Vrije University Amsterdam
Computer Science Department

A WordPress-based Web Hosting Benchmark

Master Thesis
Internet and Web Technology

Scientific Supervisor:
Dr. Guillaume Pierre

Author:
Alexandra Andronescu
2001632

July, 2011
Amsterdam

TODO: Acknowledgements

Abstract

When it comes to the stress tools or benchmarks currently available for testing Web hosting systems, there are only a few options to choose from. We designed and implemented WordPressBench, which is a WordPress-based benchmark which aims at bringing a realistic Web traffic simulation. The distributed design makes possible to support a large amount of traffic, with no bottlenecks. The benchmark provides more control over the testing process, by generating fluctuating traffic intensity through a variable number of emulated users. By using WordPress, a blogging and website platform running on millions web servers, WordPressBench adds more realism to the simulation.

Contents

Chapter 1

Introduction

Current IT trends are moving towards fully developed Internet applications and storage services. Lightweight Internet services like e-mail, social networking and news and blog feeds are also very popular and are used daily by most Internet users. The information demand is getting very high, so the information availability is important, as well as a high level of reliability. For offering such Internet applications and storage services there is an obvious need of a strong infrastructure support which needs extensive testing beforehand. Academic researchers and industry developers need benchmarks to test their databases, load balancers, application servers, cloud computing platforms, or other Web hosting platforms. They do not only need massive load generators, but also realistic web traffic, which emulates users as precisely as possible. This is done by analyzing and modeling their behavior. Benchmarks offer an idea of how the resource provisioning system would react in extreme cases like overload, or flash crowds. Besides this, benchmarks offer a reproducible and comparable simulation needed for comparing different systems, or even the same updated system.

Benchmarks generate web traffic for stressing various components of the hosting system under test. The Web Server is the main component of the resource provisioning system, being the contact point where all requests are sent. This is why the Web Server is the most tested component. Most systems also include a Database Server, which is tested through intensive read/write requests of either static or dynamic data. Some systems might also contain a Cache Server, Image Server, or a Multimedia Server, depending on what is their purpose, which also need to be put under test.

In the last years, as soon as the demand for such benchmarks increased, a few benchmarks emerged and are still used up to this moment. Some of them are very popular, like TPC-W[], which is viewed as an industry standard for testing hosting systems, despite the fact that is declared outdated. Although it was originally built for testing hardware by generating massive load, nowadays its modified versions are used in academia for testing resource provisioning sys-

tems. TPC-W simulates a retail book-store website and emulates thousands of browsers. Similarly, RUBBoS[] and RUBBiS[] model a bulletin board website, and an e-commerce website. These applications are centered on simulating a massive number of HTTP requests.

The main problems with the previous benchmarks is that they do not consider the best design choices, and do not offer very realistic simulations of web traffic. They offer little control over the simulation, having pre-defined scales. These are very useful when trying to compare other systems, or when trying to repeat the simulation, but do not offer flexibility. The websites used by them are custom made for the benchmarks, and are not designed for real usage, and therefore are not very convincing.

A new generation of Web application benchmarks is necessary. One benchamrk that emerged lately is WikiBench. It is an academic benchmark, made for testing server platforms, and which uses real traffic by manipulating real Wikipedia traffic database dumps []. It offers a high degree of realism and its results are reproducible. Despite the fixed traffic traces, the traffic characteristics can be modified for the simulation, lowering the realism of the simulation. The main problem of WikiBench is that it is not flexible and it does not offer control over the simulation characteristics, following past traffic traces.

We have designed and developed WordPressBench in order to surpass some of the problems of previously described benchmarks. WordPressBench was designed to offer a realistic simulation, and is not based on old traffic dumps, offering more control and therefore addresses a different area of needs. This thesis presents the design, challenges and implemenation of WordPressBench.

WordPressBench is a Web-hosting benchmark for testing various Web-server platforms by emulating real-user behavior. Its web-servers run WordPress, a real website, used by a large number of Internet users. The benchmark's main advantage is that it offers a distributed configuration of the system, following a master-slave architecture, which is closer to a real environment. The users are emulated better, being placed on multiple servers, and there is no constant number of users. Also, WordPressBench provides a more flexible interface offering control over the simulation. The benchmark does not use traffic dumps and generates all the traffic on the fly, so no additional databases or storage is needed.

The remainder of the document is organized as follows. Section 2 describes the existing benchmarks and their issues. Section 3 offers an overview of WordPressBench with the challenges we met, scope and requirements. Section 4 offers a more detailed description of each component of WordPressBench. Section 5 describes the evaluation process of the system. Section 6 offers a few ideas for future extensions after whicg the conclusion drawn.

