BigPipe: Pipelining web pages for high performance

By Changhao Jiang on Friday, June 4, 2010 at 11:34am

Site speed is one of the most critical company goals for Facebook. In 2009, we successfully made Facebook site twice as fast, which was blogged in this post. Several key innovations from our engineering team made this possible. In this blog post, I will describe one of the secret weapons we used called BigPipe that underlies this great technology achievement.

BigPipe is a fundamental redesign of the dynamic web page serving system. The general idea is to decompose web pages into small chunks called pagelets, and pipeline them through several execution stages inside web servers and browsers. This is similar to the pipelining performed by most modern microprocessors: multiple instructions are pipelined through different execution units of the processor to achieve the best performance. Although BigPipe is a fundamental redesign of the existing web serving process, it does not require changing existing web browsers or servers; it is implemented entirely in PHP and JavaScript.

Motivation

To understand BigPipe, it's helpful to take a look at the problems with the existing dynamic web page serving system, which dates back to the early days of the World Wide Web and has not changed much since then. Modern websites have become dramatically more dynamic and interactive than 10 years ago, and the traditional page serving model has not kept up with the speed requirements of today's Internet. In the traditional model, the life cycle of a user request is the following:

1. Browser sends an HTTP request to web server.
2. Web server parses the request, pulls data from storage tier then formulates an HTML document and sends it to the client in an HTTP response.
3. HTTP response is transferred over the Internet to browser.
4. Browser parses the response from web server, constructs a DOM tree representation of the HTML document, and downloads CSS and JavaScript resources referenced by the document.
5. After downloading CSS resources, browser parses them and applies them to the DOM tree.
6. After downloading JavaScript resources, browser parses and executes them.

The traditional model is very inefficient for modern web sites, because a lot of the operations in the system are sequential and can't be overlapped with each other. Some optimization techniques such as delaying JavaScript downloading, parallelizing resource downloading etc. have been widely adopted in the web community to overcome some of the limitations. However, very few of these optimizations touch the bottleneck caused by the web server and browser executing sequentially. When the web server is busy generating a page, the browser is idle and wasting its cycles doing nothing. When web server finishes generating the page and sends it to the browser, the browser becomes the performance bottleneck and the web server cannot help any more. By overlapping the web server's generation time with the browser's rendering time, we can not only reduce the end-to-end latency but also make the initial response of the web page visible to the user much earlier, thereby significantly reducing user perceived latency.

The overlapping of web server generation time and browser's render time is especially useful for content-rich web sites like Facebook. A typical Facebook page contains data from many different data sources: friend list, new feeds, ads, and so on. In a traditional page rendering model a user would have to wait until all of these queries have returned data before building the final document and sending it to the user's computer. One slow query holds up the train

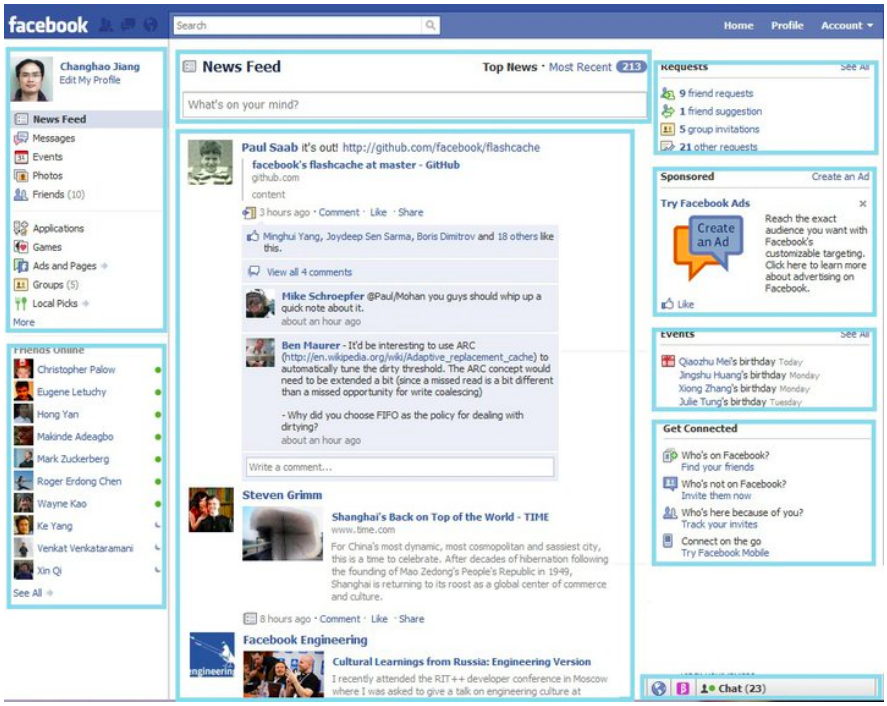
for everything else.

