# Online Client-Side Bottleneck Identification on HTTP Server Infrastructures

Ricardo Filipe, Serhiy Boychenko, Filipe Araujo

CISUC, Dept. of Informatics Engineering
University of Coimbra
Coimbra, Portugal
{rafilipe, serhiy}@dei.uc.pt, filipius@uc.pt

*Abstract*—**Ensuring short response times is a major concern for all web site administrators. To keep these times under control, they usually resort to monitoring tools that collect a large spectrum of system metrics, such as CPU and memory occupation, network traffic, number of processes, etc. Despite providing a reasonably accurate picture of the server, the times that really matter are those experienced by the user. However, not surprisingly, system administrators will usually not have access to these end-to-end figures, due to their lack of control over web browsers. To overcome this problem, we follow the opposite approach of monitoring a site based on times collected from browsers. We use two browser-side metrics for this: $i$) the time it takes for the first byte of the response to reach the user (request time) and $ii$) the time it takes for the entire response to arrive (response time). We conjecture that an appropriate choice of the resources to control, more precisely, one or two URLs, suffices to detect CPU, network and I/O bottlenecks. In support of this conjecture, we run periodical evaluations of request and response times on some very popular web sites to detect bottlenecks. Our experiments suggest that collecting data from the browsers can indeed contribute for better monitoring tools that provide a deeper understanding of the system, thus helping to maintain faster, more interactive web sites.**

***Keywords–Cloud computing; Bottleneck; Virtualization.***

## I. Introduction

In the operation of a Hypertext Transfer Protocol (HTTP) server [1], bottlenecks may emerge at different points of the system often with negative consequences for the quality of interaction with users. To control this problem, system administrators must keep a watchful eye on a large range of system parameters, like CPU, disk and memory occupation, network interface utilization, among an endless number of other metrics, some of them specifically related to HTTP, such as response times or sizes of waiting queues. Despite being very powerful, these mechanisms cannot provide a completely accurate picture of the HTTP protocol performance. Indeed, the network latency and transfer times can only be seen from the client, not to mention that some server metrics might not translate easily to the quality of the interaction with users. Moreover, increasing the number of metrics involved in monitoring adds complexity to the system and makes monitoring more intrusive.

We hypothesize that a simpler mechanism, based on client-side monitoring, can fulfill the task of detecting and identifying an HTTP server bottleneck from a list of three: CPU, network, or disk input/output (simply I/O hereafter). The arguments in favor of this idea are quite powerful: client-side monitoring provides the most relevant performance numbers, while, at the same time, requiring no modifications to the server, which, additionally, can run on any technology. This approach can provide a very effective option to complement available monitoring tools.

To achieve this goal, we require two metrics taken from the web browser: $i$) the time it takes from requesting an object to receiving the first byte (request time), and $ii$) the time it takes from the first byte of the response, to the last byte of data (response time). We need to collect time series of these metrics for, at least, one or two carefully chosen URLs. These URLs should be selected according to the resources they use, either I/O or CPU. The main idea is that each kind of bottleneck exposes itself with a different signature in the request and response time series.

To try our conjecture, and create such time series, we resorted to experiments on real web sites, by automatically requesting one or two URLs with a browser every minute, and collecting the correspondent request and response times. With these experiments, we managed to discover a case of network bottleneck and another one of I/O bottleneck. We believe that this simple mechanism can improve the web browsing experience, by providing web site developers with qualitative results that add to the purely quantitative metrics they already own.

The rest of the paper is organized as follows. Section II presents the related work in this field and provides a comparison of different methods. Section III describes the online method to detect and identify the HTTP server bottlenecks. In Section IV we try a specific approach to show monitoring results from popular web sites, thus exposing different types of bottlenecks. Finally, in Section V we discuss the results and conclude the paper.

## II. Related Work

In the literature, we can find a large body of work focused on timely scaling resources up or down, usually in N-tier HTTP server systems, [2–7]. We divide these efforts into three main categories: (i) analytic models that collect multiple metrics to ensure detection or prediction of bottlenecks; (ii) rule-based approaches, which change resources depending on utilization thresholds, like network or CPU; (iii) system configuration analysis to ensure correct functionality against bottlenecks and peak period operations.

