Evaluating the SiteStory Transactional Web Archive With the ApacheBench Tool

Justin F. Brunelle[1,2], Michael L. Nelson[2], Lyudmila Balakireva[3], Robert Sanderson[3], and Herbert Van de Sompel[3]

[1] The MITRE Corporation, Hampton, VA 23666
jbrunelle@mitre.org
[2] Old Dominion University, Department of Computer Science, Norfolk VA, 23529
{jbrunelle, mln}@cs.odu.edu
[3] Los Alamos National Laboratory, Los Alamos, NM 87544
{ludab, rsanderson, herbertv}@lanl.gov

Abstract. Conventional Web archives are created by periodically crawling a Web site and archiving the responses from the Web server. Although easy to implement and commonly deployed, this form of archiving typically misses updates and may not be suitable for all preservation scenarios, for example a site that is required (perhaps for records compliance) to keep a copy of all pages it has served. In contrast, transactional archives work in conjunction with a Web server to record all content that has been served. Los Alamos National Laboratory has developed SiteStory, an open-source transactional archive written in Java that runs on Apache Web servers, provides a Memento compatible access interface, and WARC file export features. We used Apache's ApacheBench utility on a pre-release version of SiteStory to measure response time and content delivery time in different environments. The performance tests were designed to determine the feasibility of SiteStory as a production-level solution for high fidelity automatic Web archiving. We found that SiteStory does not significantly affect content server performance when it is performing transactional archiving. Content server performance slows from 0.076 seconds to 0.086 seconds per Web page access when the content server is under load, and from 0.15 seconds to 0.21 seconds when the resource has many embedded and changing resources.

Keywords: Web Archiving, Digital Preservation

1 Introduction

Web archiving is an important aspect of cultural, historical, governmental, and institutional memory. The cost of capturing Web-native content for storage and archiving varies and is dependent upon several factors. The cost of manual Web archiving has prompted research into automated methods of digital resource capture. The traditional method of automatic capture is the Web crawler, but recent migrations toward more personalized and dynamic resources have rendered crawlers ineffective at high-fidelity capture in certain situations. For example, a

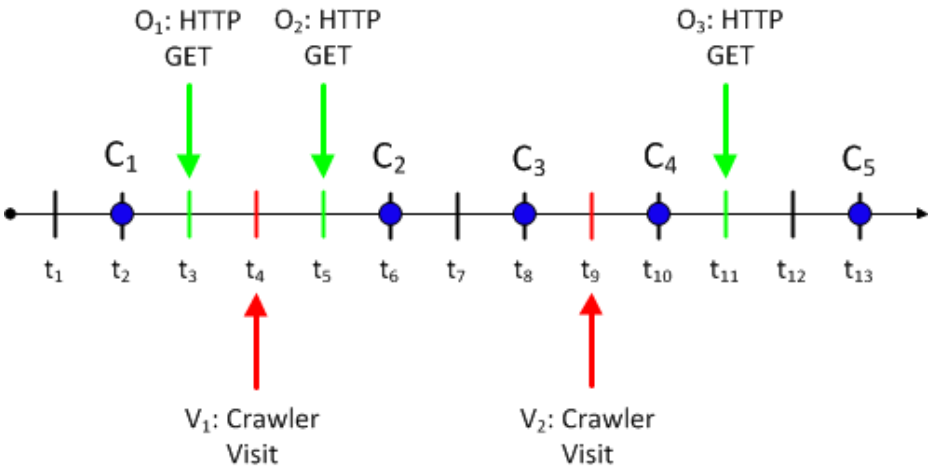


Fig. 1. User and crawler accesses control the archival interval, capturing each returned representation.

crawler cannot capture every representation of a resource that is customized for each user. Transactional archiving can, in some instances, provide an automatic archiving solution to this problem.

1.1 Transactional Archiving

The purpose of a transactional archive (TA) is to archive every representation of a resource that a Web server disseminates. A client does an HTTP GET on a URI and the Web server returns the representation of the resource at that time. At dissemination time, it is the responsibility of TA software to send to an archive the representation sent to the client. In this way, *all* representations returned by the Web server can be archived. If storing all served representations is costly (e.g., a high-traffic site with slowly changing resources), it is possible to optimize a TA in a variety of ways: store only unique representations, store every n^{th} representation, etc.