How BigPipe works

To exploit the parallelism between web server and browser, BigPipe first breaks web pages into multiple chunks called pagelets. Just as a pipelining microprocessor divides an instruction's life cycle into multiple stages (such as "instruction fetch", "instruction decode", "execution", "register write back" etc.), BigPipe breaks the page generation process into several stages:

1. Request parsing: web server parses and sanity checks the HTTP request.
2. Data fetching: web server fetches data from storage tier.
3. Markup generation: web server generates HTML markup for the response.
4. Network transport: the response is transferred from web server to browser.
5. CSS downloading: browser downloads CSS required by the page.
6. DOM tree construction and CSS styling: browser constructs DOM tree of the document, and then applies CSS rules on it.
7. JavaScript downloading: browser downloads JavaScript resources referenced by the page.
8. JavaScript execution: browser executes JavaScript code of the page.

The first three stages are executed by the web server, and the last four stages are executed by the browser. Each pagelet must go through all these stages sequentially, but BigPipe enables several pagelets to be executed simultaneously in different stages.



Pagelets in Facebook home page. Each rectangle corresponds to one pagelet.

The picture above uses Facebook's home page as an example to demonstrate how web pages are decomposed into pagelets. The home page consists of several pagelets: "composer pagelet", "navigation pagelet", "news feed pagelet", "request box pagelet", "ads pagelet", "friend suggestion box" and "connection box", etc. Each of them is independent of each. When the "navigation pagelet" is displayed to the user, the "news feed pagelet" can still be being generated at the server.

In BigPipe, the life cycle of a user request is the following: The browser sends an HTTP request to web server. After receiving the HTTP request and performing some sanity check on it, web server immediately sends back an unclosed HTML document that includes an

HTML <head> tag and the first part of the <body> tag. The <head> tag includes BigPipe's JavaScript library to interpret pagelet responses to be received later. In the <body> tag, there is a template that specifies the logical structure of page and the placeholders for pagelets. For example:

After flushing the first response to the client, web server continues to generate pagelets one by one. As soon as a pagelet is generated, its response is flushed to the client immediately in a JSON-encoded object that includes all the CSS, JavaScript resources needed for the pagelet, and its HTML content, as well as some meta data. For example:

```
<script type="text/javascript">
big_pipe.onPageletArrive({id: “pagelet_composer”, content=<HTML>,
css=[..], js=[..], …})
</script>
```

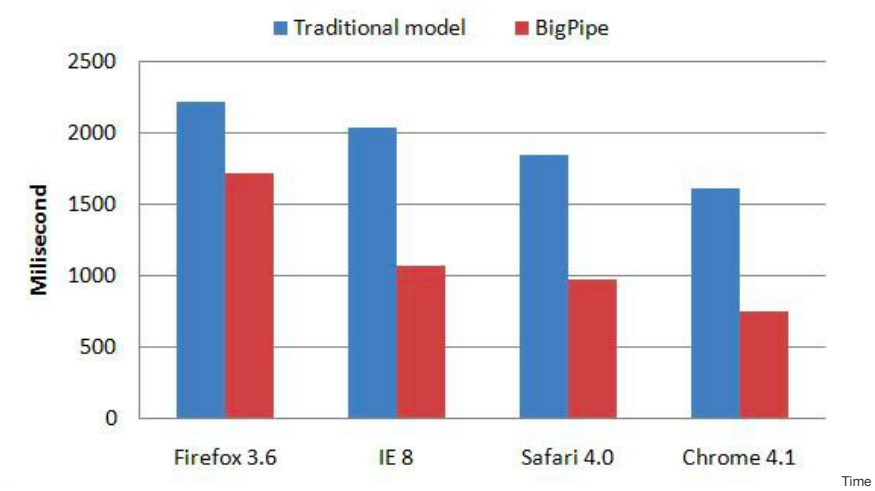
At the client side, upon receiving a pagelet response via “onPageletArrive” call, BigPipe's JavaScript library first downloads its CSS resources; after the CSS resources are downloaded, BigPipe displays the pagelet by setting its corresponding placeholder div's innerHTML to the pagelet's HTML markup. Multiple pagelets' CSS can be downloaded at the same time, and they can be displayed out-of-order depending on whose CSS download finishes earlier. In BigPipe, JavaScript resource is given lower priority than CSS and page content. Therefore, BigPipe won't start downloading JavaScript for any pagelet until all pagelets in the page have been displayed. After that all pagelets' JavaScript are downloaded asynchronously. Pagelet's JavaScript initialization code would then be executed out-of-order depending on whose JavaScript download finishes earlier.

