Avoiding Denial of Service via Stress Testing *

Saeed Abu-Nimeh, Suku Nair, and Marco Marchetti
HACNet Lab
Southern Methodist University
Dallas, TX 75275
{ sabunime, nair, marco}@engr.smu.edu

Abstract

Stress-testing has been widely used by businesses, governments, and other organizations to evaluate the strength of their web applications against various attacks. However, the quality of these tests is under constant scrutiny to determine their effectiveness. The present study compares four stress-testing tools, by performing the tests on two major web-based applications. All of the tools used are open source, and run on Win32 platform. The test scenarios are recorded from server log files to make the tests more realistic. Lastly, we discuss how to use stress-testing tools as a measure to avoid Denial of Service attacks on web servers and web applications.

Keywords

Denial of Service, Stress-testing, Ramp-up time, Think time, Response time, TTFB, TTLB, Bandwidth throttle.

1. Introduction

Web-based applications have been used ubiquitously either for internal (intranet) or external (internet) use because of the fact that they are user friendly, portable, and reliable [20]. When the web application is deployed on the server, it becomes hard to predict its behavior due to external factors, which affects it, such as: number of users browsing the application, available bandwidth, or network congestion [18]. Therefore, it is important to test the application and the server before the deployment stage to get more reliable system and better performance [22].

There are several techniques that can be used for testing web applications, such as: performance testing [21], functionality testing, regression testing, stress-testing, or load-testing [11]. In this paper we concentrate on two testing techniques: stress and load testing. Sometimes stress-testing and load-testing are used in conjunction with each other. They both simulate multiple concurrent virtual web users to place a load on a server or application, however they are really two different techniques.

Stress-testing as in [17] is used to force programs to operate under limited resource conditions. The goal is to push the upper functional limits of a program to ensure that it can function correctly and handle error conditions gracefully. On the other hand, load-testing shows how a program handles large amounts of data, excessive calculations, and excessive processing. In this case, we do not necessarily have to push or exceed upper functional limits.

According to [13] there are several circumstances which call for load-testing, such as: Determining the response time of the application if the load on the application is increased, measuring the effect on response time when a new functionality is added to a web application, or performing IT infrastructure changes.

Denial of Service (DoS) is one of the security threats that cause businesses, governments, and organizations a lot of revenue losses. In February 2000, eBay, Yahoo, Amazon, Buy.com, and others suffered from a deadly DoS that caused a total loss of $1.2 billion [23]. Another attack in January 2001 on Microsoft caused a lot of damage and affected the company’s reputation. The Computer Emergency Response Team (CERT) was also targeted in successful Distributed DoS (DDoS) attacks in March 2001.

Stress-testing web applications can be useful as avoidance against DoS. By measuring how the server performs under stress, operators can predict how that server will perform in the real world. Accordingly, they will be able to approximate the amount of traffic or the number of concurrent users the application or the server can handle in order to avoid DoS. In this study, we use stress-testing to determine the breaking point of web-based applications and to see how these applications respond when put under stress.

This paper is organized as follows: In section 2 we discuss

---

*This work was completed on an internship at Dallas Semiconductor Dallas, TX.
related and previous work. Section 3 presents systems, applications, and tools used in performing stress tests. Section 4 summarizes the experimental results. In section 5 we discuss the results and comment on using web stress-testing to avoid DoS. Finally, we draw conclusions and discuss future work in section 6.

2. Related Work

Many researchers have considered the problem of analyzing and studying the performance of web servers and web-based applications. Load generators have been used to measure performance by generating synthetic load of HTTP requests for the server. These load generators provide performance statistics, such as throughput in HTTP operations/second, the count of request methods, response codes, URL popularity, and transfer size distributions. By measuring how the server performs in a controlled environment, operators will hopefully have a reasonably good notion of how that server will perform in the real world [15].

WebStone [2], which was released in 1995, is one of the oldest workload generators. It did only simple request generation for 5 unique files, ranging from 500 bytes to 5 MB. In 1996, SPECweb96 [3] was released to be the first SPEC’s workload generator. Later on, Banga and Druschel [9] proposed measuring web servers with tools that can generate and sustain overload, which is equivalent to simulating an infinite user base. To be able to generate such loads, they presented a tool called s-clients. In addition, they introduced the notion of open-loop workload generation. Most workload generators are tightly coupled to the server they are measuring so they are considered as closed-loop where they wait to generate a new request only after a response has been received.

