Latency Amplification: Characterizing the Impact of Web Page Content on Load Times

Cătălin Avram, Kenneth Salem, Bernard Wong
Cheriton School of Computer Science, University of Waterloo

***Abstract*—Web users like sites that load quickly. Longer web page load times translate to reduced user satisfaction and loss of revenue and mindshare. The time required to load a given web page is difficult to predict because it is a complex function of many factors, such as the latencies associated with the network requests used to retrieve that content from remote servers. However, one of the most important factors is the page content, including the scripts, images, style sheets and other objects that are present on the page. In this paper we propose a simple metric for characterizing the content of a web page in terms of its impact on page loading times. This metric, called the latency amplification factor (LAF), characterizes the content of a web page in terms of how it affects the page load time. The LAF of a web page can be estimated quickly and easily, and we describe a lightweight method for doing so. In addition, we propose an extended version of the basic LAF metric, called CLAF, that relates page load time to underlying request latencies in the presence of content delivery networks. We estimated LAFs for a variety of popular web sites, and found that they varied substantially. To validate our approach for estimating LAFs, we compared estimated LAFs against measured LAFs and found that our methodology, though simple, gave reasonably accurate estimates.**

***Keywords*-performance modeling, network latency, web application performance.**

I. Introduction

The web has, in recent years, become the dominant platform for deploying large-scale applications. The shift from a mostly static information distribution medium to an application platform has led to significantly more complex web pages, which in turn has increased page load time. Several recent studies [1], [2] have found that even a small increase in web page load times can substantially increase the likelihood that a customer would switch to a competing service or store. Therefore, it is critically important for web-based companies to understand what contributes to page load time.

Factors that affect page load times include the HTML and JavaScript processing time within the browser, as well as the network latency between clients and servers. These factors are amplified by the need of complex web pages to load tens or hundreds of objects from multiple servers. These objects may include scripts, style sheets, images, and other types of content. Furthermore, the structure of modern web pages often introduces object loading dependencies, where certain objects are only retrieved after another object has been retrieved and, if the object is HTML or JavaScript, parsed and/or executed. Because of this amplification, a user with high network latency to the web server may experience orders of magnitude worse page load times than a user with low network latency.

Given the importance of page load times to user satisfaction and the strong impact of web page content on load times, it is important to understand exactly how web page content affects page load time. In this paper, we take a step in this direction by proposing a simple content metric - a means of characterizing the content of a web page and the impact of that content on page load times. One possible approach to this task would be to use some kind of "syntactic" metric, such as a count of the number of component objects on the page. However, since our real interest lies in the effect of page content on the page load time, the metric that we propose - which we call the *latency amplification factor (LAF)*, is behavioral rather than syntactic.

The LAF for a web page can be interpreted as the answer to a simple hypothetical question: if the individual request latencies between the client and the servers that hold the web page content were to increase by a constant factor, by what factor would the total load time for the web page increase? In other words, the LAF of a page is a measure of the sensitivity of a page's load time to changes in underlying request latencies for the objects on that page. A simple, small, plain HTML page that does not have any component objects would have a LAF of 1. More complex pages, with many interdependent objects, will have higher LAFs. A key feature of this metric is that it is simple and easy to compute.

This paper makes several contributions. First, we introduce the LAF metric, which captures the relationship between network latency and web page load time as a single number. Second, we present a lightweight methodology for estimating the LAF for a given web page using a page retrieval log that can be generated by any modern browser. Third, we estimate the LAFs of a selection of popular websites, and compare them to the actual increase in page load time from injecting a controlled amount of synthetic latency between the downloading client and the content servers. Finally, we consider an extended version of the LAF metric, which we refer to as *core LAF*, or CLAF. This metric distinguishes objects retrieved from content delivery networks (CDNs) from those fetched from non-CDN, or core, servers. By comparing a page's LAF and CLAF, we can characterize how effective the CDN is at reducing the page's load time.

II. Web Latency

Modern web pages are rich and complex. Each HTML, JavaScript, and CSS object on a website may reference many other objects. Thus, a browser loading a web page will have to issue many server requests to retrieve those objects. Furthermore, the presence of inter-object references, or dependencies,

means that the browser must retrieve and parse the parent object, at least partially, before it can determine the objects that the parent refers to and issue requests for those objects.

