Is Your Web Server Suffering from Undue Stress due to Duplicate Requests?

Fahad A. Arshad, Amiya K. Maji, Sidharth Mudgal, and Saurabh Bagchi, *Purdue University*

https://www.usenix.org/conference/icac14/technical-sessions/presentation/arshad

This paper is included in the Proceedings of the
11th International Conference on Autonomic Computing (ICAC '14).
June 18–20, 2014 • Philadelphia, PA
ISBN 978-1-931971-11-9

Is Your Web Server Suffering from Undue Stress due to Duplicate Requests?

Fahad A. Arshad, Amiya K. Maji, Sidharth Mudgal, Saurabh Bagchi,
Purdue University
{*faarshad, amaji, smudgals, sbagchi*}*@purdue.edu*

Abstract

An important, if not very well known, problem that afflicts many web servers is duplicate client browser requests due to server-side problems. A legitimate request is followed by a redundant request, thus increasing the load on the server and corrupting state at the server end (such as, the hit count for the page) and at the client end (such as, state maintained through a cookie). This problem has been reported in many developer blogs and has been found to afflict even popular web sites, such as CNN and YouTube. However, to date, there has not been a scientific, technical solution to this problem that is browser vendor neutral. In this paper, we provide such a solution which we call GRIFFIN. We identify that the two root causes of the problem are missing resource at the server end or duplicated Javascripts embedded in the page. We have the insight that dynamic tracing of the function call sequence creates a signature that can be used to differentiate between legitimate and duplicate requests. We apply our technique to find unreported problems in a large production scientific collaboration web service called HUBzero, which are fixed upon reporting the problems. Our experiments show an average overhead of 1.29X for tracing the PHP-runtime on HUBzero across 60 unique HTTP transactions. GRIFFIN has zero false-positives (when run across HTTP transaction of size one and two) and an average detection accuracy of 78% across 60 HTTP transactions.

1 Introduction

The affliction of duplicated web requests: A duplicate web request occurs when the client web browser sends two requests for the same web page, the second being a redundant duplicate request. This affliction does not affect poorly run web sites alone. It afflicts two of the top 10 most visited sites — CNN and YouTube [15]. Our tests (with Chrome) show that at least 22 out of top 98 (on April 4, 2014) globally ranked Alexa [1] web sites give a duplicate request on accessing their homepages. On the academic side, we found that it affects HUBzero, a widely used open source software platform (originating from Purdue) for building powerful Web sites that support scientific discovery, learning, and collaboration [14].

Why do duplicate web requests happen? There are two root causes for the problem of duplicate web requests, which have been separately pointed out in many developer forums and blog posts [3, 4, 17]. The first cause is the incorrect way in which browsers handle missing component names, or empty tags, such as, `<img src="">`, `<script src="">`, and `<link href="">`. Equivalently, this could be caused by JavaScript which dynamically sets the `src` property on either a newly created image or an existing one: The most readable and comprehensive treatment of this first cause can be found in [3]. We will refer to this first root cause as *missing resource cause*. The second cause is the same Javascript being included in the page twice, or more number of times [15]. This is the root cause behind the duplicate web requests in CNN and YouTube. Two main factors increase the odds of a script being duplicated in a single web page: team size and number of scripts. It takes a significant amount of resources to develop a web site, especially if it is a top destination. In addition to the core team building the site, other teams contribute to the HTML in the page for things such as advertising, branding, and data feeds. With so many people from different teams adding HTML to the page, it is easy to imagine how the same script could be added twice, e.g., CNN and YouTube's main pages have 11 and 7 scripts respectively. We will refer to this second root cause as *duplicate script cause*.