First, regarding analytic models, authors usually resort to queues and respective theories to represent N-tier systems [8][9]. Malkowski *et al.* [10] try to satisfy service level objectives (SLOs), by keeping low service response times.

They collect a large number of system metrics, like CPU and memory utilization, cache, pool sizes and so on, to correlate these metrics with system performance. This should expose the metrics responsible for bottlenecks. However, the analytic model uses more than two hundred application and system level metrics. In [11], Malkowski *et al.* studied bottlenecks in N-tier systems even further, to expose the phenomenon of multi-bottlenecks, which are not due to a single resource that reaches saturation. Furthermore, they managed to show that lightly loaded resources may be responsible for such multi-bottlenecks. As in their previous work, the framework resorts to system metrics that require full access to the infrastructure. The number of system metrics to collect is not clear. Wang *et al.* continued this line of reasoning in [7], to detect transient bottlenecks with durations as low as 50 milliseconds. The transient anomalies are detected recurring to depth analysis of metrics in each component of the system. Although functional, this approach is so fine-grained that it is closely tied to a specific hardware and software architecture.

In [2], authors try to discover bottlenecks in data flow programs running in the cloud. In [6], Bodík *et al.* try to predict bottlenecks to provide automatic elasticity. [5] presents a dynamic allocation of VMs based on SLA restrictions. The framework consists of a continuous "loop" that monitors the cloud system, to detect and predict component saturation. The paper does not address questions related to resource consumption of the monitoring approach or scalability to large cloud providers. Unlike other approaches that try to detect bottlenecks, [12] uses heuristic models to achieve optimal resource management. Authors use a database rule set that, for a given workload, returns the optimal configuration of the system. [13] presents a technique to analyze workloads using k-means clustering. This approach also uses a queuing model to predict the server capacity for a given workload for each tier of the system.

Other researchers have focused on rule-based schemes to control resource utilization. Iqbal *et al.* [3][14] propose an algorithm that processes proxy logs and, at a second layer, all CPU metrics of web servers. The goal is to increase or decrease the number of instances of the saturated component. [15] also scales up or down servers based on CPU and network metrics of the server components. If a component resource saturation is observed, then, the user will be migrated to a new virtual machine through IP dynamic configuration. This approach uses simpler criteria to scale up or down compared to bottleneck-based approaches, because it uses static performance-based rules.

Table I illustrates the kind of resource problem detected by the mentioned papers. The second column concerns the need to increase CPU resources or VM instances. The third column is associated to I/O, normally an access to a database. The network column represents delays inside the server network or to the client — normally browser or web services. It is relevant to mention that several articles [16][2][11] only consider CPU (or instantiated VM) and I/O bottleneck, thus not considering internal (between the several components) or external (client-server connection) bandwidth.

Finally, some techniques scan the system looking for misconfigurations that may cause inconsistencies or performance issues. Attariyan *et al.* [17] elaborated a tool that scans the system in real time to discover root cause errors in configuration. In [18], authors use previous correct configurations to eliminate unwanted or mistaken operator configuration.

TABLE I. Bottleneck detection in related work.

| Article | CPU/Threads/VM | I/O | Network |
|---|---|---|---|
| [2] | X | X | |
| [3] | X | | |
| [10] | X | X | |
| [4] | X | | |
| [7] | X | X | Internal |
| [11] | X | X | Internal |
| [15] | X | | X |

Our work is different from the previously mentioned literature in at least two aspects: we are not tied to any specific architecture and we try to evaluate the bottlenecks from the client's perspective. This point of view provides a better insight on the quality of the response, offering a much more accurate picture regarding the quality of the service. While our method could replace some server-side mechanisms, we believe that it serves better as a complementary mechanism.

It is also worth mentioning client-side tools like HTTPerf [19] or JMeter [20], which serve to test HTTP servers, frequently under stress, by running a large number of simultaneous invocations of a service. However, these tools work better for benchmarking a site before it goes online.

