Vrije Universiteit Amsterdam
Computer Science Department

A WordPress-based Web Hosting Benchmark

Master Thesis
Internet and Web Technology

Scientific Supervisor:
Dr. Guillaume Pierre
Second Reader:
Corina Stratan

Author:
Alexandra Andronescu

August, 2011
Amsterdam

To my family

Abstract

Currently there exists multiple stress tools and benchmarks for testing Web hosting systems. One major downside is their lack of control over the testing process, along with unrealistic user simulation. To overcome these problems, we designed and implemented WordPressBench, which is a WordPress-based benchmark whose purpose is to test various Web hosting systems. It generates fluctuating traffic intensity through a variable number of emulated users. The distributed design of WordPressBench makes possible to support a large amount of traffic, with no bottlenecks. 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, news and blog feeds are also very popular. 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 with 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 real workload as precisely as possible. This is done by analyzing and modeling user 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 are used for various types of testing, one of them being functional testing, when the requirements and the system's functionalities are evaluated. Non-functional benchmark testing includes performance testing, which measures the response time, the scalability and reliability of the system. Benchmarks also perform load testing, which checks if the system is scalable and supports the demand of a large number of users. Through stress testing the benchmarks check the system's reliability, availability and error handling.

Benchmarks are also used for testing various components of the system under test. The benchmark sends different types of requests which target each component. The Web server is the main component of the Web hosting system, being the contact point where all requests are sent. This is why Web servers 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 caching server, an image server, or a multimedia server, depending on the system's purpose and on what kind of information needs to be stored. These servers are tested by requesting a large amount of the data stored on these servers.

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 [1], which is viewed as an industry standard for testing hosting systems, despite the fact that it is officially outdated. Although it was originally built for testing hardware by generating massive load, nowadays its modified versions are still used in academia for testing resource provisioning systems. TPC-W simulates a retail book-store website and emulates thousands of browsers. Similarly, RUBBoS [2] and RUBBiS [3] 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 are very limited. They do not represent realistic Web sites 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 to compare other systems, or to repeat an experiment, 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 benchmark that emerged lately is WikiBench. It is an academic benchmark, made for testing server platforms, and which uses real traffic by manipulating WikiMedia database traffic dumps. Its main advantage is that 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. One problem is that WikiBench is not flexible and it does not offer a lot of control over the simulation process.

We have designed and developed WordPressBench in order to surpass some of the previous problems. It aims at bringing flexible user controls and more realistic Web traffic simulation. It generates the traffic dynamically, and it is not based on old traffic dumps. WordPressBench offers more control and flexibility, making the simulation process more precise and easier to adapt to user's simulation requirements. The users can simulate better the desired test cases. This thesis presents the challenges, design and implementation of WordPressBench.

WordPressBench is a Web-hosting benchmark for testing various Web-server platforms by emulating real-user behavior. Its web-servers run WordPress [5], a real website, used by a large number of Internet users. The system is designed with a distributed worker configuration, having a master-slave architecture. The workers are the ones generating the workload towards the WordPress Web site placed on the system under test. The users are emulated better, being placed on multiple servers, and the number of emulated users can be modified while running 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. Chapter 2 describes the existing benchmarks and their issues. Chapter 3 offers an overview of WordPressBench with the challenges we met, scope and requirements. Chapter 4 offers a more detailed description of each component of WordPressBench. Chapter 5 describes the evaluation process of the system. Chapter 6 offers a few ideas for future extensions and afterwards we draw the conclusion.

Chapter 2

Related Work

2.1 TPC-W

The most popular testing and benchmarking solution is TPC-W, which uses a custom-made retail book-store website. It was built to generate massive HTTP load for testing hardware, but it is currently used in academia to test Web hosting systems. Its main advantage is that it generates the workload dynamically, so no database or traffic dumps are used. This feature also allows a more flexible simulation because it has no other restrictions than the user's settings. 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 content.

Another advantage of TPC-W is the navigational pattern which is defined by the Customer Behavior Model Graph (CBMG). It is basically a Markov chain matrix defining all the possible states and transitions from one state to another. The CBMG matrix is defined beforehand and allows the customization of the users' behavior [7]. Each transaction has a certain probability of being chosen, probability which is specified in the matrix. Besides the CBMG, the EBs are defined by the workload intensity specified by the number of EBs and the think time between requests [10].