We will use a very simple example web page to illustrate this process. The web page displays a single user-clickable button over a background image. As the button is clicked by the user, the background rotates through a sequence of three possible images.

Loading a web page generally begins with the user initiating a page load by clicking a link or manually entering an URL. The browser then retrieves the requested web page. In this example, the user requests the page http://www.example.com/example.html, the contents of which are shown in Figure 1. We will assume that browser has previously resolved the `example.com` domain name, and can immediately fetch the requested web page from the remote server. Once the user's browser downloads and parses the root object (`example.html`), it will be able to identify references to a CSS stylesheet and two different script files.

```
<html lang="en">
  <head>
    <link rel="stylesheet" href="example.css"/>
    <script src="//code.jquery.com/jquery.js"/>
    <script src="example.js"/>
  </head>
  <body>
    <button>Change Background</button>
  </body>
</html>
```

Fig. 1. `example.html`: A simple web page with a single button.

The stylesheet (`example.css`) is shown in Figure 2. It is responsible for loading a background image for the web page. The stylesheet is dependent on an image file (`fruit.jpg`), which the browser must retrieve by issuing another asynchronous request. This example also illustrates that the various objects on which the root page (`example.html`) may come from a variety of different servers.

```
body {
  background-image:url('http://pics.com/fruit.jpg');
  background-repeat:no-repeat;
}
```

Fig. 2. `example.css`: A stylesheet specifying the background image.

As for the script files: The first (`jQuery`) is a well known JavaScript library that is used by many website developers. The second, `example.js` (not shown), attaches an event listener to the button on the page so that clicking the button will cause the background image to change.

In this paper, we are primarily interested in the *load time* of a web page. We define this to include the time to load the root web page as well as any additional objects on which the root object depends, either directly or indirectly. It is not uncommon for complex web pages to request additional objects after the page load time. This may occur, for example, in response to additional user actions (like clicking the button in our example web page), or timers in background scripts. Nevertheless, we consider a web page's load time to be the time from the initial user request for the root page until the time when all of its dependent objects have been fully retrieved, as the page is fully ready to use by the user once this occurs.

III. Latency Expansion

Our objective is to arrive at a metric to characterize the content of a web page in terms of its effect on the page's loading time. Previous work, such as WebProphet [3], has focused on building models that can predict the absolute load time for a web page, taking the page content (and many other factors) into account. In contrast to such relatively complex models, our characterization cannot, by itself, be used to predict absolute page load times. Rather, we characterize web page content by considering how much the page structure magnifies the request latencies of the individual components that make up the page. As described in Section I, we call this the latency amplification factor (LAF) for the page.

The following simple thought experiment gives some intuition for the LAF. Suppose that a browser loads a web page P, and the load time for P is $t_P(0)$ Now suppose that the browser loads the same page again, but this time the retrieval latency for every component of the page is increased by a constant amount α. Suppose that the measured page load time for this second experiment is $t_p(\alpha)$. The latency amplification factor (LAF) for P is:

$$\frac{t_P(\alpha) - t_P(0)}{\alpha} \tag{1}$$

The numerator in this expression $(t_P(\alpha) - t_p(0))$ indicates the amount by which the page load time increased, while the denominator (α) indicates the amount by which the component latencies increased. For a simple web page with a single component, the latency amplification factor will be 1. For more complex pages that contain multiple dependent components, we expect to see amplification factors greater than one. The larger the LAF for page P, the more a change in underlying latency will affect the page load time experienced by the user.

In order to estimate a page's LAF we need to extract and then analyze its *dependency graph* which represents the structure of the page contents, i.e., its components, and the relationships among those components.

A. Dependency Graph

As was illustrated by the example in Section II, loading a web page involves loading multiple objects, potentially from multiple servers. Modern browsers load multiple objects concurrently and asynchronously, however resource limits and browser settings, such as the maximum number of threads per page load, place restrictions on this process. More importantly, the structure of the web page content itself may fundamentally limit the amount of concurrency.