## III. A Conjecture on Client-Side Monitoring of HTTP Servers

We now evaluate the possibility of detecting bottlenecks based on the download times of web pages, as seen by a client. We conjecture that we can, not only, detect the presence of a bottleneck, something that would be relatively simple to do, but actually determine the kind of resource causing the bottleneck, CPU, I/O or network. CPU limitations may be due to thread pool constraints of the HTTP Server (specially the front-end machines), or CPU machine exhaustion, e.g., due to bad code design that causes unnecessary processing. I/O bottlenecks will probably be related to the database (DB) operation, which clearly depend on query complexity, DB configuration and DB access patterns. Network bottlenecks are related to network congestion.

To illustrate this possibility, we propose to systematically collect timing information of one or two web pages from a given server, using the browser side JavaScript Navigation Timing API [21]. Figure 1 depicts the different metrics that are available to this JavaScript library, as defined by the World Wide Web (W3) Consortium. Of these, we will use the most relevant ones for network and server performance: the request time (computed as the time that goes from the request start to the response start) and the response time (which is the time that goes from the response start to the response end). We chose these, because the request and response times are directly related to the request *and* involve server actions, which is not the case of browser processing times, occurring afterwards, or TCP connection times, happening before.

Consider now the following decomposition of the times of interest for us:

- Request Time: client-to-server network transfer time + server processing time + server-to-client network latency.

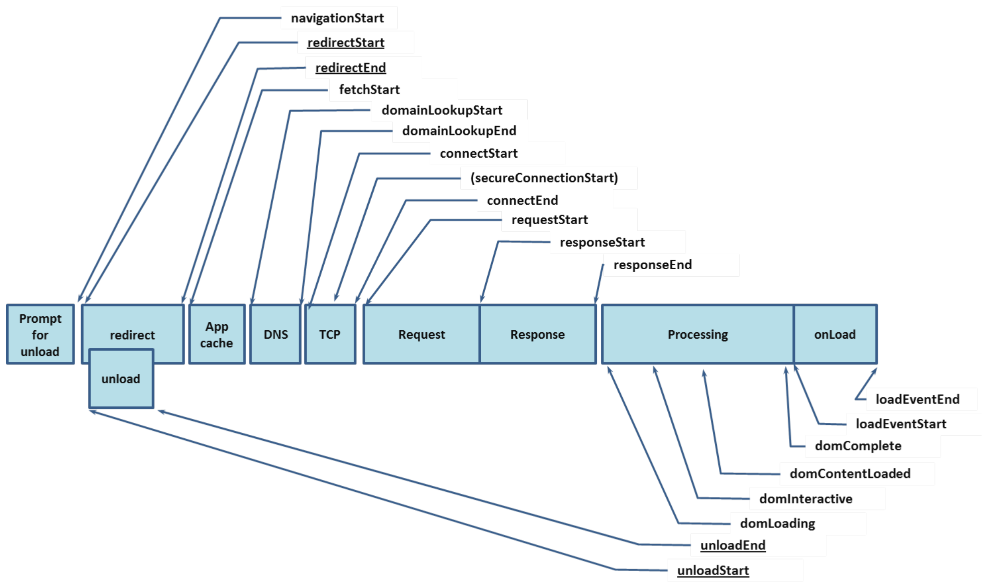

Figure 1. Navigation Timing metrics (figure from [21])

- Response Time: server-to-client network transfer time.

To make use of these times, we must assume that the server actions, once the server has the first byte of the response ready, do not delay the network transfer of the response. In practice, our analysis depends on the server not causing any delays due to CPU or (disk) I/O, once it starts responding. Note that this is compatible with chunked transfer encoding: the server might compress or sign the next chunk, while delivering the previous one.

We argue that identifying network bottlenecks, and their cause, with time series of these two metrics is actually possible, whenever congestion occurs in both directions of traffic. In this case, the request and response times will correlate strongly. If no network congestion exists, but the response is still slow, the correlation of request and response times will be small, as processing time on the server dominates. Small correlation points to a bottleneck in the server, whereas high correlation points toward the network. Hence, repeated requests to a single resource of the system, such as the entry page can help to identify network congestion, although we cannot tell exactly where in the network does this congestion occur. To this correlation-based evaluation of the request and response time series from a single URL, we call "single-page request" analysis.