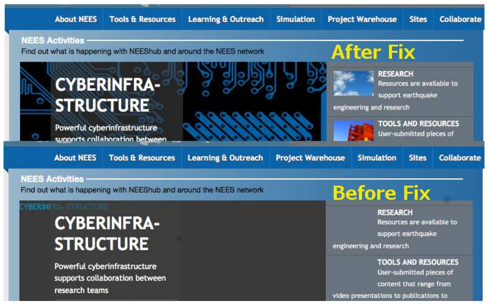
How to fix the problem? The "missing resource cause" happens because the HTML specification, version 4 [5][1] is silent on this seemingly esoteric aspect. Even though the specification indicates that the `src` attribute should

[1] HTML4 is the latest version of the specification, except for a W3C "Candidate Recommendation" for HTML5 dated 04 February, 2014.

contain a Uniform Resource Identifier (URI), it fails to define the behavior when `src` does not contain a URI. Consequently, different browsers behave in different ways. For example, Internet Explorer (IE) sends the duplicate request to the directory of the page rather than the page itself, while Firefox and Chrome send the duplicate request to the page itself. Further, the behavior of different browsers for handling different missing resources is different, *e.g.*, IE does not initiate a duplicate request with missing `script` while Firefox and Chrome do. The overall approach to handling this could be to write server-side code that will catch a similar request arising close in time to the original request and correlated with finding a missing URI in a tag. However, due to the differences in browser behaviors and for different tags, this would lead to ungainly code, with case statements for a large number of different cases. An indirect evidence comes from the fact that though this problem has been known for a while (since at least 2009), this solution is seldom deployed. The "duplicate script cause" of course has no easy solution available currently. The solution is mainly process-based — enabling better communication and coordination between developers writing or using scripts to create web pages.

Our solution approach: In this paper, we present a general-purpose solution to the above problem, in a system called GRIFFIN[2]. By "general-purpose", we mean that the solution applies unmodified to all kinds of resources and browsers. The solution has at its heart the observation that the duplicate web requests cause a repeated signal, for some definition of "signal". The signal should be defined such that it can be easily traced in a production web server, without impacting computation or storage resources and without needing specialized code insertion. We find that the *function call depth* is the signal that satisfies these conditions, while preserving enough fidelity that the repeated sequence can be easily and automatically discerned. To automatically discern the repeated pattern, we use the simple-to-calculate autocorrelation function for the signal and at a lag, equal to the size of the web request (in terms of number of HTTP commands), GRIFFIN sees a spike in autocorrelation which it uses to flag the detection.

When tested over a wide range of buggy and nonbuggy behavior, we find that GRIFFIN performs well with respect to both the detection and the false positive. We find that GRIFFIN has no false positive and an 80% detection accuracy. To make GRIFFIN feasible in real production settings, we adopt a mix of synchronous and asynchronous approaches, both without modifying the application's source code, or even needing access to the source code. Synchronously we capture the call stack depth, using a built-in functionality, in the tracing tool called SYSTEMTAP. Then, asynchronously, GRIFFIN calculates the autocorrelation function for various lags, filters the values, and flags a detection when the value exceeds a threshold. In addition to detection, GRIFFIN also provides some diagnostic insight, *i.e.*, gives an idea of the module where the root cause lies.



Find out what is happening with NEEShub and around the NEES network

Powerful cyberinfrastructure supports collaboration between

Resources are available to support earthquake engineering and research

User-submitted pieces of

Find out what is happening with NEEShub and around the NEES network

Powerful cyberinfrastructure supports collaboration between research teams

Resources are available to support earthquake engineering and research

User-submitted pieces of content that range from video presentations to publications to

Figure 1: Duplicate bug-manifestation (with missing images) before and after the fix

2 Example Bug Case

Here we present a bug-case that was observed for the beta release of the main web portal of our NSF center called NEEScomm, meant for providing a cyberinfrastructure for earthquake engineers and scientists throughout the US `www.nees.org`. GRIFFIN was able to detect it before the code update made it to the production site, and thus avoided the duplicate request problem. On accessing the homepage, the images that appear as part of background were missing (Figure 1). Listing 1 presents the code modifications that fixed the problem (no duplicate requests seen from client). In Listing 1, `$slide->mainImage` variable does not resolves to the image `XYZ.jpg` location. Instead, it resolves to the NUL character. Manual inspection revealed that the images were missing. To verify, we hard-coded a valid image location and it fixed the duplicate problem. Listing 2 shows the runtime state of the rendered HTML in Firefox browser. On lines 3 and 10, the empty `url()` is observed, while on line 4, the `src` field in `<img>` tag having a value of "/" pinpoints the root cause for the duplicate request to the base URL.

