2013 IEEE Seventh International Symposium on Service-Oriented System Engineering

Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture

Jiang Zhu, Douglas S. Chan, Mythili Suryanarayana Prabhu, Preethi Natarajan, Hao Hu and Flavio Bonomi
Cisco Systems, San Jose, CA, USA 95134
Email: {jiangzhu, dougchan, mysuryan, prenatar, hahu2, flavio}@cisco.com

***Abstract*—In this paper, we consider web optimization within Fog Computing context. We apply existing methods for web optimization in a novel manner, such that these methods can be combined with unique knowledge that is only available at the edge (Fog) nodes. More dynamic adaptation to the user's conditions (eg. network status and device's computing load) can also be accomplished with network edge specific knowledge. As a result, a user's webpage rendering performance is improved beyond that achieved by simply applying those methods at the webserver or CDNs.**

I. Introduction

Recently, web developers and researchers began focusing on optimizing the speed at which websites load. Clearly, the advantage of website performance optimization is to improve the end-users' experiences, which can lead to greater traffic to a website and thus increases its value.

There are numerous factors influencing the performance of a web page over the Internet. While it was previously believed that page load time was determined primarily by the server, however, research now validates that 80-90% [1] of load-time issues actually occur at the front end, i.e., during the rendering and execution of a webpage by the end-user's browser. Moreover, these issues can be prevented by many straightforward optimization of the webpage, for example:

- Smarter organization of stylesheets and scripts in the page composition
- Minimizing size or the number of HTTP requests for objects on a webpage
- Maximizing cache usage by diligently and strategically specifying a web object's expiration time, etc.

Such issues and the methods to avoid them are becoming well known now. Researchers have even formulated them into a set of informal "rules" to abide by, like in [1] and [2]. A survey on the Internet will also show many companies are offering free or paid website optimization services and tools that implement some or all of these rules to measure webpages' performances and identify weaknesses for improvement, like Yahoo!'s YSlow [3] and Google's PageSpeed Service [4]. Note that, at this time, it appears these optimization tools rely on website developers to manually submit their websites to be optimized. Moreover, after their optimization is executed on the server side and optionally stored in CDNs for efficient dispersion, the improved web content no longer changes and is not dynamically optimized based on a requesting user.

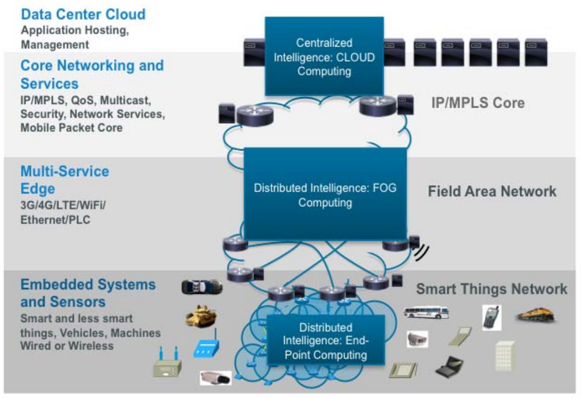
Data Center Cloud
Application Hosting, Management
Core Networking and Services
IP/MPLS, QoS, Multicast, Security, Network Services, Mobile Packet Core
Multi-Service Edge
3G/4G/LTE/WiFi/ Ethernet/PLC
Embedded Systems and Sensors
Smart and less smart things, Vehicles, Machines Wired or Wireless
Centralized Intelligence: CLOUD Computing
IP/MPLS Core
Distributed Intelligence: FOG Computing
Field Area Network
Smart Things Network
Distributed Intelligence: End-Point Computing

Fig. 1. Fog Computing architecture

In other words, there're presently no concept of automatically and dynamically optimizing a webpage based on information about a client that are already known or can only be accurately measured at devices local to or near the client's network. For example: The local network conditions or traffic statistics that are monitored by a client's local network switch node/gateway. Or, a client's feedback data that would become stale after traversing beyond a mere few hops from the local network. Or, cached contents in an edge server can be used to directly re-compose the requested files, without waited to be fetched from a remote data center. Therefore, current website performance optimization does not yet leverage advantageous properties of a ***Fog Computing architecture***.

As we have proposed in [5] and [6], the emerging Fog Computing architecture is a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud Computing data centers, typically, but not exclusively located at the edge of network. Fig. 1 presents this idealized information and computing architecture and illustrates an implementation of Fog Computing. We call it "Fog", simply because fog is a cloud close to the ground. Our novel idea of automating web site performance optimization at edge servers is the central concept and position proposed by this paper.