Barford and Crovella [10] proposed the first workload generator SURGE that reproduced server workload characteristics such as: document popularity, embedded references, and transfer size. Mosberger and Jin [14] proposed a quantitative performance analysis tool httperf that is capable of generating and sustaining overload of web servers. The tool can readily be used to perform various kinds of web-server related measurements. It also includes the open-loop model used by s-client. Nahum et al. [16] proposed WaspClient a hybrid workload generator to evaluate server performance in wide-area networks. They used SURGE’s workload model with the request generation engine from s-client.

SPECweb99 [4] is one of the standard workload generators used in the computer industry for performance evaluation, robustness testing, and evaluating server optimizations. It measures the maximum number of simultaneous connections requesting the predefined benchmark workload that a web server is able to support while still meeting specific throughput and error rate requirements. The connections are made and sustained at a specified maximum bit rate with a maximum segment size intended to more realistically model conditions. To ensure that connections are conforming, SPECweb99 monitors the bandwidth available to each connection and enforces both an upper and lower bound on that bandwidth [15].

On the other hand, there has been work in testing and analyzing the contents of web applications. Wu and Offutt [24] defined a generic analysis model that covers both static and dynamic aspects of web-based applications. Their technique identifies atomic elements of dynamic web pages that have static structure and dynamic contents. Kung et al. [12] proposed testing web applications using Object Oriented model. Ricca and Tonella [19] used the UML model to analyze and test web applications. Apparently, most of the work is done in testing contents, design, performance, or load of web-based applications. However, there has not been much work in stress-testing these applications.

3. Methodology and Evaluation Approach

3.1. Description of systems and applications

We used four open source stress-testing tools. All of them run on Win32 platforms. We tested two web-based intranet applications: ScheduleA and eZGoals.

ScheduleA: is a web-based intranet application, developed at Dallas Semiconductor [1]. It is a product sales management system for new and existing customer-based information. It tracks all budget forecasting of certain product lines between business units and field sales offices. It manages, tracks, and profiles emerging customers through the approval stage after negotiation process between the two entities. The application uses connection pooling to speed up response time. Table 1 shows the system platform and database system specifications. Figure 1 shows the structure of ScheduleA.

![Figure 1. Structure of ScheduleA.](image-url)

eZGoals: is another web-based intranet application, developed at Dallas Semiconductor. It is used to track, manage and report test engineering quarterly goals. The application has over 500 users ranging in title from "test engineer" to "executive vice president". eZGoals is developed in JAVA utilizing JSP, servlets, and JAVA beans
Table 1. ScheduleA specifications.

<table>
<thead>
<tr>
<th>System Platform</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Sun Solaris 9</td>
</tr>
<tr>
<td>Application Server</td>
<td>Oracle 9iAS</td>
</tr>
<tr>
<td>Programming Environment</td>
<td>J2EE application</td>
</tr>
<tr>
<td>Database System</td>
<td></td>
</tr>
<tr>
<td>Operating System</td>
<td>Sun Solaris 9</td>
</tr>
<tr>
<td>DBMS</td>
<td>Oracle 9i</td>
</tr>
<tr>
<td>JDBC Connections</td>
<td>J2SEE 1.4 complaint, Oracle JDBC API supplied with JDeveloper 4.0</td>
</tr>
</tbody>
</table>

with a front-controller design pattern. An Oracle database stores the data and an oracle thin client drivers are used to communicate from JAVA beans. A connection pool is also implemented to speed up response time. eZGoals has the same specifications (Table 1) as ScheduleA; however, it has Tomcat 4.1 Application Server. Figure 2 shows the structure of eZGoals.

Figure 2. Structure of eZGoals.

All experiments were performed using a single client PC running Windows 2000 (service pack 4), with Pentium III processor, 966 MHz, 512 MB of RAM, and 20 GB H.D.D. The client hardware has 33 MHz PCI 32-bit buses and one 10/100 MB Ethernet adapter.