The end result of this highly parallel system is that several pagelets are executed simultaneously in different stages. For example, the browser can be downloading CSS resources for three pagelets while rendering the content for another pagelet, and meanwhile the server is still generating the response for yet another pagelet. From the user's perspective, the page is rendered progressively. The initial page content becomes visible much earlier, which dramatically improves user perceived latency of the page. To see the difference for yourself, you can try the following links: Traditional model and BigPipe. The first link renders the page in the traditional single flush model. The second link renders the page in BigPipe's pipeline model. The difference between the two pages' load times will be much more significant if your browser version is old, your network speed is slow, and your browser cache is not warmed.

Performance results

The graph below shows the performance data comparing the 75th percentile user perceived latency for seeing the most important content in a page (e.g. news feed is considered the most important content on Facebook home page) on traditional model and BigPipe. The data is collected by loading Facebook home page 50 times using browsers with cold

browser cache. The graph shows that BigPipe reduces user perceived latency by half in most browsers.

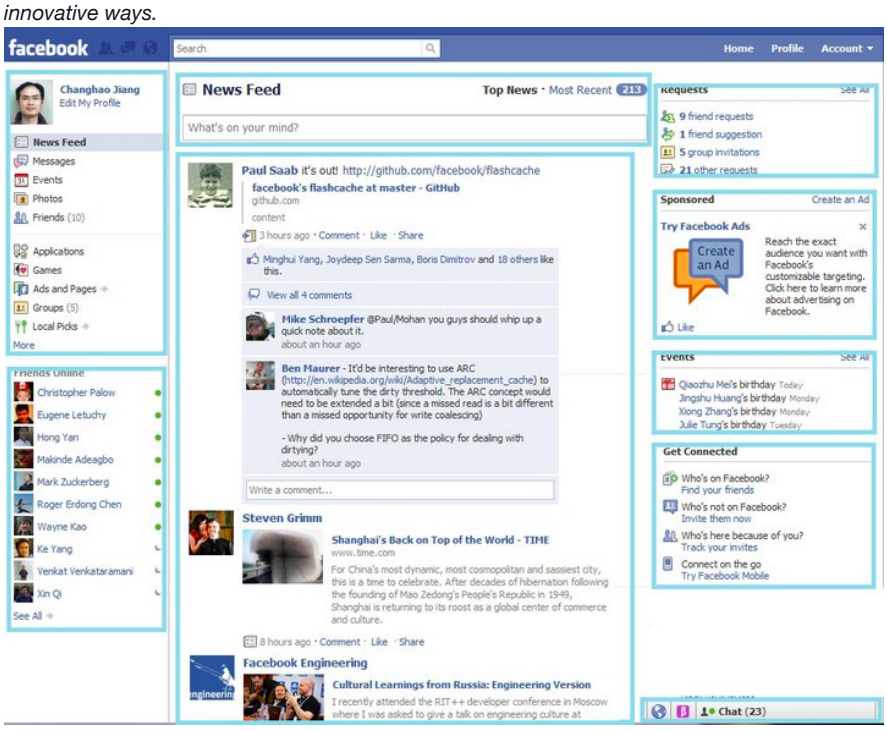


to interact latency on Facebook's home page

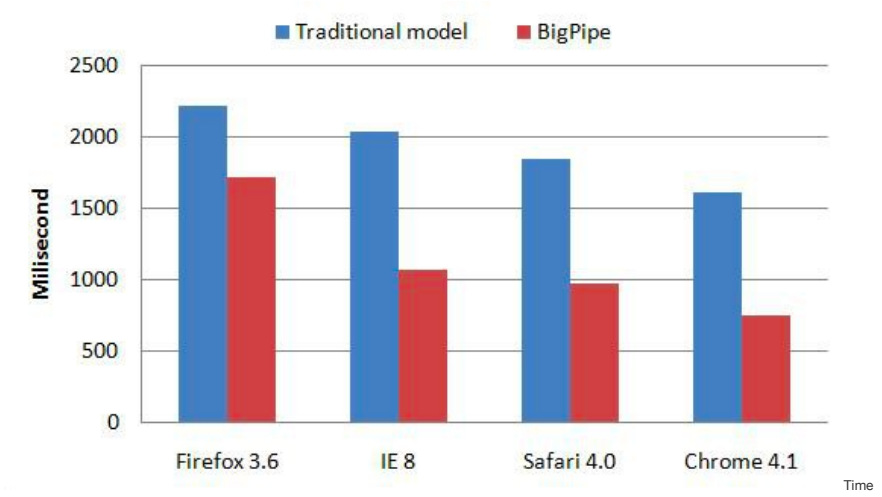
It is worth noting that BigPipe was inspired by pipelining microprocessors. However, there are some differences between the pipelining performed by them. For example, although most stages in BigPipe can only operate on one pagelet at a time, some stages such as CSS downloading and JavaScript downloading can operate on multiple pagelets simultaneously, which is similar to superscalar microprocessors. Another important difference is that in BigPipe, we have implemented a 'barrier' concept borrowed from parallel programming, where all pagelets have to finish a particular stage, e.g. pagelet displaying stage, before any one of them can proceed further to download JavaScript and execute them.

At Facebook, we encourage thinking outside the box. We are constantly innovating on new technologies to make our site faster.