Separating CPU from I/O bottlenecks is a much more difficult problem. We resort to a further assumption here: the CPU tasks share a single pool of resources, possibly with several (virtual) machines, while I/O is often partitioned. This, we believe, reflects the conditions of many large systems, as load balancers forward requests to a single pool of machines, whereas data requests may end up in separate DB tables, served by different machines, depending on the items requested. Since scarce CPU resources affect *all* requests, this type of bottleneck synchronizes all the delays (i.e., different parallel requests tend to be simultaneously slow or fast). Thus, logically, unsynchronized delays must point to I/O bottlenecks. On the other hand, one cannot immediately conclude anything, with respect to the type of bottleneck, if the delays are synchronized (requests might be suffering either from CPU or similar I/O limitations).

TABLE II. SOFTWARE USED AND DISTRIBUTION.

| Component | Observations | Version |
|---|---|---|
| Selenium | selenium-server-standalone jar | 2.43.0 |
| Firefox | browser | 23.0 |
| Xvfb | xorg-server | 1.13.3 |

The challenge is, therefore, to identify pairs of URLs showing unsynchronized delays, to pinpoint I/O bottlenecks. Ensuring that a request for an URL has I/O is usually simple, as most have. In a news site, fetching a specific news item will most likely access I/O. To have a request using only CPU or, at least, using some different I/O resource, one might fetch non-existing resources, preferably using a path outside the logic of the site. We call "independent requests" to this mechanism of using two URLs requesting different types of resources.

One should notice that responses must occupy more than a single TCP [22] segment. Otherwise, one cannot compute any meaningful correlation between request and response times, as this would always be very small.

We will now experimentally try the "single-page request" and the "independent requests" mechanisms, to observe whether they can actually spot bottlenecks in real web sites.

## IV. Experimental Evaluation

In this section we present the results of our experimental evaluation.

### A. Experimental Setup

For the sake of doing an online analysis, we used a software testing framework for web applications, called Selenium [23]. The Selenium framework emulates clients accessing web pages using the Firefox browser, thus retaining access to the Javascript Navigation Timing API [21]. We use this API to read the request and response times necessary for the "single-page request" and "independent requests" mechanisms. We used a UNIX client machine, with a crontab process, to request a page each minute [24]. The scheduler launched the Selenium process (with the corresponding Firefox browser) each minute. We emulated a virtual display for the client machine using Xvfb [25]. Table II lists the software and versions used.

One of the criteria we used to choose the pages to monitor was their popularity. However, to conserve space, we only show results of pages that provided interesting results, thus omitting sites that displayed excellent performance during the entire course of the days we tested (e.g., CNN [26] or Amazon [27]) — these latter experiments would have little to show regarding bottlenecks. On the other hand, we could find some bottlenecks in a number of other real web sites:

- **Akamai/Facebook photo repository** — We kept downloading the same 46 KiloBytes (KiB) Facebook photo, which was actually delivered by the Akamai Content Delivery Network (CDN). During the time of this test, the CDN was retrieving the photo from Ireland. This experiment displays network performance problems.
- **SAPO [28]** — this webpage is the 5th most used portal in Portugal (only behind Google – domain .pt and .com, Facebook and Youtube) and the 1st page of Portuguese language in Portugal [29]. This web page shows considerable performance perturbations on the server side, especially during the wake up hours.
- **Record sports news [30]** — This is an online sports newspaper. We downloaded an old 129 KiB news item [31] and an inexistent one [32] for several days. The old news item certainly involves I/O, to retrieve the item from a DB, whereas the inexistent may or may not use I/O, we cannot tell for sure. We ensured a separation of 10 seconds between both requests. One should notice that having a resource URL involving only CPU would be a better choice to separate bottlenecks. However, since we could not find such resource, a non-existing one actually helped us to identify an I/O bottleneck.

## B. Results

We start by analyzing the results of Facebook/Akamai and SAPO, in Figures 2, 3 and 4. These figures show the normal behavior of the systems and allow us to identify periods where response times fall out of the ordinary.