Figure 1 provides a visual representation of a typical scenario where a page P is both changed and access at irregular intervals. This scenario assumes an arbitrary page that will be called P changes at inconsistent intervals. This timeline shows page P changes at points C_1, C_2, C_3, C_4, and C_5 at times t_2, t_6, t_8, t_{10}, and t_{13}, respectively. A user makes a request for P at points O_1, O_2, and O_3 at times t_3, t_5, and t_{11}, respectively. A Web crawler (that captures representations for storage in a Web archive) visits P at points V_1 and V_2 at times t_4 and t_9, respectively.

Since O_1 occurs after change C_1, an archived copy of C_1 is made by the TA. The Web crawler visits V_1 captures C_1, and makes a copy in the Web archive. In servicing V_1 or O_1, an unoptimized TA will store another copy of C_1 at t_4

and an optimized TA could detect that no change has occurred and not store another copy of C_1.

Change C_2 occurs at time t_6, and C_3 occurs at time t_8. There was no access to P between t_6 and t_8, which means C_2 is lost – an archived copy exists in neither the TA nor the Web crawler's archive. However, the argument can be made that if no entity observed the change, should it be archived? Change C_3 occurs and the representation of P is archived during the crawler's visit V_2, and the TA will also archive C_3. After C_4, a user accessed P at O_3 creating an archived copy of C_4 in the TA.

In the scenario depicted in Figure 1, the TA will have changes C_1, C_3, C_4, and a conventional archive will only have C_1, C_3. Change C_2 was never served to any client (human or crawler) and is thus not archived by either system. Change C_5 will be captured by the TA when P is accessed next.

1.2 SiteStory

Los Alamos National Laboratory has developed SiteStory[4], an open-source transactional Web archive. First, mod_sitestory is installed on the Apache server that contains the content to be archived. When the Apache server builds the response for the requesting client, mod_sitestory sends a copy of the response to the SiteStory Web archive, which is deployed as a separate service. This Web archive then provides Memento-based access to the content served by the Apache server with mod_sitestory installed, and the SiteStory Web archive is discoverable from the Apache Web server using standard Memento conventions (see Section 4 of [14]).

Sending a copy of the HTTP response to the archive is an additional task for the Apache Web server, and this task must not come at too great a performance penalty to the Web server. The goal of this study is to quantify the additional load mod_sitestory places on the Apache Web server to be archived.

2 Prior Work

Extensive research has been done to determine how Web documents change on the Web. Studies of "wild" pages (such as Cho's work with crawlers [4] or Olston's work in recrawl scheduling [10]) have shown that pages change extremely frequently.

Prior research has focused on crawlers and robots to find pages and monitor their change patterns [3, 6, 17]. These crawlers follow the links on pages to discover other pages and archive and recrawl the discovered pages over time to compile an archive. This method is unsuitable for an intranet that is closed to the public Web; crawlers cannot access the resources of archival interest [8]. As a way to have finer control over the archival granularity, transactional archiving should be used. Transactional archiving implementations include TTApache [5] and pageVault [7]. These implementations were also shown not to substantially increase the access time seen by Web users; pageVault saw an increase of access

[4] `http://mementoWeb.github.com/SiteStory/`

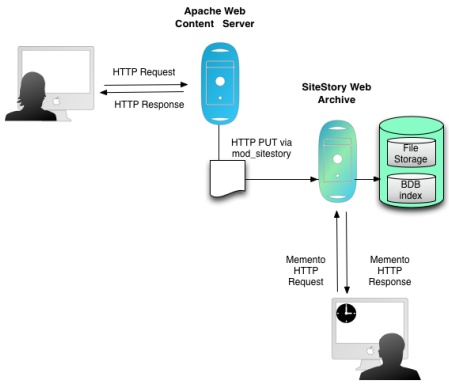


Fig. 2. SiteStory consists of two parts: mod_sitestory which is installed on the Apache server to be archived, and the transactional archive itself. Image taken from the SiteStory GitHub at `http://mementoWeb.github.com/SiteStory/`.

time from 1.1 ms to 1.5 ms, and TTApache saw a 5-25% increase in response time, depending on requested document size.

Memento is a joint project between Old Dominion University and Los Alamos National Laboratory. The Memento Framework defines HTTP extensions that allow content negotiation in the dimension of time [15, 16]. When used with Memento-aware user agents like MementoFox [11], users can set a desired datetime in the past and browse the Web as it existed at (or near) that datetime. Unlike other, single-archive applications like DiffIE [12, 13], Past Web Browser [9], or Zoetrope [1], Memento provides an multi-archive approach to presenting the past Web. Integrating multiple Web archives can give a more complete picture of the past Web [2].