When we performed the experiments, we used different number of concurrent users: 5, 15, and 30. We simulated the concurrent users using a single client described above. After studying the production server log files, we found that the average number of concurrent users using ScheduleA application is 7 users ranging from 5 to 9 users. However, the average number of concurrent users using eZGoals is 10 users ranging from 5 to 15 users. In order to gradually increase the load on the server, we started testing with 5 users, which is the minimum number of concurrent users using both applications. Then, we tested using 15 users, which is the maximum number of concurrent users in both applications. In order to push or exceed upper functional limits for both applications, we doubled the amount of maximum concurrent users, so we tested using 30 users. Since all of the users will not start using the application on the same time, we wanted the system to wait before adding a new user after the test begins, thus we set the ramp-up period to 3 seconds. We performed the tests for one time, so we set the loop count to 0 iterations. We set the delay between consecutive requests to 100 ms. We took the measures immediately after the tests were launched and stopped taking them after the tests finished, thus the warm-up time and cool-down time were set to 0 seconds. In order to use the maximum amount of bandwidth available we discarded the bandwidth throttle. The average number of redirects per page is 10 ranging from 5 to 15 redirects. Hence, we set HTTP redirects to 15 redirects per page. Since the users of the application are intermediate users (not beginners nor experts), they know how to navigate through the system, hence most of them take less than one second to perform a key stroke or a mouse click, therefore we set the think time to 750 milliseconds. When we looked into the server log files, we found that the maximum duration for a user to access the applications was 2.5 minutes. In order to generate more loads on the application we doubled the test duration and set it to 300 seconds. Finally, cookies were enabled to allow caching and avoid network congestion.

In the next section we discuss each testing tool in details; we give an introduction about the tool, how to install it, and how to run it.

3.1.1. JMeter. JMeter [6] is a JAVA open source load-testing tool. It is used to performance test both static and dynamic web applications. Also, it can be used to perform FTP, JDBC, Servlets, Perl scripts performance tests. JMeter requires JDK1.4 or higher. It runs on Both Win32 and UNIX platforms. To install JMeter, one needs to download the zip file from JMeter website, and then extract the .zip file in a folder and run jmeter.bat under bin directory.

3.1.2. Web Application Stress Tool. Web Application Stress Tool (WAS) [5] is a free tool from Microsoft. It is used to simulate multiple browsers requesting pages from a web site. It can also be used to gather performance and stability information about a web application. It requires a WinNT platform with service pack 4.0 or higher. To install WAS one needs to download the .exe file from Microsoft website. Then follow installation instructions. It will be installed under windows program files.

3.1.3. OpenSTA. Open System Testing Architecture (OpenSTA) [8] is designed around CORBA; it was originally developed to be commercial software by CYRANO. It has the capability of performing scripted HTTP and HTTPS heavy load tests with performance measurements from Win32 platforms. It requires Win NT 4.0 Service pack 5.0 or higher. To install the tool one needs to download the

---

1The source code of WAS is not provided by Microsoft, so one might argue that WAS is not open source.
.msi file from their website. Then follow installation instructions. It will be installed under windows program files.

3.1.4. TestMaker. TestMaker [7] is developed in JAVA. It is capable of testing: HTTP, HTTPS, SOAP, XML-RPC, and Mails (SMTP, POP3 and IMAP). It also can be used in building test agents. It requires JDK1.4 or later. To install the tool, one needs to download TestMaker zip file from their website. Then, unzip in a folder and run TestMaker.bat file.

3.2. Evaluation Approach

In order to get realistic testing scenarios, we used the intranet server log files to obtain sequence of derivation [24] between the user and the server, and a realistic flow path of user interactions with the application.

We recorded the test scenario for testing ScheduleA as the following: The user logged in using a "Business Manager" role. Then, the user started a new challenge. The user checked the details of the challenge. Then, the user added a new account. Finally, the user updated the new account and logged out.

The test scenario for testing eZGoals was recorded as follows: The user logged in using an employee role. The user created a profile to the employee. Then, the user created a new goal for that employee. A manager evaluated and reviewed the goal. Then, the user checked all goals timeline. Finally, the user generated a report for all goals and logged out.

After recording test scenarios, we used the load generator, which is built in the stress-testing tools, to mimic the browser’s behavior and replay the requests to the server using multiple concurrent users.

Our study relied on evaluation metrics. These evaluation metrics determine the input variables used to plug to the testing tools. The input variables were used interchangeably to obtain different output deliverables. Most of the testing tools use similar input variables, however they are different in their output deliverables. Table 2 represents some evaluation metrics for input variables with their description. In Table 3 we discuss some of the output deliverables provided by the stress tools.