Figure 2 shows the response of the Akamai site for a lapse of several days. We can clearly observe a pattern in the response that is directly associated to the hour of the day. During working hours and evening in Europe, we observed a degradation in the request and response times (see, for example, the left area of the blue dashed line on September 19, 2014, a Friday). The green and the red lines (respectively, the response and the request times), clearly follow similar patterns, a sign that they are strongly correlated. Computing the *correlation coefficient* of these variables, $r(Req, Res)$, for the left side of the dashed line we have $r(Req, Res) = 0.89881$, this showing that the correlation exists indeed. However, for the period where the platform is more "stable" (between the first peak periods) we have $r(Req, Res) = -0.06728$. In normal conditions the correlation between these two parameters is low. This allows us to conclude that in the former (peak) period we found a network bottleneck that does not exist in the latter. However, our method cannot determine where in the network is the bottleneck. Interestingly, in Figure 3, we can observe that the bottleneck disappeared after a few days. On September $29^{th}$, we can no longer see any sign of it.

Regarding Figure 4, which shows request and response times of the main SAPO page, we can make the same analysis for two distinct periods: before and after 9 AM (consider the blue dashed line) of December 13, 2013 (also a Friday).

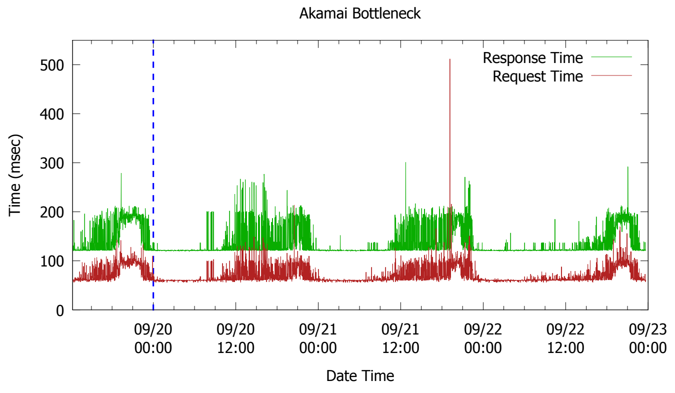

Figure 2. Akamai/Facebook bottleneck.

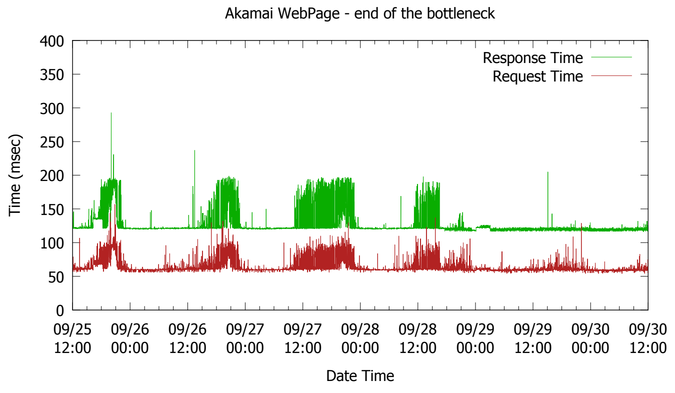

Figure 3. Akamai/Facebook - end of the bottleneck.

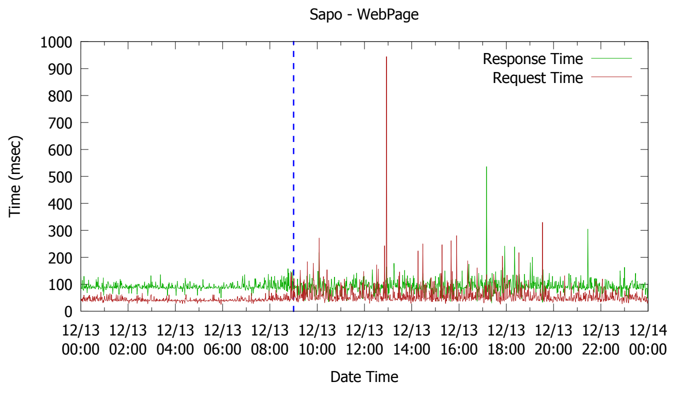

Figure 4. SAPO bottleneck.