To understand how current browser versions (Chrome 32, Firefox 26) behave under unexpected input, we did a synthetic injection in HTML tags: `<span style:background=X>`, `<img src=X>`, `<script src=X>`, `<iframe src=X>`, `<link href=X>`. Here `X`, the injected character, had ASCII codes in the range 32-126 excluding alphanumeric characters. We found that, in addition to duplicate requests due to empty strings which have been reported before [3], the characters '?' and '#' also resulted in duplicate requests.

[2] GRIFFIN is a mythical creature with the front legs, wings, and head of a giant eagle, and the body, hind legs, and tail of a lion. It is often used to guard treasures.

<span style:background=SPACE,EMPTY> resulted in a duplicate request for both browsers. For Firefox, <img src=SPACE>, <script src=SPACE,EMPTY>, and <link href=SPACE> created duplicate requests. These injections provide evidence that browsers do behave differently and erroneously under unexpected special characters for URIs.

```
--- a/modules/mod_fpss/tmpl/Movies/default.php
+++ b/modules/mod_fpss/tmpl/Movies/default.php
- <span style=”background:url(<?php echo $slide->mainImage; ?>) no-
      repeat;”>
+ <span style=”background:url(media/system/images/XYZ.jpg) no-repeat;”>
- <img src=”<?php echo $slide->mainImage; ?>” alt=”<?php echo $slide->
      altTitle; ?>” />
+ <img src=”media/system/images/XYZ.jpg” alt=”<?php echo $slide->
      altTitle; ?>” />
- <span class=”navigation-thumbnail” style=”background:url(<?php echo
      $slide->thumbnailImage; ?>) no-repeat;”>&nbsp;</span>
+ <span class=”navigation-thumbnail”style=”background:url(media/system/
      images/XYZ.jpg) no-repeat;”>&nbsp;</span>
```

Listing 1: Code modification to fix unnecessary duplicate requests

```
<div class=”slide” style=”position: absolute; opacity: 0; z-index: 89;”
      >
  <a class=”slide-link” href=”/fpss/track/35/L3Jlc291,,”>
    <span style=”background:url() no-repeat;”>
      <img alt=”NEEShub Release 5.0” src=”/”>
    </span>
  </a>
.
.
.
<span class=”navigation-thumbnail” style=”background:url() no-repeat;”>
       </span>
```

Listing 2: Runtime state of generated HTML as observed by Firebug

3 Design

Here we detail the design of Griffin to detect duplicate web requests. At a high level, it comprises three steps: model application behavior at the web server (in terms of the function calls and returns), create a signal of the function call depths, and compute the auto-correlation of the signal to trigger detection. Figure 2 shows these steps in Griffin.

3.1 Synchronous Tracing

We leverage SystemTap [12], a tracing/probing framework that can provide synchronous tracing data on Linux hosts. To enable tracing, SystemTap allows to write probe-point scripts. Probe-point scripts tell SystemTap two things. *(1). What event do you want to trace? (2). What do you want to print at the traced event-location?*. Griffin logs both function-entry and function-return events and prints timestamp, thread-id, function call depth, funcation name, file name, line number, and class name, if available. Further tracing implementation details are available in [7].