4. Experimental Results

In this section, we compare the output deliverables of each tool for ScheduleA and eZGoals. We test the same application using four tools and compare the results. The goal of these experiments is to see whether different stress tools provide similar results if the input is the same. We use fixed input variables as discussed in section 3.1.

<table>
<thead>
<tr>
<th>Table 2. Evaluation metrics (Input variables)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
</tr>
<tr>
<td>Number of threads, number of virtual users, number of concurrent users</td>
</tr>
<tr>
<td>Ramp-up period</td>
</tr>
<tr>
<td>Loop count</td>
</tr>
<tr>
<td>Request delay</td>
</tr>
<tr>
<td>Test run time</td>
</tr>
<tr>
<td>Warm-up time</td>
</tr>
<tr>
<td>Cool-down time</td>
</tr>
<tr>
<td>Bandwidth throttle</td>
</tr>
<tr>
<td>HTTP redirects</td>
</tr>
<tr>
<td>Think time</td>
</tr>
</tbody>
</table>

4.1. ScheduleA Results

In Figure 3, we show that the request rate increases when the number of concurrent users increases. Due to the increase in number of users, the number of requests increases correspondingly. According to the increase in number of requests, the server reply rate increases as shown in Figure 4. When the number of requests increases, the number of server responses increases. Therefore, the reply rate increases. When the number of users increases, the server delay increases. The increase in number of users causes an increase in number of requests. Consequently, the response time increases as illustrated in Figure 5. If the test attempts to access a link that is broken or requests a page that returns an HTTP response other than 200 (OK), this is considered an error. As shown in Figure 6 although the number of users increases, the error rate remains almost con-
Table 3. Evaluation metrics (Output deliverables)

<table>
<thead>
<tr>
<th>Output deliverable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server throughput</td>
<td>The actual number of requests per second the server can handle.</td>
</tr>
<tr>
<td>Requests count</td>
<td>The number of requests the server responded to the client.</td>
</tr>
<tr>
<td>Response time</td>
<td>Time between receiving a stimulus and delivering an appropriate response or reaction (in milliseconds).</td>
</tr>
<tr>
<td>TTFB</td>
<td>Average Time To First Byte: Time in milliseconds for the client to get the first byte.</td>
</tr>
<tr>
<td>TTLB</td>
<td>Average Time to Last Byte: Time in milliseconds for the client to get the last byte.</td>
</tr>
<tr>
<td>Error rate</td>
<td>Number of errors occurred per second. The error can be anything other than HTTP 200 response.</td>
</tr>
</tbody>
</table>

stant. Some dynamic pages retrieve information from the server, so they take more time than pages, which display static contents. “Customer information” is a servlet that retrieves customer’s information from the Database. Figure 7 shows how the response time for dynamic pages increases when the number of users increases.

4.2. eZGoals Results

We get similar observations for testing the second application, however the figures are different. In Figure 8, we show that the request rate increases when the number of users increases. Consequently, the server reply rate will increase as shown in Figure 9.

The increase in number of requests causes more delay in the server. Therefore, the response time increases. This is shown in Figure 10. The error rate for the second application is 0%. All the pages respond with 200 HTTP responses and no links are broken. In Figure 11, we show how the response time for “User information” page increases when the number of users increases. “User information” is a servlet that retrieves user information from the Database.

5. Discussion

The previous experiments provide an initial exploration into the diversity of output deliverables provided by the stress tools. The results show that different outputs are obtained even though the test scenarios are the same and the input variables are fixed. The results in Figure 3, Figure 5, Figure 9 and Figure 10 show that although the input variables are fixed (number of concurrent users, test duration, etc.), the response time and the reply rate are different. This is due to external factors affecting the network during the test duration, such as: available bandwidth, traffic in the network, number of users logged into the network, and others. All of these factors affect the test and result in different outcomes. These factors affect response time, reply rate (See Figure 3, Figure 6, and Figure 8), however they don’t have effect on error rate or request rate. Error rate and number of requests are not affected with the network traffic. On the other hand, the error rate and number of requests can be affected by the way the tests are initiated. Some tools use different testing techniques than others. For instance, when simulating multiple users JMeter deals with each test as a block of instructions, however WAS and OpenSTA interleave the tests. This results in a different error rate and different request rate depending on the test duration.
5.1. Avoiding DoS

In Figure 5 and Figure 10, the response time increases when the number of users increases. This is true till the server reaches the maximum capacity of users it can handle. The same thing happens for the server throughput. As the number of users increases, the number of requests per second increases till the server reaches the maximum capacity of users it can handle.