3 Experiment design

SiteStory was tested with a variety of loads on a variety of resources. Three different tests were run during the experiment.

3.1 Experiment Machines

The SiteStory benchmarking experiment was conducted with a pre-release version of SiteStory installed on a machines referred to as PC1. PC1 has a single core 2.8 GHz processor, ran the prefork version of the Apache 2 Web server, and the mod_sitestory-enabled Apache server provided content from `localhost:8080`. The SiteStory archive was installed as a separate service at `localhost:9999`. Although the developers have experimented with optimizations discussed in Section 1.1, SiteStory currently archives all returned representations regardless of whether the representation has changed or not.

3.2 Experiment Runs

Three separate experiments were run on PC1. The first experiment tests the throughput of a content server enabled with SiteStory software. The second experiment performs a series of accesses to 100 static resources to test the access rates, response times, and round trip times possible. The third experiment performs a series of accesses to 100 dynamic, constantly changing set of 100 resources to demonstrate a worst-case scenario for SiteStory – everything is archived on each access.

3.3 Connection Handling: ab

This first experiment to measure the differences in throughput when SiteStory is running and when SiteStory is turned off was run twice a day for 45 days, resulting in 90 data points. The experiment uses the ab (ApacheBench) tool[5], with a total of N connections made with a concurrency of C connections, where N and C are specified by the user. The ab utility records the response, throughput, and other server stats during a test. Essentially, the ApacheBench utility issues HTTP GET requests for content to establish a benchmark for performance.

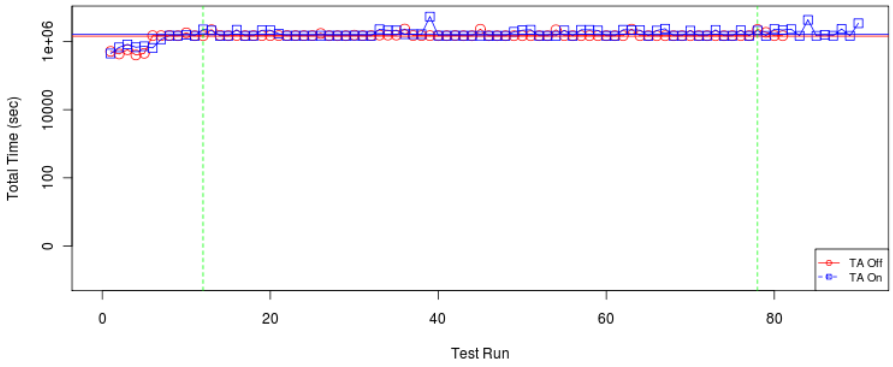
Three different HTML resources were targeted with this test: a small, medium, and large file of sizes 1kB, 250 kB, and 700 kB. We used combinations of N=(1,000, 10,000, 100,000, 216,000) and C=(1, 100, 200, 450) as parameters to the ab utility. We chose the file sizes, connection, and concurrency values to match the values observed in our study of MITRE's Corporate Intranet. For simplicity and brevity, this report discusses the runs of 10,000 connections with concurrencies 1 and 100, and runs of 216,000 connections with concurrencies 1 and 100. This subset of results illustrates typical results of all other tests.

We modified the three resources between each set of connections to ensure the resource is archived each run. To modify the resources, we ran a script to update a timestamp displayed on each page and change the image that was embedded in the page. These modifications would ensure that not only the image was changed and able to be re-archived, but the surrounding HTML was changed, as well. Since SiteStory re-archives content whenever a change is detected, each test run results in each resource being re-archived. It is essential to make sure the resource is re-archived to observe the effect of an archival action on the content server performance.