3.2 Modeling Application Behavior

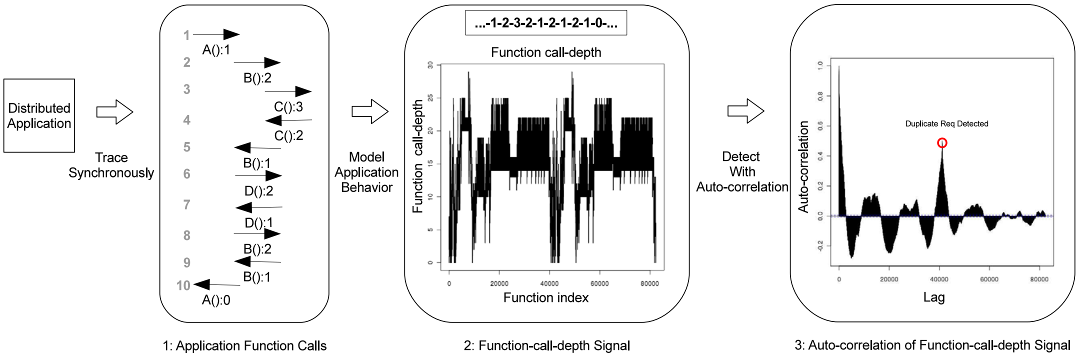
For modeling purposes, we define a numeric metric called *function call-depth* that represents the runtime function call-depth. At every function-call, the call-depth is incremented and at every return, it is decremented. *Our foundational intuition for modeling application behavior is that the flow of an application can be roughly represented by how function call depth changes.* The function call depth sequence for a given high-level web operation can be considered as a fingerprint of the high-level operation. For further exploration of this intuition, let us first define some terms: *web-request, web-click, http-transaction*. Starting from the lowest level, a web request is the HTTP request sent by the web browser, such as, GET and POST. A web click is a human user clicking in the browser to send web requests. A single web click can generate multiple web requests. A set of web clicks done in a particular sequence, as permitted by the workflow in the website, is called an http transaction. An http transaction can consist of one or more web clicks; in typical usage this will be more than one web click. An example of an http-transaction of size two is going to the homepage followed by going to the login page (HomePage→Login).

Now coming back to our intuition for detecting duplicate web requests, consider that a duplicate web request will create a duplicated signal of the function call depths. It is easy to concoct a synthetic example where this intuition is violated. For example, consider two legitimate consecutive web clicks and the corresponding web requests: (a (b (c c') b') a') (d (e (f f') e') d') giving a call-depth sequences of (1 2 3 3 2 1) (1 2 3 3 2 1). This would give the appearance to Griffin of duplicated web requests. However, we find that for real web pages, the length of web clicks in terms of the number of function calls and returns tends to be much larger. This kind of accidental matching of the function call depth signal happens only very rarely for these real situations.

To get the call-depth at runtime, we add a function called `thread_indent_depth(long)` to SystemTap 's native scripts. This function returns a number corresponding to the depth of nesting. We call this function `thread_indent_depth(1)` in the probe-point SystemTap script. Here, the argument one means that at every function-call, increment the depth by one. We submitted this function to the SystemTap repository and it has been merged into SystemTap 's master-branch and is available out-of-the-box after SystemTap is installed [6].

3.3 Duplicate Detection Algorithm

With the function call-depth sequence captured, the next goal is to detect whether the sequence has a repetitive pattern and to do this efficiently with respect to time. To do this, we use a common signal analysis technique to detect repeating patterns, *auto-correlation* [19] of the function call-depth signal. Auto-correlation of a signal x is defined by R_{xx} (Equation 1) as a function of lag-value t, where t varies from zero (perfect signal match with R_{xx}=1) to n, the sequence length in terms of the number



1: Application Function Calls

2: Function-call-depth Signal

3: Auto-correlation of Function-call-depth Signal

Figure 2: Overview of the duplicate-detection workflow.

of function calls and exits. Ideally for GRIFFIN to detect duplicate web requests resulting from a single user web click, it would be possible to segment the web requests for each web click. But that is not always possible in practice, as we discuss in [7]. Auto-correlation can be viewed as a sequences of *shift, multiply, sum* operations for all lag values on function call-depth signal. Intuitively, we are using auto-correlation to estimate the similarity between the signal and its time shifted versions for various values of the time shift. If the function-depth signal is exactly repeated twice, we expect to see a peak of 0.5 around the lag value of *n/2*.