ScheduleA is designed to support connection pooling. Connection pooling means that connections are reused rather than created each time a connection is requested. To facilitate connection reuse, a memory cache of database connections, called a connection pool, is maintained. Connection pooling is designed to provide improvement to performance by reusing connections. However, if it is not used correctly, it can kill the performance of the application or the server.

We did an experiment with ScheduleA where we increased the number of concurrent users to 50. By this increase, we reached to the application break point and the server was down. We tested the application using the same conditions mentioned in section 3.1, except that the number of users was increased. After looking into the server logs and the application design, we found that connections were still alive even after the tests were terminated. Therefore, the number of active connections increased exponentially as they are not dying, which in return denied new connections from being established due to memory limitations. Correspondingly, this caused DoS.

Figure 12 shows how the response time increases gradually, when the number of users increases. When DoS happens the response time starts increasing exponentially, till it reaches a point where the server does not respond. Similarly, the server throughput increases as the number of users increases. Figure 13 shows, how the throughput increases till the server reaches the maximum capacity. When this happens, the throughput decreases till it reaches 0. This is where the server is down and there is DoS. By studying both the server throughput and the response time, we could determine the application break point and see that there is a potential DoS if the number of concurrent users exceeds 30.

The previous experiment showed that there are two internal factors that can cause DoS to web-based applications: First, hardware infrastructure, such as: application-server’s memory, router’s speed that controls available bandwidth. If the hardware infrastructure is limited, the number of users that want to use a web application will be limited. Second, application infrastructure and the application design which
affects the number of users using the application. The previous experiment showed how inappropriate use of connection pooling caused DoS to the application.

6. Conclusion

The paper presented a study of web application stress-testing tools. The purpose of these tools was to measure the performance of web applications, see how they react when put under stress, and find their breaking point. We tested two web-based applications developed at Dallas Semiconductor. The applications used servlets and JSPs to communicate with a database server. Our particular workload was derived from a realistic sequence of interactions between the user, the application, and the web server. We used the load generator that is built-in each of the testing tools to mimic the web browser’s behavior and replay the requests to the web server. The results showed that performing similar test scenarios would provide different results depending on network state. External factors affecting the network during the test duration, such as: available bandwidth, traffic in the network, number of users logged into the network, would affect the results of the tests. In addition, the structure of the tool, and the testing strategy the tool uses might affect the results. For instance, JMeter simulates concurrent users as a whole block, i.e. user 1 will finish testing before user 2. However, WAS and OpenSTA interleave the tests, i.e. user 10 might be simulated before user 3. As a result, different error rates and different request rates might occur depending on the test duration. Moreover, the results showed that there are two factors that affect the quality of web-based applications: First, hardware infrastructure, such as: memory size on the server, routers’ speed, and available bandwidth. Limited hardware infrastructure degrades the performance of servers, which consequently degrades the performance of web applications. Second, application infrastructure or the design of the application, such as: using connection pooling. Similarly, this affects the quality of the application, consequently affecting the performance of the application and the server. In addition, the study showed that the tools share input variables such as: number of concurrent users and test duration. Also, they share output deliverables such as: response time, and server throughput. However, there is no tool, which provides all output deliverables that the tester expects. Hence, testers need to use a combination of stress-testing tools to achieve more extensive results. In addition, we found that using web application stress-
testing tools can be used as an early alarm against DoS. In order for companies and businesses to avoid DoS, they need to stress-test their web applications before deploying them on the production server. Therefore, they will be able to determine the maximum number of concurrent users the server can handle, the maximum server throughput, the response delay, the maximum server utilization, and the response time during earlier stages of the design phase. These figures help in identifying the weak points of the application or hardware limitations that might cause DoS and need improvement.

The results motivate future work to develop new tools that provide more comprehensive deliverables to help testers in evaluating the quality of web-based applications. Future work will explore the need for new metrics to enhance the evaluation process, help companies and businesses avoid DoS, and protect their assets and services. Future work will focus on stress-testing as a tool to avoid DoS.

Acknowledgement

This work was completed on an internship at Dallas Semiconductor, Dallas, TX. We thank the team in the MIS department for this opportunity. Also, we would like to thank Stacey Elliott from SMU for his help and comments.

References


Figure 13. DoS effect on throughput.