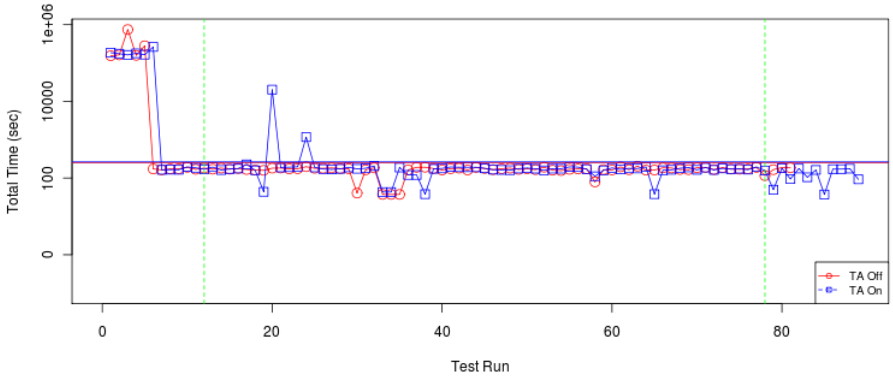
We ran each ab test twice: once while SiteStory was turned on, and once while it was turned off. This shows how SiteStory affects the content server performance. A subset of the results are provided in Figure 3. The red lines represent the runs in which SiteStory was turned off, while the blue lines represent the runs in which SiteStory was turned on. Each entry on the x-axis represents an independent test run. The y-axis provides the amount of time it took to execute the entire ab run. The horizontal lines represent the averages over the entire experiment. The dotted, vertical green lines indicate machine restart times due to power outages. The power outages were noted to show when a cache and memory resets may have occurred that could impact the performance of the machines.

[5] `http://httpd.apache.org/docs/2.0/programs/ab.html`

To illustrate how SiteStory affects the content server's performance, please reference Figure 3 that portrays the changes in the total run time of the ab test when SiteStory is on (actively archiving served content) and off (not archiving served content).



(a) Total run time for the ab test with 10,000 connections and 1 concurrency.



(b) Total run time for the ab test with 10,000 connections and 100 concurrency.

Fig. 3. Total run time for 10,000 Connections.

3.4 100 Static Resources: Clearing the Cache

The second experiment uses the `curl` command to access 100 different HTML resources, none of which change. After running the ab tests in Section 3.3, a

theory was formulated that a reason for some of the anomalies was from server caching. This additional test shows the effect of clearing the server cache on SiteStory by accessing a large number of large files in sequence. This access essentially thrashes the server cache. Each resource has text, and between 0 and 99 images (the 0th resource has 0 images, the 1st resource has 1 image, etc.). These resources were generated by a Perl script that constructed 100 different HTML pages and embedded between 0-99 different images in the generated resources. The resources were created with different sizes, and different numbers of embedded resources to demonstrate how SiteStory affects content server performance with a variety of page sizes and embedded images.

Figure 4 demonstrates the accesses of the 100 resources. The dark blue and red lines indicate the average run time for accessing a resource (in seconds). The filled areas around the lines are the standard deviation (σ) of the observations over the duration of the experiment.

3.5 100 Changing Resources: Worst-Case Scenario

We ran the same experiment from Section 3.4 in which each resource changes between runs to provide a "worst case scenario" of data connections vs. archiving and run time. We executed a script in between each run in which each resource was updated to make SiteStory archive a new copy of the resource. This means that each access resulted in a new archived copy of each resource. The results of this run are shown in Figure 5(a).

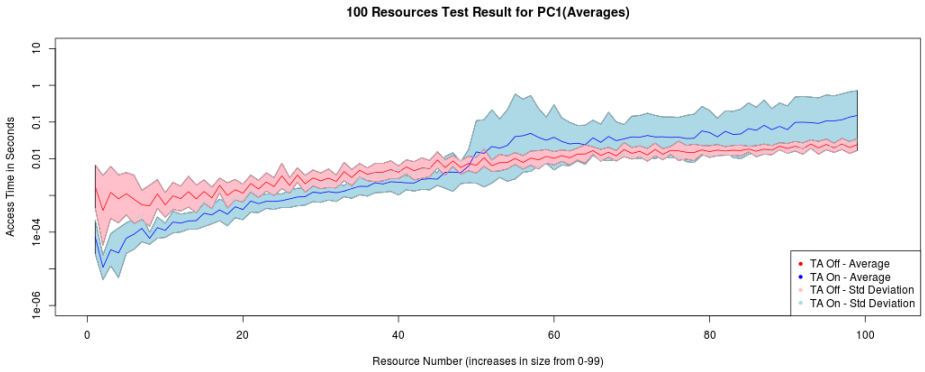
Note that Figure 5 show a "burdened" system. An artificial user load was induced on the servers to simulate a production environment in which many users are requesting content. A script was run during the test that made curl calls to the server pages to induce the load. Figure 5 shows the impact of SiteStory operating in a burdened environment.