Changhao Jiang is a Research Scientist at Facebook who enjoys making the site faster in innovative ways.



Pagelets in Facebook home page. Each rectangle corresponds to one pagelet.



to interact latency on Facebook's home page

Like · Comment · Share

Nico Cinefra, Tony Rogers, Norlan-Ayessa Mumar and 1,167 others like this.

139 shares

View previous comments

Ashok Yadav awesome
April 23, 2012 at 11:51am · 1

Jerry Gowen Why doesn't it work in IE8? Same system using firefox works ok. Re installed IE8 2 times and it STILL will not let me get on FB. Great new garbage....
April 29, 2012 at 10:22am · 1

Sakunthala Saminathan great article! I was looking for this kind of information for a long time !
May 3, 2012 at 6:49am · 1

Deepak Pillai I wanna do something better than BigPipe.. And am sure that i will do it!!
May 31, 2012 at 2:20am · 2

Keating Chiang There are many BigPipe problems with IE8 & IE9.
Do you know why?
Please search "bigpipe null not object facebook" on Google.
June 8, 2012 at 9:04pm · 1

Brian Mitchell Mine fails with both ie7 and ie8 . works with firefox. what is the fix?
June 10, 2012 at 11:18am · 1

Mauro Ferrá de Rosa Hi! I following this implementation, and I dont understand this very well how this work. the code load and empty page with the javascript that fire at the same time a group of request in "parallel" for every pagelet?
June 18, 2012 at 11:39am · 1

Michael Spivack Dang.
June 29, 2012 at 2:01pm · 1

Joli Aleta This is very cool.
July 4, 2012 at 2:41am · 1

Kris Oye Oh I got to be like +666 ... how special
July 14, 2012 at 12:03am

Kris Oye Seemingly cool architecture. Had to undo the like tho (666 is just to creepy). I'd like to try and benchmark a page of my own.
July 14, 2012 at 12:29am

Anthony Barone I took one for the team and am the 666th like lol
July 16, 2012 at 5:31pm

Gordon Rankin I'd really like to know what is generating the html back end. My understanding is that PHP (which I thought Facebook was built on) cannot asynchronously process multiple threads? Meaning the the pagelets are being generated one after another making the whole pipelining analogy a poor one. Or are facebook now using java or something like node.js to generate the pagelets now? I would love to know
July 20, 2012 at 9:25am · 3