$$R_{xx}[t] = \frac{C_t}{C_0} \text{ where t=0,\ldots,n}$$
$$C_t = \frac{1}{n} \sum_{s=max(1,-t)}^{min(n-t,n)} [X_{s+t} - \bar{X}][X_s - \bar{X}] \qquad (1)$$

After auto-correlation computation for all lag-values, we find the index at which the auto-correlation first becomes negative, call this t_0. For values of auto-correlation beyond t_0, we find if there is any value greater than a threshold value τ. If yes, we flag a duplicate-detection. For the duplication of a set of web requests once, we expect ideally an auto correlation peak of 0.5. But to tolerate the normal variation in function call-depth signal, we set the threshold τ to be a little lower than 0.5. We report on our sensitivity empirical study in Section 5.3. The reason for starting the search beyond t_0 is that then we eliminate the high values of autocorrelation that we will see due to the original signal being correlated with itself with small time lags. The pseudocode for GRIFFIN's detection algorithm is available in [7].

3.4 Usage Modes

We envision GRIFFIN to work in two scenarios, pre-production *testing* and *in-production*. In testing, developer's have control of the environment and trace segmentation is not an issue. Here, a possible concern by developers could be GRIFFIN's detection latency, which is in order of seconds. For in-production mode, operators' main concern could be the overhead of configuring and tuning GRIFFIN and the application tracing overhead, which is incurred in the critical path of all web requests and responses. GRIFFIN's configuration is minimal with only one threshold parameter for which we provide a recommendation (threshold=0.4) with our sensitivity analysis. To further minimize the tracing overhead, an operator can run GRIFFIN in time intervals of low load on the web server .

4 Experimental Setup

4.1 Configurations: Hardware, Software, Tracing

NEEShub infrastructure is running Apache/2.2.16 (Debian) web server in Prefork MPM (Multi-Processing Module) [2] mode, *i.e.*, with multiple processes and one thread per process, on a VM with Intel(R) Xeon(R) CPU E5-2643 0 @ 3.30GHz with 6GB RAM. The PHP-runtime (`libphp5.so`) version is 5.3.3 and is compiled with `--enable-dtrace` option in order for SYSTEMTAP (ver 2.4) to be able to intercept PHP-function calls and returns with its probes.

4.2 Evaluation Metrics

We evaluate GRIFFIN's detection performance with traditional definitions of accuracy and precision. Accuracy is defined as the percentage of true positives and true negatives. Precision is defined as the percentage of true positives out of all detections. We establish the ground truth through manual verification, at client-end, by checking duplicate requests for each web-click using browser debugging tools, Firebug and Chrome-dev-tools. We measure the overhead of GRIFFIN in two areas, tracing overhead and detection overhead. Tracing-overhead is the fraction of total time, taken by SYSTEMTAP 's probes while processing a given web-click. Detection overhead or detection latency is measured in the standard way as the time elapsed for all the detection steps.

5 Evaluation

5.1 Experimental Workload

Griffin's testing was conducted on a replica of the production site (www.nees.org), technically referred to as a "staging machine" where developers merge their code after doing the unit testing on their own development box. We made no modifications or synthetic error injections. Therefore, we expected to find few, if any, problems with the website.

We tested Griffin's duplicate-detection performance by sending a total of 60 HTTP transactions of varying sizes. The size of a transaction is measured by the number of web clicks incorporated within the transaction. Thus, the transaction HomePage→Login has a size of two. Also, for the analysis (autocorrelation computation), the signal is considered the entire transaction. We used 20 transactions for each of the sizes 1,2,3. These 60 HTTP transactions were executed following different possible user workflows as enabled by the web portal. We tried to cover all the workflows that a typical user would follow while visiting the website.

Ideally, the analysis in Griffin will consider the traces corresponding to a single web click from a single user. Within a single user, we expect that different web clicks are handled by threads of different IDs. We empirically validated that this is *always* the case for all our transactions.