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 [9].

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, and represent the number of items in the inventory. The purpose of scaling the user number to the number of items was to avoid getting incorrect results for a large number of users and a very small database, but in the same time it limits the user control over the simulation. The other metric includes the cost in its measurements, defined as $/WIPS [8]. It is defined as the ratio between the total price of the System Under Test (SUT) and the WIPS value. SUT includes all the software costs including the maintenance, and the hardware costs like database servers, commerce servers, load balancers, internal networks needed to implement and run the application.

Another weakness is that when an EB ends its session, a new user session is created, in order to maintain the same constant number of users at each moment of the simulation. The interface does not allow to modify the number of users during the simulation. The user sessions are generated from the same machine and are maintained through session cookies [11].

2.2 RUBBoS and RUBiS

RUBBoS [2] 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 [3] is a benchmark whose Web server is an auction website, modeled after eBay.com. It defines two types of user behavior, 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, and 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.

2.3 WikiBench

WikiBench [2] 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 main strength of WikiBench is the delivery of a high degree of realism and reproducible results useful when measuring different systems. Despite the fixed traffic behavior, 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 because it simulates Web traffic very realistically and the simulations are reproducible when provided the same Web traffic traces. One major downside of WikiBench is that it requires traffic dumps which take up space and restrict the control over the simulation and user behavior.

WikiBench 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

Before designing the benchmark, we build up a list of requirements that needed to be satisfied by the benchmark. The first section describes these requirements. The functionalities and the results yield by the benchmark are described in the second section. In the design phase, we encountered a few impediments while trying to satisfy the requirements. These problems, along with a few generic steps for building a Web hosting benchmark are listed in the third section.

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.

WordPressBench was designed with a **master-slave architecture** in mind. By following this architecture, scalability is provided. Therefore, it does not overload the master when a large number of users are added and there is no risk of creating a bottleneck. Requests are sent from the slave machines to the Web servers running WordPress, which process the log files with the statistics data.

Flexibility 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 does not offer a graphic interface, but generates log files ready to be send input to generate graphs. The log files contain the average response time and the corresponding relative time from the beginning of the simulation when it was measured. The time will be displayed on OX axis, and the average response time on the 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 around the specified number. 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 for stressing Web hosting 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 reliable website platform, 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 distributing the user requests, user behavior would be simulated better if there is no constant number of users. Therefore, the benchmark user has the power of modifying the number of users at any moment in time.

The third issue that we faced was finding a way to implement read-write ratio of operations. Since for every write there must be at least one read to get the information first, we decided to have a read-only and read-write ratio. We first decided to have two types of websites, one for read-only working as a regular website, and one for blogging. When implementing, we realized that for the regular website we needed to generate content, and this had to be proportional with the number of requests. We finally decided to have a single blogging website for both read-only and read-write users whose behavior is determined by different transition matrices.

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 user-name and password previously created by an administrator user, and it allows extensive actions, like adding new pages, new blog-posts, 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.

We encountered a few design and implementation problems when building the real-time system. Our goal was to create an application which allows the benchmark user to modify the traffic intensity and view the statistics during the simulation. To do this, a large number of TCP requests are exchanged permanently between the Controller and all the Workload Generators. We used multi-threaded servers, to avoid blocking operations and a custom-made communication protocol between the components.

Chapter 4

The WordPressBench System

4.1 System Architecture

WordPressBench needs to be able to issue very large amounts of requests to the system under test. We therefore designed it around a master-slave architecture. The master Master is defined by the Controller which creates the work-pool 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 [6]. Figure 4.1 pictures the system with all the components and relations between them. Each component will be described in detail in the following sections.

4.2 Components

4.2.1 WordPress Web Servers

The WordPress Web Server use Apache servers and 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 blog-posts, 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 blog-post. 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.