4 Results

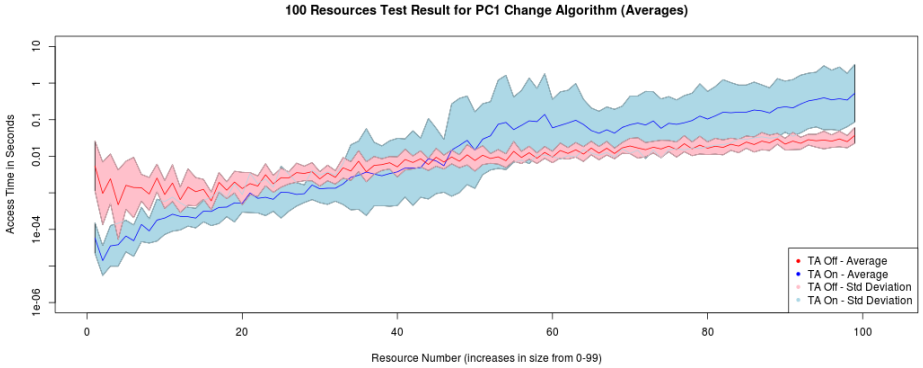
This section explores the results of the tests, from which we conclude whether or not SiteStory affects its host content server in an acceptable manner.

4.1 ab Results

For the ApacheBench tests described in Section 3.3, several obvious patterns emerge. Primarily, there is little separation between the total run times of the ab tests when SiteStory is on and when SiteStory is off. One can observe only minor differences in the plotted results. The results differ very little between any given run of the tests, and the averages across the experiment are almost identical in all tests. In the run of N=10,000 and C=1, the average total run times were 6.156 seconds when SiteStory was off, and 6.214 seconds when SiteStory was on. In the run of N=10,000 and C=100, the average total run time was 2.4 seconds when SiteStory was off, and 2.42 seconds when SiteStory was on. In the run of N=216,000 and C=1, the average run time was 8.905 seconds when SiteStory was off, and 8.955 seconds when SiteStory was on. In the run



(a) Total access time for the 100 static resources on PC1.

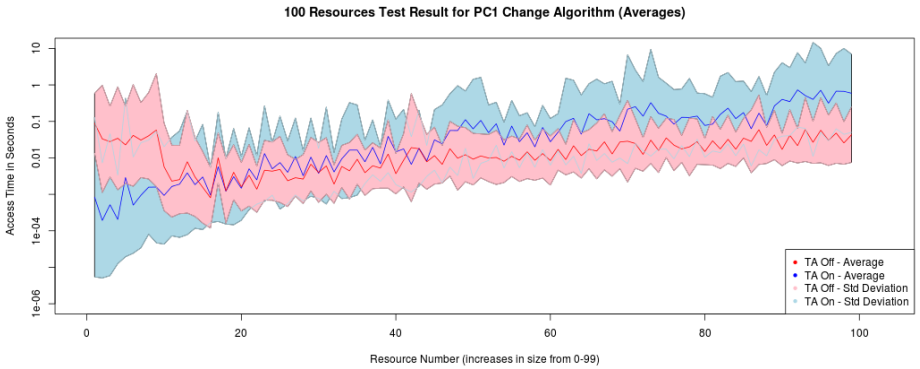


(b) Total access time for the 100 changing resources on PC1.

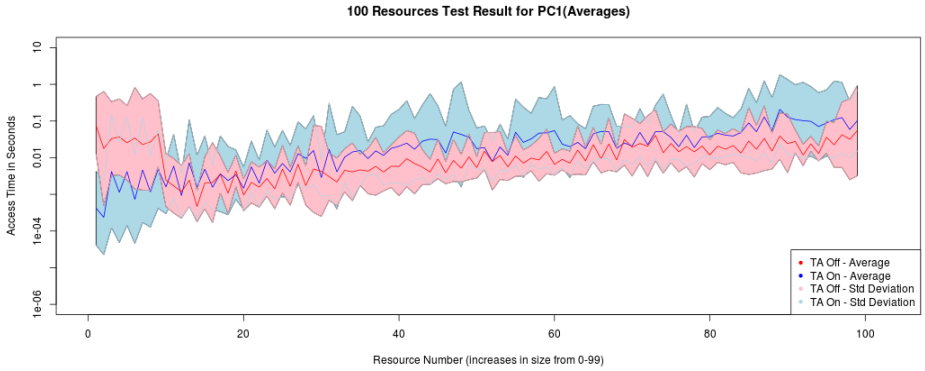
Fig. 4. 100 resources accessed on PC1. Resource n has n embedded images.

of N=216,000 and C=100, the average total run time was 4.698 seconds when SiteStory was off, and the average total run time was 4.706 when SiteStory was on. This indicates SiteStory does not significantly affect the run time of the ab statistics, and therefore does not affect the performance of the content server with regard to content delivery time.