5.2 Accuracy and Precision Results

Out of the 7 duplicate request problems (among the 60 HTTP transactions), Griffin was able to correctly find 4 duplicated requests i.e., *HomePage, Topics-page, SimulationWiki-page and Wiki-page*. SimulationWiki page was due to a Javascript-based duplication, while the other three were due to missing-resources. Griffin missed 3 cases of duplicated requests, *warehouse, simulation and education* pages.

Griffin's accuracy and precision with different HTTP transaction sizes is presented in Table 1. Griffin provides an average accuracy of 80% across HTTP transactions of size one and two with no false positives. With three web clicks, Griffin's performance degrades– here 0% precision is misleading in the sense that out of the 20 HTTP transactions of size three, only one (HOMEPAGE→LOGIN→LOGGINGIN (Figure 3)) had a duplicate request which Griffin did not detect. Griffin falsely flagged 4 out of 20 transactions giving a false positive rate of 20% for HTTP transactions of size three. The reason why Griffin did not detect HOMEPAGE web-click within HOMEPAGE→LOGIN→LOGGINGIN transaction is due to the significant difference of LOGGGINGIN function call-depth signal from the signals of HOMEPAGE and LOGIN web clicks (see the increase in function call-depth signal between index 100K to 150K in Figure 3). Here, HOME and LOGIN web clicks have an average function call-depth of 15.61 and 15.47 respectively while LOGGINGIN has an average of 32.42 making it significantly different. With HTTP transaction of size 3, Griffin is performing its analysis after combining these three signals into one. Thus, the divergence in the single combined signal means that the autocorrelation values, even with one duplication, tend to be low, and stay below the threshold. In practice, the HTTP transactions of size 3 will be very rare because of the discrimination that Griffin will be able to do using the thread ID [7].

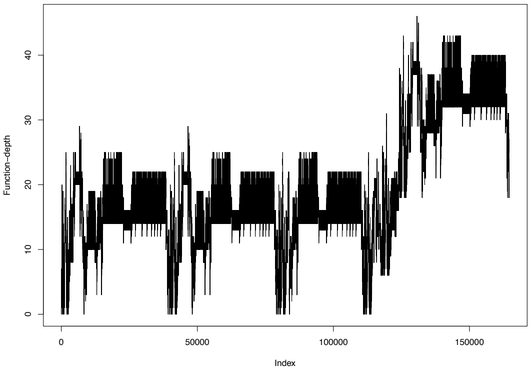
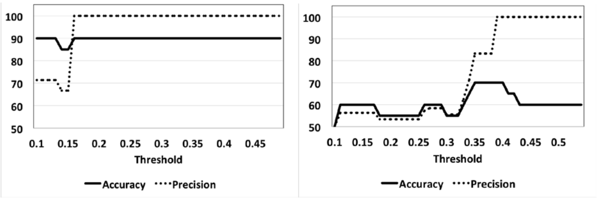


Figure 3: HOME→LOGIN→LOGGINGIN: Function call-depth signal for three web clicks from browser

	Accuracy	Precision
one-click	90% = $\frac{18}{20}$	100% = $\frac{3}{3}$
two-clicks	70% = $\frac{14}{20}$	100% = $\frac{4}{4}$
three-clicks	75% = $\frac{15}{20}$	0% = $\frac{0}{4}$

Table 1: Summary of Performance results

With the ideal (and practically common) case of analysis over HTTP transaction of size 1, Griffin shows 90% accuracy and 100% precision. As an example, the function call-depth and autocorrelation for HOME web-transaction is presented in Figure 2. We see that the autocorrelation has a clear peak value of 0.4998 near a lag-value of 40,000 which is detected by Griffin (with a threshold set at 0.4). Manual checking, both at user-end and at server-end revealed that HOME web-request ("/") is being sent twice by the user's browser. Further inspection on the server revealed that a field called `hits` in the back-end database is incremented on every HOME web-transaction. We reported this hitherto unknown problem to the web developer at NEES, and it was subsequently fixed and not pushed into the production environment. Testing Griffin with HTTP transactions of size 2, we observe a drop in accuracy (to 70%). This happens due to the significant variability in the basic signal due to the very different nature of the function call invocations in the two web clicks. Expectedly, autocorrelating a divergent signal gives low autocorrlation values, which sometime fall below the Griffin threshold (0.4).