Visually, we can easily see the different profiles of the two areas. The correlation for these two areas are:

- $r(Req, Res)_{before9AM} = 0.36621$
- $r(Req, Res)_{after9AM} = 0.08887$

The correlation is low, especially during the peak period, where the response time is more irregular. This case is therefore quite different from the previous one, and suggests that no network bottleneck exists in the system, during periods of intense usage. With the "single-page request" method only, and without having any further data of the site, it is difficult to precisely determine the source of the bottleneck (CPU or I/O).

To separate the CPU from the I/O bottleneck, we need to resort to the "independent requests" approach, which we followed in the Record case. Figures 5, 6, 7 and 8 show time series starting on February $18^{th}$, up to February $21^{st}$ 2015. We do not show the response times of the inexistent page as these are always 0 or 1, thus having very little information of interest for us. In all these figures, we add a plot of the moving average with a period of 100, as the moving average is extremely helpful to identify tendencies.

Figures 5 and 6 show the request time of the old 129 KiB page request. The former figure shows the actual times we got, whereas in the latter we deleted the highest peaks (those above average), to get a clearer picture of the request times. A daily pattern emerges in these figures, as woken hours have longer delays in the response than sleeping hours. To exclude the network as a bottleneck, we can visually see that the response times of Figure 7 do not exhibit this pattern, which suggests a low correlation between request and response times (which is indeed low). Next, we observe that the request times of the existent and inexistent pages (refer to Figure 8) are out of sync. The latter seems to have much smaller cycles along the day, although (different) daily patterns seem to exist as well. For the reasons we mentioned before, in Section III, under the assumption that processing bottlenecks would *simultaneously* affect both plots, we conclude that the main source of bottlenecks in the existent page is I/O. This also suggests the impossibility of having the request time dominated by access to a cache on the server, as this would impact processing, thus causing synchronized delays. A final word for the peaks that affect the request time: they are weakly correlated to the response times. Hence, their source is also likely to be I/O.

## V. Discussion and Conclusion

We proposed to detect bottlenecks of HTTP servers using client-side observations of request and response times. A comparison of these signals either over the same or a small number of resources enables the identification of CPU, network and I/O bottlenecks. We did this work having no access to internal server data and mostly resorting to visual inspection of the request and response times. If run by the owners of the site, we see a number of additional options:

- Simply follow our approach of periodically invoking URLs in one or more clients, as a means to complement current server-side monitoring tools. This may help to reply to questions such as "what is the impact of a CPU occupation of 80% for interactivity?".

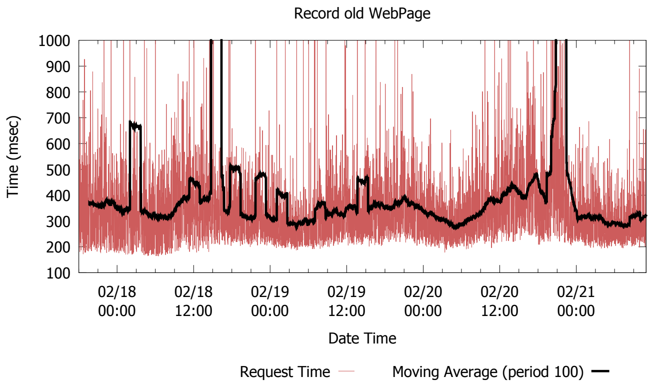

Figure 5. Record old page — request times.

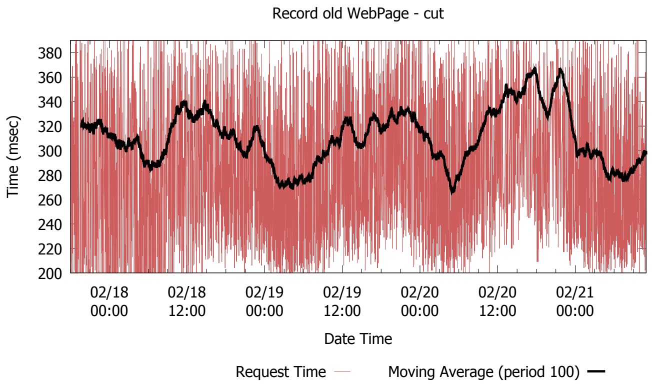

Figure 6. Record old page — response times with peaks cut.