The rest of the paper is organized as follows: We provide in Section II an overview of Fog Computing paradigm, delineate its characteristics, and those of the platform that supports Fog services. Section III takes a close look at website performance optimization rules. Then, in Section IV we discuss ways that our proposed concepts are implemented with edge servers.

978-0-7695-4944-6/12 $26.00 © 2012 IEEE
DOI 10.1109/SOSE.2013.73

II. The Fog Computing Architecture

In the Cloud Computing model, web applications and information processing are centralized at data centers in the core of the Internet. While there are numerous technical and economical advantages with this model, it becomes a problem for latency-sensitive applications, which require nodes in the vicinity to meet their delay requirements. An emerging wave of Internet deployments, most notably the Internet of Things (IoTs), requires mobility support and geo-distribution in addition to location awareness and low latency. A new platform is needed to meet these requirements; a platform we call Fog Computing [5] [6], or, briefly, Fog,

Instead of cannibalizing Cloud Computing, Fog Computing enables a new breed of applications and services, and that there is a fruitful interplay between the Cloud and the Fog, particularly when it comes to data management and analytics. Compute, storage, and networking resources are the building blocks of both the Cloud and the Fog . This notion of "Edge of the Network", however, implies a number of characteristics that make the Fog a non-trivial extension of the Cloud. We list them with pointers to motivating examples:

- Edge location, location awareness, and low latency. The origins of the Fog can be traced to early proposals to support endpoints with rich services at the edge of the network, including applications with low latency requirements (e.g. gaming, video streaming, augmented reality).
- Geographical distribution. In sharp contrast to the more centralized Cloud, the services and applications targeted by the Fog demand widely distributed deployments. The Fog, for instance, will play an active role in delivering high quality streaming to moving vehicles, through proxies and access points positioned at the network edge, near the users.
- Large-scale sensor networks to monitor the environment, and the Smart Grid are other examples of inherently distributed systems, requiring distributed computing and storage resources.
- Support for mobility. It is essential for many Fog applications to communicate directly with mobile devices, and therefore support mobility techniques, such as the LISP protocol, that decouple host identity from location identity, and require a distributed directory system.
- Real-time interactions. Important Fog applications involve real-time interactions rather than batch processing.
- Predominance of wireless access. The wireless access points or cellular mobile gateways are prime examples of a Fog network node.
- Support for on-line analytic and interplay with the Cloud. The Fog is positioned to play a significant role in the ingestion and processing of the data close to the source.

III. Website Performance Optimization Rules

We now provide an overview of website performance optimization rules, those from [1] and [2], and specifically those that we have identified to be most applicable in our setting. They are organized according to similarity in concept.

A. Minimizing HTTP Requests

The more HTTP requests by a webpage will incur the more network round trips. This will result in slower rendering or execution of a website. Typically a webpage may spawn additional HTTP requests for images, style sheets (eg. CSS), script files (eg. Javascript).

- When there are multiple stylesheets required by a webpage, combine them into one stylesheet file before transmitting to the user. The same applies to Javascripts (JS) for a webpage.
- When there're multiple CSS background images required by a webpage, then combine them into one image, as in CSS Sprites [1]. After that, modify the originating webpage to call the images from the CSS sprite.
- For inline images, embed the content of an HTTP response in place of an URL.
- Modify the webpage such that all the embedded objects have an associated Expiration Header. This will prevent the client's browser from requesting objects that are still ”fresh” too often.
- Specific to our framework, the expiration time can be set according to the Fog server cache's expiration rules
- Examine the HTML file and check for all HTTP links for redirections. If there're any, replace them with the original full URL.

B. Minimizing Size of Web Objects

- Apply compression (eg. gzip) to all HTML, script and stylesheet files to be sent to the user and indicate in the HTTP Response's Content Encoding field that "gzi" has been applied
- Perform a procedure known as "minification". For example, examine the JS and CSS code and determine if there're unnecessary characters from code (eg. comments, white space, etc.), if so, then remove them
- Perform obfuscation: For JSs, reduce length of symbol names (eg. as in the procedure known as "munge")
- Graphics optimizing: crushing PNGs, stripping metadata, optimizing GIF animations, progressive JPEGs, etc. [2].

C. Reorganizing the Webpage Composition

The location of various components within the webpage file can significantly affect the speed at which the page is executed.

- Modify the webpage such that the stylesheet is now located at the top of the file, allowing progessive rendering.
- Modify the webpage such that the scripts are now located at the bottom of the file, thus allowing parallel downloads.
- Certain scenarios can benefit from having the scripts and stylesheet files located either inline with the HTML webpage or externally (in a separate file). We describe these conditions in further details in Section IV-C.