(a) One-click

(b) Two-click

Figure 4: Sensitivity of GRIFFIN for one and two-clicks

	Tracing Overhead (Avg)	Tracing Overhead (Std. Dev)	Sequence Length (Avg)	Sequence Length (Std. Dev)
one-click	24.0%	6.6%	67,071	54,165
two-clicks	32.8%	11.6%	131,511	76,630
three-clicks	29.1%	9.1%	141,427	33,727

Table 2: Tracing Overhead

5.3 Sensitivity and Overhead

GRIFFIN's sensitivity to different parameters, sequence length, threshold and number of traced contiguous web clicks is critical from a usability perspective. With an increasing number of contiguous web clicks, GRIFFIN's accuracy and precision drop. The pattern of accuracy decreasing with increasing number web clicks holds true with increasing sizes of the traces. We present GRIFFIN's sensitivity with different thresholds in Figure 4.We set GRIFFIN threshold to 0.4 as the default value for GRIFFIN to provide us zero false positives, *i.e.*, 100% precision. The user can decrease the threshold for fine tuning her system, but we suggest to not go below 0.35 (based on Figure 4b) as that can result in possible false positives.

The detection latency as a function of the sequence length (*i.e.*, the number of trace events due to SYSTEMTAP probes) shows the expected behavior of greater latency with increasing sequence length [7]. This is due to a larger number of autocorrelation computations for a longer trace length. However, the upper range of the sequence length is typically about 100K and with that we have a detection latency of about half a minute, which should be fast enough to be useful for the subsequent manual process of fixing the problem. The average tracing overhead across the 60 tested HTTP transactions is 28.6% with a standard devitation of 10.0%. The overhead for HTTP transactions for each size is presented in Table 2. The tracing overhead is independent of the length of the sequence and the differences seen are due to statistical variations.

5.4 Diagnostic-context

When GRIFFIN detects duplicate web-requests, a diagnostic-context about the detection would help the developers as a starting point for debugging. At detection-time, in addition to the autocorrelation value, we also have the lag when this autocorrelation value exceeded the threshold, call this t_{max}. We use t_{max} alongwith the information provided by an additional SYSTEMTAP probe that records the HTTP-request going from `apache-core` to PHP-runtime, to provide the *diagnostic-context*. With the t_{max}, we get the nearest next fired `apache-core` to PHP event. We then extract a high-level component (module name) from the file name. For the duplicate bug of Figure 1, this simple scheme is able to correctly flag `mod_fpss` module in Joomla, the Content Management System, on which HUBzero is built.

6 Related Work

Most of the existing approaches to handle duplicate requests are *not* at the application-level. TCP [9] is the classic example that uses sequence numbers along with a windowing-based mechanism to do duplicate detection of IP packets. Stateless protocols like HTTP have to deal with the request-response nature and maintain state at the application-level. Application-level works include similarity detection [16] deployed at web-proxy caches to eliminate redundant network traffic, duplicate-content detection [18] with clustering and similarity metrics [11]. These are directed at generic payloads and are therefore less accurate than GRIFFIN in general.

Finding relevant system events to detect and diagnose failures is often equated to the problem of finding a needle in a haystack. Over the last decade, several researchers have proposed solutions to this challenging problem [10, 20, 8, 13]. The high-level objective here is to mine vast amounts of system data to find relevant signatures for failures. Our work falls within this broad umbrella. We automate the process of detecting duplicated web requests by looking at a compressed signal from system events, specifically function calls and returns.