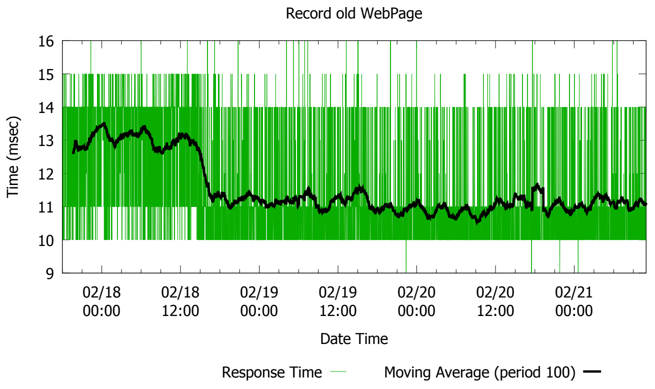

Figure 7. Record old page — response times.

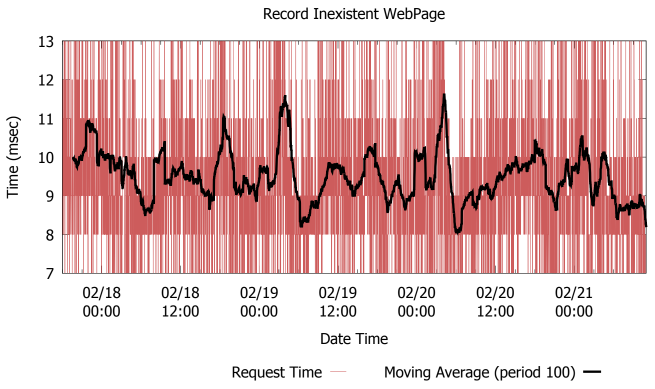

Figure 8. Record inexistent page — request times.

- A hybrid approach, with client-side and server-side data is also possible. I.e., the server may add some internal data to *each* request, like the time the request takes on the CPU or waiting for the database. Although much more elaborate and dependent on the architecture, instrumenting the client and the server sides is, indeed, the only way to achieve a fully decomposition of request timings.
- To improve the quality of the analysis we did in Section IV, site owners could also add a number of very specific resources, like a page that has known access time to the DB, or known computation time.
- It is also possible to automatically collect timing information from real user browsers, as in Google Analytics [33], to do subsequent analysis of the system performance. In other words, instead of setting up clients for monitoring, site owners might use their real clients, with the help of some Javascript and AJAX.

In summary, we collected evidence in support of the idea of identifying bottlenecks from the user side. Nonetheless, to unambiguously demonstrate the results we found, we recognize the need for further evidence, from a larger number of sites, and from supplementary monitoring data from the server.

## REFERENCES

[1] RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1, Internet Engineering Task Force (IETF), Internet Engineering Task Force (IETF) Std., June 1999. [Online]. Available: http://www.faqs.org/rfcs/rfc2616.html

[2] D. Battre, M. Hovestadt, B. Lohrmann, A. Stanik, and D. Warneke, "Detecting bottlenecks in parallel dag-based data flow programs," in Many-Task Computing on Grids and Supercomputers (MTAGS), 2010 IEEE Workshop on, 2010, pp. 1–10.

[3] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," Future Generation Computer Systems, vol. 27, no. 6, 2011, pp. 871–879.

[4] Y. Shoaib and O. Das, "Using layered bottlenecks for virtual machine provisioning in the clouds," in Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on, 2012, pp. 109–116.

[5] N. Huber, F. Brosig, and S. Kounev, "Model-based self-adaptive resource allocation in virtualized environments," in Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, ser. SEAMS '11. New York, NY, USA: ACM, 2011, pp. 90–99. [Online]. Available: http://doi.acm.org/10.1145/1988008.1988021

[6] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," in Proceedings of the 2009 conference on Hot topics in cloud computing, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855533.1855545

[7] Q. W. *et al.*, "Detecting transient bottlenecks in n-tier applications through fine-grained analysis," in ICDCS. IEEE Computer Society, 2013, pp. 31–40. [Online]. Available: http://dblp.uni-trier.de/db/conf/icdcs/icdcs2013.html#WangKLJSMKP13

[8] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in Autonomic Computing, 2007. ICAC '07. Fourth International Conference on, June 2007, pp. 27–27.