We use a *dependency graph* to represent the structural dependencies of a given web page. A page's dependency graph contains one node for each retrievable object referred to, either directly or indirectly, by the given page. There is a directed edge $V_1 \rightarrow V_2$ in the dependency graph if the object V_1 refers to the object V_2. Dependency graphs are rooted directed graphs, with the root node representing the web page which the dependency graph represents. Figure 3 illustrates the dependency graph for our example in section II.

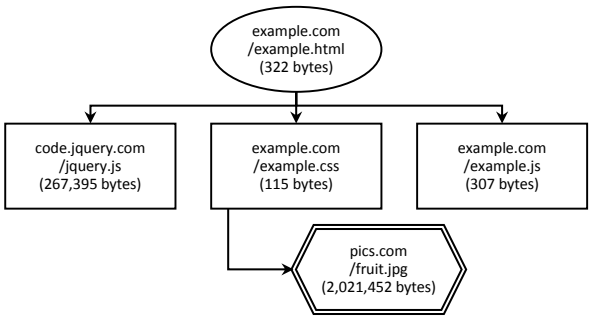

Fig. 3. Dependency Graph for example.html from Section II

B. Producing Dependency Graphs

A number of different techniques can be used to generate a dependency graph for a given web page. One option, which we will refer to as the *white box* approach, is to analyze the web page content, along with the content of any dependent objects, to extract object references. This approach will correctly identify static references to objects. However, it requires the ability to parse and extract object references from all types of commonly used web objects. In our simple example, a white box approach must be able to parse HTML, JavaScript, and CSS objects. The main disadvantage of this approach is that, since it is based on static analysis, it will fail to detect dynamically-generated object references, which is very common especially within JavaScript objects.

Li et al [3] propose a *black box* approach, embodied in their WebProphet system, to generate a dependency graph similar to the graph that we use. WebProphet builds the dependency graph for a page by inspecting a log of object retrievals generated by the browser as it loads the page. Each log entry identifies a retrieved object, and provides some information about the timing of the retrieval, such as the retrieval request time and the request response time. To infer dependencies among the objects, WebProphet retrieves the page multiple times. On each retrieval, WebProphet uses a proxy-based technique to add artificial latency to the retrieval time of a target object from the trace. WebProphet then identifies other objects (in addition to the target) whose retrieval times are delayed, and infers that they are dependent on the target.

For our work, we instead use a lightweight black box approach to construct the dependency graph for a page. Like WebProphet, we use a browser to retrieve the web page for which we wish to construct the dependency graph, and then inspect the browser-generated request log. The log contains one entry for each object retrieved, with each entry containing the following information:

- The object's URL and total size in bytes.
- The *request time*, i.e., the time at which the request for the object was sent by the browser.
- The *wait delay*, which is the interval between the request time and the receipt of the first byte of the object.

In particular, we use logs in the standard HTTP Archive (HAR) [4] format, which contains the necessary information and can be generated natively by modern browsers such as Chrome and Firefox.

We create dependency graphs with one node for each object referred to in the log, annotating each node with the object's size and site (extracted from the URL). We assume that object V_2 is dependent on object V_1, and create the corresponding directed edge from V_1 to V_2 in the graph, if and only if V_2's request time is after the end of the wait delay for V_1.

Clearly, this simple method of creating the dependency graph will identify all actual dependencies, both static and dynamic. However, it will also identify *false dependencies*, i.e., dependencies that do not reflect the structure of the web page. For example, an edge from V_1 to V_2 may exist in the graph simply because object V_2 was loaded after object V_1, and not because of an actual structural dependency between V_1 and V_2. On the other hand, this lightweight approach is considerably simpler than the black box approach used by WebProphet and does not require multiple page retrieval iterations.

C. Estimating the LAF

Latency amplification occurs because the browser must make multiple sequential round trips to retrieve objects. If the browser must make k network round trips to some site to retrieve content and the request latency for that site increases by an amount α, then the time the browser requires to retrieve all data from that site will increase by a factor $k\alpha$ - an amplification by k of change in request latency. Thus, to estimate a LAF from a dependency graph, we want to estimate how many sequential round trips are implied by that graph.