Liang Yongning Shawn Huang
July 31, 2012 at 3:14am

Ben Sterrett @Gordon Rankin, "After flushing the first response to the client, web server continues to generate pagelets one by one." The pipeline analogy is still fine- you're still in multiple steps of the same process at the same time.
August 11, 2012 at 5:37am · 1

Stacy Ash Brawn WTG
September 13, 2012 at 11:37pm

Kim Chunghwan 괜히 페북이 아니구나;;;;
See Translation
September 27, 2012 at 7:07pm

Indra Jeet proved very helpful .....
October 18, 2012 at 5:16am · 1

Ali Shaheer Ejaz Great tips to make site fast as well faster
November 9, 2012 at 11:05am

Feras Jobeir good bye frames
January 25, 2013 at 1:52am · 1

Linda Shirrell why do i have errors that come up bigpipe
February 6, 2013 at 2:16pm

Hùng Lê Xuân Love this idea. Supper fast
March 28, 2013 at 12:59am · 1

Jet Li I see.. so its like asynchronous request.. as like wiki sez in instruction pipeline which is instead of waiting to finish, all stages are performed parallel.. its just like a multitasking in computer..
May 26, 2013 at 9:36am · 1

Mohamad Afiq Abdullah I want Job
June 3, 2013 at 6:33pm · 1

Subhash Patel I also will create.....so all work in javascript through
June 6, 2013 at 12:51pm · 1

Ahmad Hafiz awesome stuff
June 8, 2013 at 5:15pm · 1

Rajat Singh informative
July 28, 2013 at 8:22am · 1

Shashank Sharma Yeah it awesome....it shows concept of 0MQ messaging library in async processing of pagelets
August 22, 2013 at 1:05pm · 1

Sheikh Heera Nice read, more details expecting.
September 16, 2013 at 3:37pm · 1

Ravi Ranjan Sinha the combinaton for stealJs(for priority based fetching) along with Tiles for jsp will achieve parallelism. then how its different ? is its implementation as an engine is optimized .. can u elaborate
September 17, 2013 at 11:31pm

Erik Reppen Would love to see a blog post on how this approach has panned out over time. It's hard to tell based on looking at minimized code but it looks like it would add some maintenance/ease-of-modification tradeoffs.
November 14, 2013 at 10:34am

Ritesh Raj Now that's why FB is way above the rest and here we struggle to cater to a user base of few thousands.
January 17, 2014 at 4:27am · 1

Belajar Bahasa Inggris Kucing Lucu / Funny Cats
http://a1-weblog-9z.blogspot.com/.../Unit-Link-Terbaik-Di...
March 16, 2014 at 5:11am · 1

Ankit Patidar It is same as rendering the page let part with different ajax request at once. if we user ajax and fetch the data in small pieces after this the html markup is rendered client side then we can achieve the same
April 22, 2014 at 12:14am

Simon Tadros is this tech open source ?
April 29, 2014 at 10:44am · 1

Cahya Putra Great...

I love it, it's mean this tech can increase request/second of web server.
June 13, 2014 at 8:19pm

Don Marcony Neves Facebook Engineering
June 18, 2014 at 2:12pm

Fakhru Uddin Ahmmed I am working on special services for that i need an web browser which can connect to very busy web server faster, can i have the contact address of yours , if it is possible
August 13, 2014 at 4:40am

Gil Padi Multiple server request bot lower and higher tier system interface. mobile, tablet, computer, etc..
August 27, 2014 at 2:59pm

Tamuno-emi Amachree lovely
October 10, 2014 at 6:24pm

Yanis Juod j45
October 18, 2014 at 2:17am

RamaKrishna Chowdary Very Very Good Concept ...!
November 4, 2014 at 3:57am · 1

Jae Task This is cool but why wait for all pagelet's to complete before downloading CSS and JavaScripts? Also, why should a module that has its content, CSS and JavaScript not be able to execute before the others have completed? It seems to be wasting some time... See More

January 21 at 7:59am

Eliezer how can I implement this in my website, VK.com? j/k xD
casacristorey.com

Welcome! | VK

VK is the largest European social network with more than a 100 million active users. Our goal is to keep old friends, ex-classmates, neighbours and...

VK.COM

February 17 at 8:13pm · 1

Ali Gajani Gordon Rankin: NodeJs is being used
March 30 at 1:15pm