Additionally, C=1 resulted in more consistent executions across each run whereas the runs with C=100 are more inconsistent, as indicated by the spikes in runtime. This could potentially be because of server caching, connection limitations, or even machine memory restrictions. The runs of C=100 also begin with a much longer total run time before dropping significantly and leveling out at runs 9 and 10. This is due to additional processes running on the experiment machines that induced extra load in runs 1-8. However, the spikes and inconsistencies do not affect a single run, and do not affect only the runs in



(a) Total access time for the 100 static resources on a burdened PC1.



(b) Total access time for the 100 changing resources on a burdened PC1.

Fig. 5. 100 resources accessed on a burdened PC1. Resource n has n embedded images.

which SiteStory is on or those when SiteStory is off. As such, these anomalies are disregarded since they affect both runs.

Finally, the runs of 216,000 connections take much longer to complete than the runs of 10,000 connections – specifically, 2.736 seconds longer, on average. This is intuitive since more connections should take longer to run. Additionally, the runs of C=1 take 3.9 seconds longer than the runs of C=100. By executing more connections in parallel, the total run time is intuitively shorter.

The ab test provides evidence that SiteStory does not significantly affect server content delivery time. As such, a production server can implement SiteStory without users observing a noticeable difference in server performance.

4.2 100 Resource Results

The runs of the 100 resources are more interesting, and provide a deeper insight into how SiteStory affects the server's performance than the ab test. This section

examines the results of both the static and changing resource tests, as they provide interesting contrasts in performance. The results are listed in Table 1.

When comparing the unchanging vs changing resources (such as Figure 4(a) vs. 4(b)), it is apparent that σ is, on average, two times higher for the changing resources than the unchanging resources. (The average σ for unchanging resource is 0.0839 and 0.1680 for changing resources.) Additionally, the average access times when SiteStory is off remains approximately the same when the resources change or remain the same. The interesting result is that the average access time increases from 0.15 seconds per GET to 0.21 seconds per GET for the changing resources when SiteStory is on. This is intuitive considering SiteStory needs to re-archive the accessed content during an access when the resource changes.

The most important observation in Figures 4(a) and 5(b) is that the run time of this test is approximately 0.5 seconds higher on average when SiteStory is on vs. when SiteStory is off. This number is reached by comparing the difference in average run time for each test when SiteStory is on vs. off. For each on-off pair, the average difference was taken to reach the approximate 0.5 second difference across all tests. That is, the difference between the average run times of the tests in Figures 4(a) when SiteStory is running (red) vs when SiteStory is off (blue) is 0.08 seconds. When the same comparison is performed across all tests and the average of these results is taken, an overall impact of SiteStory on server performance is realized.

Each figure begins with SiteStory off taking more time than when SiteStory is on, but this can be attributed to experiment anomaly or similar server access anomaly. Inevitably, the run time when SiteStory is on becomes slower than when SiteStory is off as the resource size increases. This demonstrates that the performance difference of a server when SiteStory is on vs. off is worse when there is a large amount of embedded resources, such as images. PC1's average page access time increases by, on average, 0.006 seconds per embedded image. One could come to the conclusion that servers providing access to image-laden resources would see the biggest performance decrease when utilizing SiteStory.

Table 1. 100 Resource Test Results

Case	Avg. Unburdened Run Time	Unburdened σ	Avg. Burdened Run Time	Burdened σ
Static Resources				
SS Off	0.121	0.0254	0.192	0.2021
SS On	0.206	0.1811	0.292	0.3103
Changing Resources				
SS Off	0.132	0.0346	0.225	0.2174
SS On	0.354	0.4244	0.292	0.6137

5 Conclusions

In this work, we stress tested and benchmarked a pre-release version of SiteStory with the ApacheBench (ab) utility. Our experiment environment replicates resource sizes and access loads observed in MITRE's Corporate Intranet. The results of this study show that SiteStory does not significantly affect the performance of a server. While different servers and different use cases cause different performance effects when SiteStory is archiving content, the host server is still able to serve sites in a timely manner. The type of resource and resource change rate also affects the server's performance – resources with many embedded images and frequently changing content are affected most by SiteStory, seeing the biggest reduction in performance.

