

With pipelining enabled, the request and receival of packets is separated into 2 stages. The browser can send multiple request packets without waiting for the packets to be received. As a result, the throughput of downloading of objects is increased.



Processor Pipeline



Web Pipeline

However, there are some differences between the HTTP pipeline and the instruction level pipeline. Basically, the execution of each stage in the processor pipeline is synchronous but the stages for the web pipeline are asynchronous.

In Instruction pipelining the fetch stage is periodic whereas in web pipelining, the request packet is event triggerred (only upon user request or trigger from the html parser) or asynchronous. This is true for the case of retrieval of data packets. This contributes to an undetermined throughput rate for the download process. In fact, the latency of the download process is neither known in the first place, since it all depends on the load of the network and several other factors.

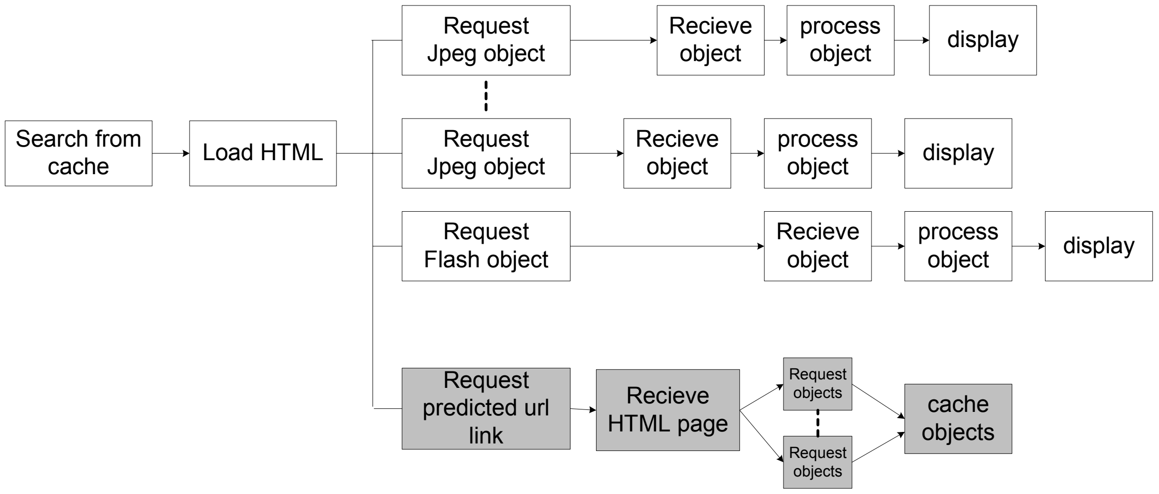
There are problems that are more significant in web pipelining than in processor pipelining. When a user wants to load a url that is in the last link in a pipe, it would force the browser to wait for all other pages in the pipe to be received. This situation can be undesirable if the last link expects a packet that can arrive in a shorter period of time upon request than the preceding packets. In fact, the web pipeline, at times, does not really require a "completion in order" to ensure the webpage is displayed correctly.

The current HTTP pipelining also has some other dependencies before the issue of request packets. If a web page is requested by the browser, the html file must be received and parsed before it can request to load the embedded objects (e.g jpeg) within the page. If the page contains a frame, more than 1 html files have to be loaded. As a result, there is information (data) dependence prior to the request of the embedded data objects.

3. Pipelining concepts for web page retrieval

The concepts in computer architecture can be used to implement the pipeline for the web. Upon a request of the url link, the browser attempts to fetch the latest copy of the html files and object from the cache, if any. If none is cached, the browser sends a request to the destination web server.

Instead of requesting the entire html page, a request is sent to query for any embedded objects in the requested url link. If there are any, the web server sends the information of the embedded object(s) to the browser. The browser then sends concurrent requests to retrieve all desired objects to be downloaded. Hence, this method bypasses the need to download the entire HTML file. Rather it receives the embedded object information earlier and sends all request for the object before actually downloading the lengthy HTML file(s) and parsing them on the fly to look for the embedded objects. Thus, this concept reduces the data dependence time discussed above. Actually, the motivation for this is very similar to the motivation for the need for bypassing or forwarding in microprocessors and that is to reduce the delay in dispatch because of data dependency.



When assigning different requests to the various embedded objects within the webpage, the number of requests can be viewed as the number of EUs in the micro-architecture of a processor. Increasing the request exploits the concurrency of requests and the downloading process. Since each request is normally made through a separate connection to the server, there normally is a maximum restriction on the number of simultaneous connections we can have. Most popular servers allow a maximum of only 4 simultaneous connections from a machine to better serve all customers and also for security reasons. And so do the browsers.

In addition, the concepts of speculative data prefetching and branch prediction that are used in modern processor architectures can be seen in certain web-browsers. Based on the access pattern of the user, certain web-site's home pages may be prefetched by the browser in the background when the user is viewing the currently loaded web-page in the browser. This if predicted correctly would load the prefetched page when the user wants. But unlike in the processor, a mis-predicted fetch of the other website is not flushed from the browser cache.

If the above concept is similar to branch prediction, the browsers also employ another type of prefetching which would look similar to speculative data prefetching. Almost all web-pages in the world has one or more links to other webpages or websites. During the idle time that the browser gets when the user is viewing the currently loaded page, browsers look for the links that are found in the page and cache those pages. Different browsers have different schemes here. Some browsers could opt to prefetch these links based on the text of the link (and user preferences/favourites) to reduce hard-disk activity and the machine's activity whereas others prefetch just about all links in the page.

4. Conclusion

As we have seen, there exists problems or scope for improvement in the design of a web browser and the web itself just as there exists in processor archtiecture. And solutions to some of the problems in both areas have similar concepts. Concepts like pipelining and parallel execution can be deployed in even in areas like imaging and in almost any process where throughput always needs to be improved upon.