A very basic way to estimate the number of round trips implied by a dependency graph is to assume that any objects that can be retrieved concurrently will be retrieved concurrently. Furthermore, as a starting point we can assume that retrieving each object in the graph involves one network round trip. Under these simplifying assumptions, if there is a path of length k from root to leaf in the graph then the client will require at least k sequential round trips. Since the client is assumed to retrieve the objects along different paths in parallel, the LAF (the number of sequential round trips) would be the length of longest root-to-leaf path in the dependency graph.

Although this basic approach would give a simple approximation of the LAF, it fails to take into account a variety of relevant factors, such as the sizes of the objects, the properties of the TCP connections over which the object requests are issued, and the behavior of modern browsers. To estimate the LAF from the dependency graph, we therefore enhance the basic procedure to take the following factors into account:

Object Size: The number of network packets required to retrieve an object depends on the object's size.

TCP Characteristics: The number of round trips required to retrieve an object depends on the TCP window size, which is governed by TCP's congestion protocol. Additionally, there is a fixed overhead associated with the TCP handshake required to establish a connection.

Connection Reuse: The browser can reuse a connection to retrieve multiple objects from the same site.

Our algorithm for estimating the LAF from a given dependency graph can be outlined as follows:

1) Estimate the number of round trips required to retrieve each individual node in the dependency graph.
2) For each distinct root-to-leaf path in the graph, count the total number of round-trips required along that path by summing the individual round trip counts for the nodes along the path.
3) Choose the maximum round-trip count over all root-to-leaf paths in the graph, and report that as the LAF.

To estimate the number of round trips required for each node, we use a simple model of the client's browser. Specifically, we assume that the browser will initiate the request to retrieve a node as soon as that node's predecessors have been retrieved, and that there is no limit on the number of concurrent requests that the browser may have outstanding. These assumptions will of course lead to inaccuracies in our model. However, they also ensure that the model is browser and machine independent, which is critical as most large-scale websites have very diverse clients. The number of round trips required to retrieve an object thus depends on the size of the object and on the state of the TCP connection (specifically, on the TCP window size) at the time the object is retrieved.

Consider a node in the graph with size s bytes, and suppose that the TCP window size for the connection used to retrieve the node is of size w_{start} bytes when retrieval starts. The value of s for each node can be found in the dependency graph, and we will explain shortly how w_{start} is determined. We estimate r, the number of round trips required to retrieve this node, as:

$$r = \left\lceil \log_2 \left(\frac{s}{w_{start}} + 1 \right) \right\rceil \tag{2}$$

The factor of $\log_2$ in this expression arises from the fact that the TCP window size doubles with each round trip until the object has been completely retrieved. We can also define w_{finish}, which indicates what the TCP connection's window size will be after the node has been retrieved. If $s \leq w_{start}$, then no window size doubling will occur and w_{finish} will be the same as w_{start}. Otherwise, the window size will double one or more times during the retrieval of the node. Thus, the final window size can be determined using the following expression.

$$w_{finish} = 2^{r-1} w_{start} \tag{3}$$

where r is the round trip count for the node, as defined by Equation 2. This model assumes that objects are relatively small, and that most users will finish retrieving their objects before reaching their actual maximum window size.

The value of w_{start} for a node depends on which TCP connection is used to retrieve the object. A node's w_{start} will depend on whether that node has any ancestors in the dependency graph that are retrieved from the same site. We consider two cases. If there are no such ancestors, then we set $w_{start} = 4440$ bytes, which corresponds to TCP's default initial window size for new connections. On the other hand, if there are one or more such ancestors, then the browser will have one or more TCP connections already open to the target site. In this case, we conservatively set the node's w_{start} to the maximum w_{finish} among all of its same-site ancestors. This is a conservative choice because choosing the largest w_{finish} results in the smallest possible estimate for the number of round trips (Equation 2) for the current node, which in turn gives the smallest possible estimated LAF.

IV. LAF Examples

We have chosen a set of ten popular web pages for testing. Most of our choices appear among the top ten most visited web sites in the Alexa top 500 global sites ranking [5]. Our set includes a variety of different types of sites, including search engines, web portals, a social network, and e-commerce, news, and media delivery sites.