SiteStory does not significantly increase the load on a server or affect its ability to serve content – the response times seen by users will not be noticeably different in most cases. However, these graphs demonstrate the impact of SiteStory on performance, albeit small – larger resources with many embedded resources take longer to serve when SiteStory is on as opposed to when SiteStory is off due to the increased processing required of the server. However, the significant finding of this work is that SiteStory will not cripple, or even significantly reduce, a server's ability to provide content to users. Specifically, SiteStory only increases response times by a fraction of a second – from 0.076 seconds to 0.086 seconds per access when the server is under load, and from 0.15 seconds to 0.21 seconds when the resource has many embedded and changing resources. These increases will not be noticed by human users.

6 Acknowledgments

This work is supported in part by NSF grant 1009392 and the Library of Congress. A Corporate Case Study to investigate the feasibility of a transactional archive in a corporate intranet was funded by a Fiscal Year 2011 Innovation Grant from the MITRE Corporation. MITRE employees Jory T. Morrison and George Despres were integral to the MITRE Innovation Grant and Case Study.

References

1. E. Adar, M. Dontcheva, J. Fogarty, and D. Weld. Zoetrope: interacting with the ephemeral web. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 239–248. ACM, 2008.
2. S. Ainsworth, A. Alsum, H. SalahEldeen, M. C. Weigle, and M. L. Nelson. How much of the Web is archived? In *JCDL '11: Proceedings of the 11th annual international ACM/IEEE Joint Conference on Digital Libraries*, pages 133–136, 2011.
3. B. Brewington, G. Cybenko, D. Coll, and N. Hanover. Keeping up with the changing Web. *IEEE Computer*, 33(5):52–58, 2000.
4. J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the 26th international conference on very large data bases*, pages 200–209, 2000.

5. C. E. Dyreson, H.-l. Lin, and Y. Wang. Managing versions of Web documents in a transaction-time Web server. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, 2004.
6. D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. *Software: Practice and Experience*, 34(2):213–237, 2004.
7. K. Fitch. Web site archiving: an approach to recording every materially different response produced by a Website. In *9th Australasian World Wide Web Conference*, pages 5–9, July 2003.
8. K. Hagedorn and J. Sentelli. Google Still Not Indexing Hidden Web URLs. *D-Lib Magazine*, 14(7), August 2008. `http://dlib.org/dlib/july08/hagedorn/07hagedorn.html`.
9. A. Jatowt, Y. Kawai, S. Nakamura, Y. Kidawara, and K. Tanaka. Journey to the past: proposal of a framework for past web browser. In *Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 135–144. ACM, 2006.
10. C. Olston and S. Pandey. Recrawl scheduling based on information longevity. In *Proceeding of the 17th international conference on World Wide Web*, pages 437–446. ACM, 2008.
11. R. Sanderson, H. Shankar, S. Ainsworth, F. McCown, and S. Adams. Implementing Time Travel for the Web. *Code4Lib Journal*, 13, 2011.
12. J. Teevan, S. T. Dumais, and D. J. Liebling. A longitudinal study of how highlighting web content change affects people's web interactions. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, 2010.
13. J. Teevan, S. T. Dumais, D. J. Liebling, and R. L. Hughes. Changing how people view changes on the web. In *UIST '09: Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 237–246, 2009.
14. H. Van de Sompel, M. L. Nelson, and R. Sanderson. HTTP framework for time-based access to resource states – Memento draft-vandesompel-memento-06. `http://tools.ietf.org/pdf/draft-vandesompel-memento-06.pdf`, 2013.
15. H. Van de Sompel, M. L. Nelson, R. Sanderson, L. L. Balakireva, S. Ainsworth, and H. Shankar. Memento: Time Travel for the Web. Technical Report arXiv:0911.1112, 2009.
16. H. Van de Sompel, R. Sanderson, M. L. Nelson, L. L. Balakireva, H. Shankar, and S. Ainsworth. An HTTP-Based Versioning Mechanism for Linked Data. In *Proceedings of the Linked Data on the Web Workshop (LDOW 2010)*, 2010. (Also available as arXiv:1003.3661).
17. J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 136–147, 2002.