Chapter 2

Related Work

The most popular testing and benchmarking solution is TPC-W, which uses a custom-made retail book-store website. It was first build to generate massive HTTP load for testing hardware, but it is currently used in academia to test resource provisioning systems. It contains a workload generator defined as Emulated Browsers which emulate the Internet user behavior. The EBs create the requested number of user sessions and afterwards send request to the Web server, requesting random pages and creating new ones. The navigational pattern is defined by the Customer Behavior Model Graph (CBMG) predefined beforehand. It is basically a Markov chain matrix defining all the possible states and transitions from one state to another. Each transaction has a certain probability of being chosen, probability which is specified in the matrix. Besides the CBMG, for the EBs are also defined the workload intensity specified by the EBs number and the think time between requests.

There are three categories of user interactions defined by TPC-W, and these are obtained by varying the ratio between read-only requests (browsing activities) and read-write request (buying activities). So the browsing mix contains 95% of read-only interactions, the shopping mix 80% of read-only interactions, and the ordering mix 50% of read-only interactions.

TPC-W defines two metrics to support the benchmark's measurements. It uses Web Interactions Per Second (WIPS) at a certain scale factor, noted as WIPS@scale-factor. The scales are predefined to the following fixed values: 1,000, 10,000, 100,000, 1,000,000 and 10,000,000 books or product items. The other metric also includes the cost in its measurements, defined as $/WIPS. This is the ratio between the total price of the Web servers, database servers, commerce servers, load balancers, networks, and corresponding software, which are needed to implement the application being emulated, and the WIPS value.

One downside is that when an EB ends its session, a new user session is created, in order to maintain a constant number of users at each moment of

the simulation. The user sessions are generated from the same machine and are maintained through session cookies.

RUBBoS is similar to TPC-W, only that it models an online news website, similar to Slashdot. It is able to emulate up to 500,000 users. RUBBoS uses the idea of cache, and the users are expected to access the latest news articles and comments. The old data is moved periodically by a daemon to a database for storage.

RUBiS is a benchmark whose WebServer is an auction website, modeled after eBay.com. It defines two types of user behaviors, similar to TPC-W, which have different read-write patterns. The browsing mix is made of only read-only interactions and the bidding mix includes 15% read-write interactions. RUBiS defines a state transition matrix that indicates the probability to go from one state to another, with a random think time between interactions (between 7 seconds and 15 minutes). The load is given by the clients number, but the database contains at least 33,000 items for sale. RUBiS maintains a history of the auctions, and keeps at most 500,000 auctions in the old-items table.

One of the issues common to previous benchmarks is the fact that they don't use a real-world web application, so their websites lack complex functionalities and advanced security. The second problem is their lack of flexibility and configurability. They offer only a few pre-defined mixes, with fixed read-write ratio, with no possibility to modify them. Besides this, the constant number of user does not match the reality, where great traffic fluctuations could take place. Another issue worth mentioning is the one regarding their system design, which is not distributed. A single machine generating all the requests is not plausible in the real-world.

WikiBench is an academic benchmark especially made for testing Web server platforms. It was created to bring realism by using real traffic database dumps from the WikiMedia Foundation. WikiMedia allows to download real traffic logs of requests made to Wikipedia. WikiBench is different from the previous benchmarks, being centered on processing the WikiMedia traffic traces and transforming them into simulated requests. The benchmark offers a high degree of realism, and its results are reproducible. Despite the fixed traffic behaviour, its users have the possibility to lower the intensity of traffic requests, or change the read/write ratio with the cost of altering the original traffic traces.

WikiBench is a very good solution, which simulates Web traffic very realisticly. This application challenged us to create a solution which is not based on traffic dumps, and which creates its own traffic on the fly, while running the simulation. The user would have flexibility and total control over the simulation.

Chapter 3

Challenges

WordPressBench was created with a few objectives in mind, the most important requirements being listed below in the first section, along with a brief explanation. The functionalities of the benchmark have emerged as soon as the requirements were settled, so the functionalities present in the graphic interface are described in the second section of this chapter. While building the benchmark we met a few challenges, and the ones worth mentioning are present in the third section of this chapter, along with the chosen solutions.

3.1 Requirements

WordPressBench aims at offering a **realistic user simulation**. This goal is achieved by generating variable traffic intensity represented by variable number of users at a certain time. In order to provide control over the simulation, the benchmark user can select the fluctuation range of the users number. The realism is also given by using WordPress, a real Web server platform used by tens of millions of users and running on millions of Web servers. It provides complex functionalities, advanced security and a MySQL database. Another aspect contributing to the realism of the user emulation the fact that the user requests are made from different machines.

WordPressBench was designed with a **distributed architecture** in mind. It follows a master-slave architecture, without over-loading the master and without the risk of becoming a bottleneck. Requests are sent from the slave machines to the Web servers running WordPress, which process the log files with the statistics data.

The **flexibility of the functionalities** is another major requirement of WordPressBench. This includes the possibility of setting the traffic intensity range through the graphic interface (the number of users). Another setting is

defined by the read and write ratio, expressed in percentage, which indicate the predominant type of actions: reading or writing into the database. This benchmark, does not provide fixed configuration, or fixed scales. The results can be reproducible by using the same settings, although the workload data would be different, since it is generated randomly. Though the simulation results on the same set of Web servers, with the same settings, should remain the same.