For each chosen web page, we used the PhantomJS [6] web browser to retrieve the page while capturing a request log. In all cases, the web client was located at the University of Waterloo, in Canada. Using these logs, we generated dependency graphs and then estimated the LAF for each page. We repeated this process 20 times for each web page. Figure 4 shows the mean estimated LAF for each page. (The figure also shows measured LAFs, which we will discuss in Section V.)

Except for GMail, we did not see a wide variation among the 20 different estimated LAFs for each web page. Each LAF was generated using a different retrieval log, and we expect that different retrieval logs (for the same page) should include different request timing information because of natural variations in the request times and wait delays across the different runs. Nonetheless, such variations had only a small effect on the LAF estimates.

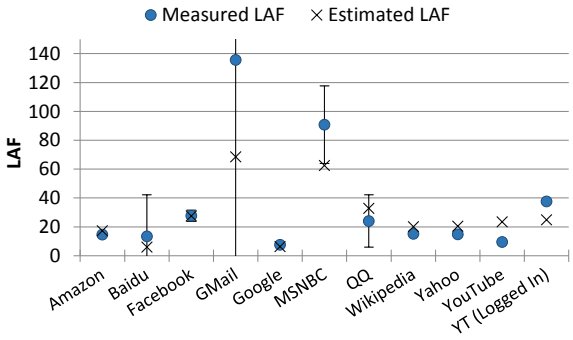

Fig. 4. Mean measured and estimated LAF for different web pages. Error bars show the 90% confidence interval around the measured mean.

In this experiment, we observed a wide range of LAFs as illustrated in Figure 4. Not surprisingly, the search engines (Google and Baidu), with their simple pages, had the lowest LAFs. At the other end of the spectrum were GMail (webmail) and MSNBC (news page), which had LAFs an order of magnitude higher than those of the search engines. Previous work [7] has suggested that news web pages often load objects from a large number of hosts. This was certainly true for the MSNBC front page, which has references to objects from almost 60 distinct hosts. The other websites we tested typically only reference about 10 hosts, and the search engines reference only 3. The large number of hosts translates to a lot of separate TCP connections, which, together with small initial congestion window size values, account for the large LAF value for the MSNBC page. In contrast to MSNBC, GMail's

high LAF value is due to its high structural complexity, which is manifested as a long critical path in its dependency graph.

The remaining sites we tested are between these extremes, with LAFs from just under 20 to just over 30. These LAFs, though much lower than those of GMail and MSNBC, are still very significant. For example, with a LAF near 30, a 30 ms change in network latency results in an almost 1 second change in user-perceived page load time.

V. LAF Model Validation

In order to validate our LAF estimation methodology against empirical data, we measured web page loading times while manipulating the latency between our browser and the web-servers. By comparing such measurements against baseline measurements with no added latency, we can determine the actual LAF of a web page. We then compare these measurements to the estimates made using the methodology from Section III.

Suppose that $t_P(0)$ is the measured baseline page load time for a web page P, with no added latency. Similarly, suppose that $t_P(\alpha)$ is the page load time for P if we introduce an additional latency $\alpha/2$ to each incoming and outgoing network packet between the browser and the webservers, for a total additional round trip latency of α. The *measured* LAF for P can then be calculated using Equation 1.

To introduce network latency, we used the network traffic control tool (`tc`) and the technique described by Nussbaum and Richard [8] to allow us to add latency to both incoming and outgoing traffic.

Figure 4 shows a comparison of measured and estimated LAFs, with $\alpha = 100$ms. for the measured LAFs. We repeated each page load measurement 20 times. For each web page we measured, the figure shows the mean estimated LAF, the mean measured LAF, and a 90% confidence interval around the measured LAF. For most of our test pages the estimated LAF was reasonably close to the measured value, which suggests that our estimation methodology is capturing the important factors that contribute to page load times.