7 Conclusion

In this paper, we have presented a systematic method and an automated tool called GRIFFIN for detecting an important problem that afflicts many web servers, namely, duplicate client browser requests. This causes an artificially high load on servers and corrupts server and client state. Culling together many blog posts and developer forum reports, we identify the two fundamental root causes of the problem and come up with a solution that handles both, without needing special case logic for the two root causes or for different browsers. We use GRIFFIN for detecting the problem in a production web portal for an NSF center at Purdue and identify that the problem is more widespread than previously identified. Our evaluation on the production site revealed no false positive. The dynamic system tracing using SYSTEMTAP is lightweight and the detection latency small enough (less than half a minute) as to be useful in practice. Our contributions were considered significant enough that the problem was fixed in the web portal and our addition to the dynamic tracing facility was accepted in its official release.

References

[1] Alexa Internet, Inc. http://www.alexa.com/.

[2] Apache MPM prefork. http://httpd.apache.org/docs/2.2/mod/prefork.html.

[3] Empty image src can destroy your site. http://www.nczonline.net/blog/2009/11/30/empty-image-src-can-destroy-your-site/.

[4] Empty SRC And URL() Values Can Cause Duplicate Page Requests. http://www.bennadel.com/blog /2236-Empty-SRC-And-URL-Values-Can-Cause -Duplicate-Page-Requests.htm.

[5] HTML 4.01 Specification. http://www.w3.org/TR/html4/.

[6] Systemtap call-depth feature request. https://sourceware.org/bugzilla/show_bug.cgi?id=16472.

[7] Arshad, F., Maji, A.K. Mudgal, S., and Bagchi, S. Is your web server suffering from undue stress due to duplicate requests? http://docs.lib.purdue.edu/ecetr/458, Apr. 24 2014. Technical Report, School of Electrical and Computer Engineering, Purdue University.

[8] Bodik, P., Goldszmidt, M., Fox, A., Woodard, D. B., and Andersen, H. Fingerprinting the datacenter: Automated classification of performance crises. In *Proceedings of the 5th European Conference on Computer Systems* (New York, NY, USA, 2010), EuroSys '10, ACM, pp. 111–124.

[9] Cerf, V., and Kahn, R. A protocol for packet network intercommunication. *Communications, IEEE Transactions on 22*, 5 (May 1974), 637–648.

[10] Cohen, I., Zhang, S., Goldszmidt, M., Symons, J., Kelly, T., and Fox, A. Capturing, indexing, clustering, and retrieving system history. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2005), SOSP '05, ACM, pp. 105–118.

[11] Coskun, B., and Giura, P. Mitigating sms spam by online detection of repetitive near-duplicate messages. In *Communications (ICC), 2012 IEEE International Conference on* (June 2012), pp. 999–1004.

[12] Eigler, F. C., Prasad, V., Cohen, W., Nguyen, H., Hunt, M., Keniston, J., and Chen, B. Architecture of systemtap: a linux trace/probe tool.

[13] Fu, Q., Lou, J.-G., Lin, Q.-W., Ding, R., Zhang, D., Ye, Z., and Xie, T. Performance issue diagnosis for online service systems. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on* (2012), IEEE, pp. 273–278.

[14] McLennan, M., and Kennell, R. Hubzero: A platform for dissemination and collaboration in computational science and engineering. *Computing in Science & Engineering 12*, 2 (2010), 48–53.

[15] Souders, S. High-performance web sites. *Commun. ACM 51*, 12 (Dec. 2008), 36–41.

[16] Spring, N. T., and Wetherall, D. A protocol-independent technique for eliminating redundant network traffic. *SIGCOMM Comput. Commun. Rev. 30*, 4 (Aug. 2000), 87–95.

[17] Steve W. Monkey Code. http://code.alittlegoofy.com/2008/12/i-found-something-peculiar-about.html.

[18] Vallés, E., and Rosso, P. Detection of near-duplicate user generated contents: The sms spam collection. In *Proceedings of the 3rd International Workshop on Search and Mining User-generated Contents* (New York, NY, USA, 2011), SMUC '11, ACM, pp. 27–34.

[19] Venables, W. N., and Ripley, B. D. *Modern Applied Statistics with S*. Springer Publishing Company, Incorporated, 2010.

[20] Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. I. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 117–132.