3.2 Functionalities

WordPressBench offers a graphic user interface which eases the interaction with the benchmark. The interface allows the start and stop of the simulation, which can be fired at anytime. After pressing the "Start" button, an initiation process will be performed over the database in order to clear the previous data, after which the actual simulation will start. A graph will be shown in real-time, indicating the the time on OX axis and the response time on OY axis. The user running the benchmark has the possibility of choosing the range of traffic intensity, measured in number of users. The number of users will be fluctuating randomly within this range. Another possible setting for the simulation is the read-write ratio, expressed in percentage. The read and write are the only two atomic actions available, so their percentage is complementary to each other, summing 100%.

3.3 Challenges

This section will provide a briefly description of the challenges we encountered during the design and implementation stages of WordPressBench. We will start with the explanations of some of the design decisions we made.

Benchmarks whose purpose is stressing resource provisiong systems are defined by large HTTP requests toward the resources of a certain website. The first step in designing our benchmark was to find a solution for a reliable website, able to support a large amount of users at once. We needed a real web platform, already developed and which allows distributed support. We considered that it is not the purpose of the project to build a new platform from scratch, which would take a considerable amount of time. We decided to choose among the open-source solutions, already available. WordPress platform came out immediately, because it is already used by millions of users and it is constantly updated. It is easy to install, highly extensible and it has been proved to be very reliable.

The second major design decision regarding WordPressBench was to define the user behavior. Our goal is to simulate the HTTP request as realistic as possible. Firstly, the requests needed to be generated from different machines. Since there is almost impossible to provide a machine for each request, we decided

to have at least a group of requests generated from the same machine. This decision determined building a distributed set of workload generators. Besides distributivity, user behavior would be simulated better if there is no constant number of users. Therefore, the benchamark user has the power of modifying the number of users at any moment in time.

Regarding the development details of the workload generators, we faced difficulties when trying to login as a registered user. The login was performed using a username and password previously created by an admin user, and it allows extensive actions, like adding new pages, new blogposts, or comment as a registered user. The problem was that not only the cookies needed to be sent back, but also certain HTTP POST methods needed to be used. The solution was to analyze the login WordPress source code and determine which data was expected. TO CONTINUE

Chapter 4

WordPressBench System

4.1 System Architecture

WordPressBench has a master-slave architecture, with a Master defined by the Controller which creates the workpool with tasks according to the user's settings. The tasks are then executed by the slaves, the Workload Generators. The slaves generate HTTP requests to the Web servers running WordPress. The WordPress server is the unmodified version downloaded from the official WordPress website[].

4.2 Components

4.2.1 WordPress Web Servers

The WordPress Web Server use Apache servers and use a MySQL database. It also requires to have PHP installed on the machine. We used WordPress version 3.2 available at the development time. It allows users to read blogposts, pages and comments. This can be done by searching by keyword, by author, by date, by category, by viewing recent posts, or simply by browsing each single blogpost. Anonymous users are allowed to post comments by providing their e-mail address. Their information remains in browser's cache through cookies, until the information is overwritten.

Other write operations are allowed only for registered users. After logging in, the users are offered extended edit functionalities, like adding comments to pages and blogposts as registered user, and adding pages or blogposts. Users with 'Administrator' rights have additional functionalities, like adding new users, or clearing the data from the database, operation performed in the initiation stage of the development. TO CONTINUE

4.2.2 Workload Generator

The Workload Generator is a component running on slaves machines. It basically contains a Markov matrix with all the possible states and the probabilities to make a transition to another state.

There are two transition matrices, one for anonymous user, and another for logged-out user. While running, the next states are chosen from the transition matrix, but the information for creating the next URL is gathered from the previous HTML response page.

As was mentioned before, each state has an URL associated, so in order to move to another state, the URL needs to be send to the Web server which retrieve back the pages corresponding to the URL path. URLs have encoded different IDs, or keywords which are extracted from the previous HTML page received from the server.

In case a user loggs in, the session is maintained through cookies, which need to be sent back to the Web server at each request until the user logs out. There are states which get data from HTTP POST variables, not only from GET variables encoded in URLs. In this case, the slave needs to get the keywords, IDs, from the previus page, and send it along with other random data generated on the fly, and needed by the Web server.

While running, the Workload Generator gather statistics about the connections, requests and responses. Statistics like response time, number of errors, number of attempts before getting the response, or other such information is stored for each user, and aggregated into log files which are sent to the Master.

4.2.3 Controller

4.2.4 Graphic Interface

4.3 Workflow

- the connection between components
- how does it work as a whole

4.4 Tools and Technologies

- the technologies used: WordPress, Apache, MySQL, Java, HTTP protocol, TCP connections

Chapter 5

Evaluation

5.1 Evaluation System

5.2 Evaluation Results

Chapter 6

Conclusion

6.1 Future Work

6.2 Conclusion

Bibliography