[9] G. Franks, D. Petriu, M. Woodside, J. Xu, and P. Tregunno, "Layered bottlenecks and their mitigation," in Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on, Sept 2006, pp. 103–114.

[10] S. Malkowski, M. Hedwig, J. Parekh, C. Pu, and A. Sahai, "Bottleneck detection using statistical intervention analysis," in Managing Virtualization of Networks and Services. Springer, 2007, pp. 122–134.

[11] S. Malkowski, M. Hedwig, and C. Pu, "Experimental evaluation of n-tier systems: Observation and analysis of multi-bottlenecks," in Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on. IEEE, 2009, pp. 118–127.

[12] R. Chi, Z. Qian, and S. Lu, "A heuristic approach for scalability of multi-tiers web application in clouds," in Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on, 2011, pp. 28–35.

[13] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy, "Autonomic mix-aware provisioning for non-stationary data center workloads," in Proceedings of the 7th international conference on Autonomic computing, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/1809049.1809053

[14] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Sla-driven automatic bottleneck detection and resolution for read intensive multi-tier applications hosted on a cloud," in Advances in Grid and Pervasive Computing. Springer, 2010, pp. 37–46.

[15] H. Liu and S. Wee, "Web server farm in the cloud: Performance evaluation and dynamic architecture," in Proceedings of the 1st International Conference on Cloud Computing, ser. CloudCom '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 369–380. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10665-1_34

[16] B. Singh and P. Nain, "Article: Bottleneck occurrence in cloud computing," IJCA Proceedings on National Conference on Advances in Computer Science and Applications (NCACSA 2012), vol. NCACSA, no. 5, May 2012, pp. 1–4, published by Foundation of Computer Science, New York, USA.

[17] M. Attariyan, M. Chow, and J. Flinn, "X-ray: automating root-cause diagnosis of performance anomalies in production software," in Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 307–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387910

[18] F. Oliveira, A. Tjang, R. Bianchini, R. P. Martin, and T. D. Nguyen, "Barricade: defending systems against operator mistakes," in Proceedings of the 5th European conference on Computer systems, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 83–96. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755924

[19] "Papers — HP Web server performance tool," http://www.hpl.hp.com/research/linux/httperf/, retrieved: May, 2015.

[20] "Performance tools — Apache JMeter$^{TM}$," http://jmeter.apache.org/, retrieved: May, 2015.

[21] "Papers — Navigation Timing," https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/NavigationTiming/Overview.html, retrieved: May, 2015.

[22] J. Postel, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[23] "Papers — Selenium Browser automation," http://www.seleniumhq.org/, retrieved: May, 2015.

[24] "Crontab - quick reference — admin's choice - choice of unix and linux administrators," http://www.adminschoice.com/crontab-quick-reference, retrieved: May, 2015.

[25] "Xvfb," http://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml, retrieved: May, 2015.

[26] "Breaking news, u.s., world, weather, entertainment & video news - cnn.com," http://edition.cnn.com, retrieved: May, 2015.

[27] "Amazon.com: Online shopping for electronics, apparel, computers, books, dvds & more," http://www.amazon.com, retrieved: May, 2015.

[28] "SAPO," http://www.sapo.pt, retrieved: May, 2015.

[29] "Alexa — Top Sites in Portugal," http://www.alexa.com/topsites/countries/PT, retrieved: May, 2015.

[30] ":.:Jornal Record:.:," http://www.record.xl.pt, retrieved: May, 2015.

[31] ":.: Albiol: Relação arrefeceu entre Casillas e Arbeloa - Entrevistas - Jornal Record :.:," http://www.record.xl.pt/Entrevistas/interior.aspx?content_id=826333, retrieved: May, 2015.

[32] "Inexistent record page," http://www.record.xl.pt/naoexiste.

[33] B. Clifton, Advanced Web Metrics with Google Analytics. Alameda, CA, USA: SYBEX Inc., 2008.