IV. Improving Website Performance Via Fog Boxes

We now describe methods and examples on employing the optimization rules of Section III at the edge nodes in a Fog Computing setting. Fig. 1 illustrates this overall architecture.

Users are connected to the Internet via the edge servers. All web requests that the user makes first goes through the edge (or Fog) servers, which subsequently obtain them from the core network where the web servers reside and potentially modify and locally cache these files.

A. Overall Process Flow

There're two possible process flows: (1) optimizing webpage for its initial request and (2) optimizing webpage for its subsequent request(s). We first describe the latter (2) and it will become apparent that (1) involves a subset of procedures in (2).

Consider a client is requesting a webpage that has previously been requested by another client connected to the same edge server. As a result, this webpage and all of the objects embedded therein have already been loaded from the web server of the cloud and cached locally in the edge server's cache. The overall process flow is as follows:

1. The edge server introspects the HTTP request made by its client and detects that all the files requested are available in the cache and that they have not lived beyond the expiration time
2. The edge server analyzes the requested webpage based on a predefined set of web-performance-optimization rules and determines which rules are applicable.
3. If there are rules applicable, the edge server makes modification to this webpage's composition according to the applicable rules. These modifications may also include incorporating into the webpage any of the cached files (and versions of them modified according to the applicable rules) associated with the webpage.
4. The edge server then sends the modified webpage and cached files to the client. The edge server also caches these modified files so that they can be sent to subsequent requests from users as well.
5. The client's browser executes this webpage and receives the same content and viewing experience as the unmodified version but with faster load and response time.

Now, for the process flow (2), consider a webpage that has not been previously requested by a client belonging to the same Fog server. (Or that the cached version of it has been expired.) The difference here is that, because the webpage and its files are not cached already, there could be limitations to serve them to the user without waiting for all relevant files to be downloaded, as in the store-and-forward mode.

Consequently, the Fog server can only examine partial segments of the HTML page as it arrives, before it is sent to the user already. As a result, optimization on the entire webpage is unlikely to be feasible. The Fog server can, however, optimize the incoming portions of the webpage based on portions that have been examined already. Therefore, not all optimization rules can be executed in this scenario. But apparently the optimization rules applicable with process flow (2) is a subset of all the rules used in process flow (1).

We should point out that our process currently assumes both the end-users and the websites agree to our optimization of the webpages. In the future, perhaps when webpage optimization becomes prevalent, we can consider adding a header to HTTP for indicating whether optimization is preferred, something similar to the Accept-Encoding header that indicates content compression options.

B. Dynamic and Custom Optimization Based on Client'S Device and Local Network Conditions

Unlike the web server in the Cloud that is separated from the user by wide area networks (WAN), an Fog server has the distinct advantage of knowing the network conditions local to an end user. For example, the edge server knows if the local network is experiencing congestion. Or for example, a wireless access point (AP) is aware of the conditions of the wireless channel between it and the clients, for which affects the data rate that can be used.

Our method of how to use this knowledge is one of the novelties of our invention. Specifically, the edge server can use this information to customize the optimization for its users.

In one example, consider a wireless client that is situated near an AP and that there's no network congestion. Consider also that this client is not a mobile device, but a PC/laptop that has a large display screen. Since there is enough bandwidth to support transferring the webpage and all its objects in a timely fashion, there is no need to shrink the sizes of the graphic files that typically leads to lower viewing resolution. (On the other hand, we should still do so, if it's recognized that this client is a mobile device with a small display screen, which makes it hard for the user to appreciate high resolution graphics.)

Now consider this wirless PC/laptop user has roamed to an area with poor reception, leading to a decrease in data rate with the access point. Consider also that this information is sent to the edge node serving the users. (And of course, this operation can also be performed directly on the access point, which is a prime exmple of an edge node.) At this point, in order to timely deliver the webpage and its objects to the user, the edge server can choose to send graphics of a lower resolution to this user. Then, we have eliminated the user's wait for graphics to be downloaded slowly when network conditions are unfavorable. Note that this example also applies to the case when there is a local network congestion instead of poor wireless reception.

C. Per User Optimization for Inline or External Scripts

An optimization rule often has parameters that represent various tradeoffs. Knowledge of the user's browsing behavior can help dynamically select the best parameters. An example of this is deciding whether to make JS and CSS inline or external. As discussed above, advantage of an inline JS or CSS is that HTTP requests are minmized; but if they're external files, then the browser could have cached and reused them when another webpage contains the same JS and CSS. (A website typcially has an unifying theme or style, so webpages therein usually share the common CSS and JS.)

It is pointed out in [1] that if during a session the page views of an user for the same website is high and that the website has common JS and CSS across its webpages, then performance is improved with the external approach. This is because that user's browser would have cached the JS and CSS files.