For several sites, including Baidu, QQ, MSNBC and GMail, there was significant variance in measured page load times, resulting in relatively large confidence intervals around the mean measured LAF. For Baidu and QQ, both of which include a substantial amount of content served from China, this is due to the relatively long network path from those servers to our measurement point in Canada. MSNBC's content is served from closer sites, but as we have previously noted, that page includes content from a large number of servers, delays to any one of which can affect our measured page load times. GMail uses a single long request to download a substantial and dynamically adjusted amount of data in the background to the client. In addition to the substantial variance this introduces into our measured LAF, such requests also highlight a challenge for our LAF estimation methodology: In estimating the LAF, we include all objects on which the root page depends. However, the page may be usable well before all of those objects have been retrieved.

VI. Accounting for CDNs

We estimate a page's LAF by asking what would happen to the page load time if all of the objects on which a page depends took longer to retrieve. However, in some cases, we may wish to focus only on certain objects, and ask how sensitive the page's load time is to the retrieval time of those objects. For example, many web service providers make use of content delivery networks (CDNs) to cache static page content at the edge of the network, closer to end users. For a web page from such a provider, we may wish to characterize only that part of the page that is not served by the CDN.

It is very easy to modify our methodology to answer such questions. In the remainder of this section, we show how to estimate the *core latency amplification factor (CLAF)* of a page. The CLAF is similar to the LAF, but it only considers the effect of page content that is not served by CDNs. A page's LAF can be interpreted as an estimate of the number of sequential network round trips that will be required to load that page, while its CLAF can be interpreted as an estimate of the number of sequential network round trips to core (non-CDN) servers. Comparing the LAF and CLAF of a page provides a measure of the CDN's effectiveness for that page.

After generating the dependency graph for a web page as described in Section III-B, we tag all CDN nodes. We then use the same algorithm described in Section III-C but force the number of round trips for tagged (CDN) nodes to be 0 in order to obtain the CLAF. We use a simple pattern-matching approach to determine whether an object is served from a CDN: we query the DNS (using `dig`) for information about the host part of an object's URL and look for certain key words in the answer section of the DNS reply. If the DNS response contains any reference to well known CDNs such as "akamai" or "cloudfront" or the character sequence "cdn", we tag the node. This approach is not comprehensive, but it allows us to approximate the impact of CDNs on the amplification factor.

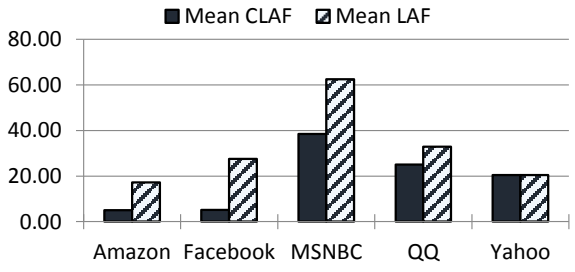

Fig. 5. LAF and CLAF for Different Web Pages

Figure 5 shows both the CLAF and LAF for several sites. The five sites presented are the only ones for which our basic approach has been able to identify the CDNs used. We know that websites under Google's administration (Google, GMail and YouTube) make use of CDNs, however the simple pattern matching approach was unable to identify the servers responsible for these services. As expected, the CLAF values are consistently lower than the LAF. For websites like Amazon and Facebook, the reduction is significant; for both websites,

all but one host are CDN nodes. With MSNBC and QQ, one third to one half of the hosts have been identified as CDNs, so the reduction is more moderate. For these two sites however, even the lower CLAF values are still quite high. This suggests that, for some sites, using a CDN may not be sufficient to ensure that a user has low page load time.

In the case of Yahoo, our methodology identified only 2 CDN hosts out of the 12 hosts it uses. However neither of the associated nodes were on a critical path when calculating the LAF, so their omission from the CLAF calculation did not affect the final value. It may be possible that other CDN hosts are being used that our methodology was unable to identify.

VII. Related Work

Butkiewicz et al [7] have presented a detailed analysis of the complexity of modern web pages. This analysis, which included more than 1700 web sites, considers how web page features are related to the time required to render the page. This analysis also introduces a regression-based model that can be used to predict the rendering time of a page given values for key page features of the page content. The model we present in this paper is complementary to this work. While our model is sensitive to the content of the page, they predict the impact of individual request latencies on the page loading time.

Much like LAF, WebProphet [3] models a page using a graph, with edges representing dependencies among the objects on the page. Its main objective is to predict the absolute load time of a web page under hypothetical conditions. To infer dependencies, WebProphet retrieves the page while adding latency to the download of specific objects. By identifying which other objects' latencies change as a result of these injections, WebProphet can infer that dependencies exist. In contrast, our proposed technique produces a much simpler model that is intended only to relate page load time to latencies of the requests for the objects on the page. Our model cannot, for example, be used to predict the effect of a change in browser configuration on page load time. Although WebProphet's model is considerably richer, this richness comes with a price: its models are larger, more expensive to build, and more expensive to use than our models. We are trying to answer a narrower yet still important question using a model that is simple and easy to use.

A 5000 broadband access network study conducted by Sundaresan et al [9] also emphasizes the importance of latency in page loading time, however their focus is on imporving the speed of establishing network connections by caching DNS records and maintaining active TCP connections within the home router.

The "What-If Scenario Evaluator" (WISE) [10] takes a machine learning approach to model the effects of changes in a web service's deployment configuration. For example, WISE was used to predict the effect on users' page load times when a CDN node serving those users was moved. WISE requires an extensive collection of monitoring data which it uses to infer dependencies among system parameters of interest, such as client response times and CDN node locations. Chen et al [11] focus on the effect of changes in the inter-tier latencies in a multi-tier service architecture. They propose a metric called the *link gradient* to capture the effect of the latency between two tiers on the overall performance of the system.

VIII. Conclusion

Although user satisfaction depends strongly on the load time of a web page, there is no simple way to characterize the relationship between the content of a page and its load time. We have proposed a simple metric, the LAF, to address this problem. This metric is easy to estimate and to interpret. We have estimated LAFs for some of the world's most popular websites. Some are surprisingly high, indicating the small changes in network latencies can lead to large increases in page load times.

We validated our procedure for estimating LAFs by measuring LAFs and comparing those measurements to our estimates. In most cases, our LAF estimations were reasonably accurate. Finally, we have considered an enhanced metric called CLAF that can distinguish web page content loaded from CDNs from other content. Comparing the LAF and CLAF for a web page gives a way of quantifying CDN effectiveness for that page.

References

[1] R. Kohavi and R. Longbotham, "Online experiments: Lessons learned," *Computer*, vol. 40, no. 9, pp. 103 –105, sept. 2007.

[2] M. Mayer, "What google knows," in *Proceedings of the Third Annual Web 2.0 Summit*, San Francisco, CA, USA, November 2006.

[3] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg, and Y.-M. Wang, "Webprophet: Automating performance prediction for web services," in *NSDI*. USENIX Association, 2010, pp. 143–158.

[4] J. Odvarko, "Http archive specification version 1.2," http://www.softwareishard.com/blog/har-12-spec/.

[5] Alexa, "Global top sites list," http://www.alexa.com/topsites/global.

[6] PhantomJS, http://phantomjs.org/.

[7] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Understanding website complexity: measurements, metrics, and implications," in *Internet Measurement Conference*, P. Thiran and W. Willinger, Eds. ACM, 2011, pp. 313–328.

[8] L. Nussbaum and O. Richard, "A comparative study of network link emulators," in *SpringSim*, G. A. Wainer, C. A. Shaffer, R. M. McGraw, and M. J. Chinni, Eds. SCS/ACM, 2009.

[9] S. Sundaresan, N. Feamster, R. Teixeira, N. Magharei *et al.*, "Measuring and mitigating web performance bottlenecks in broadband access networks," in *ACM Internet Measurement Conference*, 2013.

[10] M. M. B. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. H. Ammar, "Answering what-if deployment and configuration questions with WISE," in *SIGCOMM*, V. Bahl, D. Wetherall, S. Savage, and I. Stoica, Eds. ACM, 2008, pp. 99–110.

[11] S. Chen, K. R. Joshi, M. A. Hiltunen, W. H. Sanders, and R. D. Schlichting, "Link gradients: Predicting the impact of network latency on multitier applications," in *INFOCOM*. IEEE, 2009, pp. 2258–2266.