The problem then is how to identify each user. One well known method employed by websites is to track each user by a cookie. But while a cookie can be automatically piggybacked in HTTP requests for webpages with dynamic content, cookies are almost never present for static pages in order to conserve the size of each HTTP request. As such, it is not feasible to rely on cookies to track users who request static pages. Another method for tracking users is for a webserver to track an user via its IP address. But if a user belongs to a network that is behind a firewall, then a user will have the same IP address as any other users from this network requesting the same webpages. Thus, this tracking method is also unreliable.

Fortunately, we can solve these issues in a Fog architecture. This is because the Fog server belongs to the same network as the users. So the Fog server can have knowledge of each user based on their (unique) MAC addresses or the Fog server can observe each user via their (unique) local IP addresses.

In summary, the steps to realize our idea is as follows:

1. The Fog server keeps track of each users' website requests, via either the user's local IP address or MAC address.
2. When a tracked user requests more than N number of webpages from the same website, where N is a tunable parameter and typically it should be set to 2. This indicates the user is exploring webpages under this website and can benefit if the user's browser can cache the common CSS or JS files. So, from then on, this website's webpages will have their CSS or JS sent externally for this user.

D. Obtaining and Applying User Experience Feedback

Measuring a user's webpage rendering speed: It is possible to embed code snippets in an HTML or script file such that when the browser has rendered to that spot of the file, it sends a feedback message to the webserver[1]. By strategically placing these code snippets, a webserver has the ability to measure the speed at which a user's brower loads a page.

However, an accurate measurement can only occur if the webserver and the user are not separated by other networks, as in a WAN. Because a hop from a network to another can incur heavy delays, thus adding variance to the measurement. On the other hand, if the webserver is located in the same local network as the user, then the rendering time can be accurately measured. This neccessary scenario can occur in our fog architecture, where the Fog server and the users are on the same network. For example, when the edge server is a wireless AP serving the user.

Applying knowledge of a user's webpage rendering speed: Having accurate knowledge of an user's webpage rendering speed can customize the optimization for each user even more dynamically and fine tune the result achieved via the above rules. Here is an example:

Consider a situation when the Fog server has determined a user has a large screen device and has enough network bandwidth for larger and higher resolution graphic files. As discussed in Section IV-B, the Fog server will send such files and expect the user's browser to render them in a timely fashion. In addition to doing so, the Fog server will also embed code into the webpage to measure the user's browser rendering speed. In particular, a check is put on components that require more processing, like scripts or large graphic files.

If the feedback returned from the user indicates the browser is displaying or executing these components sluggishly, then this may indicate the user's device may be low on computing power or other resources. Perhaps this is because the user has many applications opened, thus using up the device's memory and computing resources. Or it could be that the user's current batch of applications are more demanding. When this occurs, in subsequent webpage requests, despite the fact that there's enough bandwidth in the network, the Fog server will select to send a smaller lower resolution graphic file, thus minimizing the computation costs on the user and expediting the webpage's rendering, which is the most important objective with webpage optimization.

The Fog server will continue to monitor feedbacks from the user and determine if the browser's rendering speed is picking up, at which time the Fog server can progress to sending larger higher resolution graphic files again.

V. Conclusion and Discussion

There are many straightforward methods for optimizing webpage performances. We apply these methods in a novel manner within a Fog Computing architecture, such that these methods can be combined with unique knowledge that is only available at the edge (Fog) nodes. More dynamic adaptation to the user's conditions (eg. network status and device's computing load) can also be accomplished with network edge specific knowledge. As a result, a user's webpage rendering performance is improved beyond that achieved by simply applying those methods at the webserver or CDNs. We are currently applying our proposed concepts to develop a proof-of-concept system and we plan to report additional follow up results in a future paper.

References

[1] S. Souders, *High Performance Web Sites: Essential Knowledge for Frontend Engineers*. O'Reilly, 2007.

[2] ——, *Even Faster Web Sites: Performance Best Practices for Web Developers*. O'Reilly, 2009.

[3] Yahoo! YSlow. [Online]. Available: http://developer.yahoo.com/yslow/

[4] Google PageSpeed Service. [Online]. Available: http://developers.google.com/speed/pagespeed/service

[5] F. Bonomi, "Cloud and Fog Computing: Trade-offs and applications," in *Intl. Symp. Comp. Architecture (ISCA), EON Workshop*, Jun. 2011.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its role in the Internet of Things," in *ACM SIGCOMM Mobile Cloud Computing (MCC)*, Aug. 2012.

[1]For example, this can be done straightforwardly with the Add Instrumentation function of PageSpeed [4].