Additional write operations are allowed only for registered users. After logging in, they have access to extended edit functionalities, like adding comments

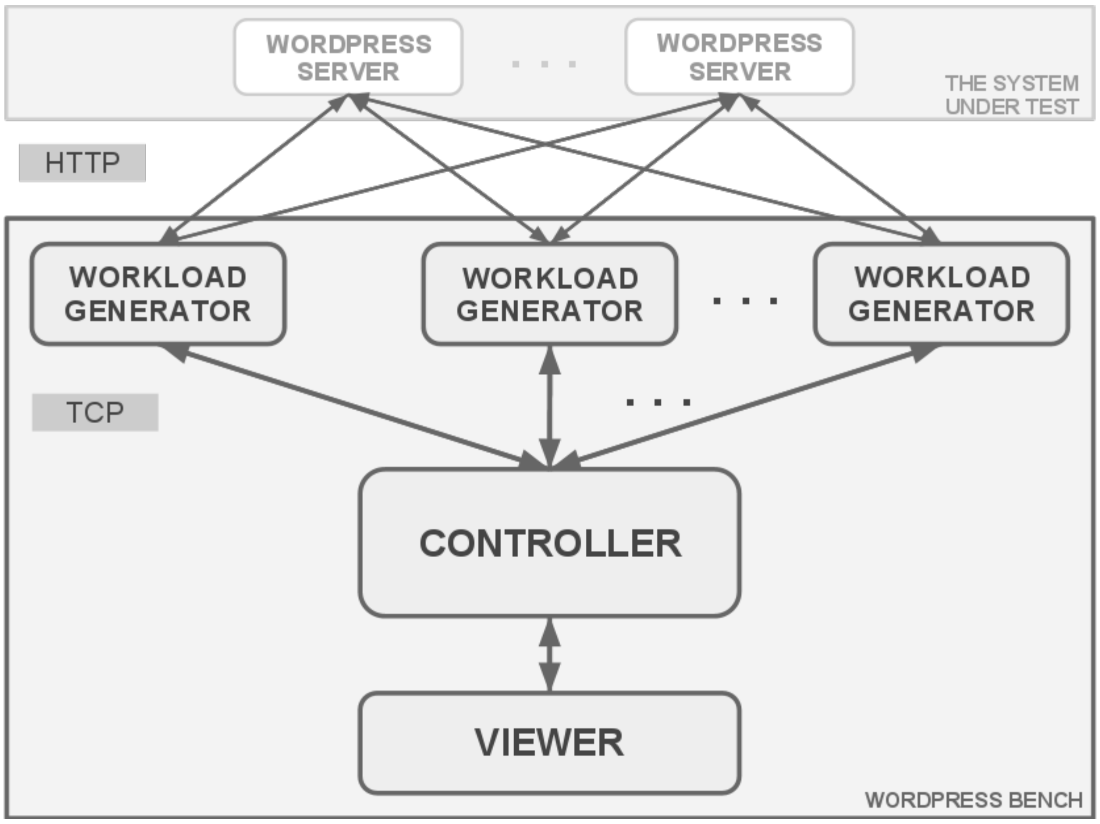

Figure 4.1: System Design

to pages and blog-posts as registered users, or adding pages, blog-posts and blog-post categories. Users with 'Administrator' rights are allowed to create new users, or to clear the data from the database, operation performed in the initiation stage of the development.

4.2.2 Workload Generator

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

There are three transition matrices, one for anonymous user, one for logged-out user, and another for anonymous user. This is mainly because the registered user has access to additional states and the flow of actions could be totally different from an anonymous user. The read-only user has the most restrictive operations.

The transition matrix describes the users' navigational pattern, by specifying how users navigate through the website, the functions they are allowed to use in a certain state and how often, and the frequency of transitions from one state

	HOMEPAGE	PAGE	SEARCH	BLOGPOST	ARCHIVE_MONTH	CATEGORY	AUTHOR
HOMEPAGE	0.05	0.15	0.15	0.15	0.15	0.15	0.15
PAGE	0.15	0.15	0.15	0.05	0.1	0.1	0.1
SEARCH	0.05	0.05	0.05	0.05	0.05	0.05	0.05
BLOGPOST	0.2	0.2	0.2	0.2	0.2	0.25	0.2
ARCHIVE_MONTHLY	0.1	0.1	0.1	0.1	0.1	0.1	0.1
CATEGORY	0.1	0.05	0.05	0.05	0.05	0.05	0.1
AUTHOR	0.05	0.05	0.05	0.05	0.05	0.05	0.05
BLOGPOST_PAGED	0.1	0	0	0.1	0	0	0
SEARCH_PAGED	0	0	0.1	0	0	0	0
ARCHIVE_MONTHLY_PAGED	0	0	0	0	0.15	0	0
CATEGORY_PAGED	0	0	0	0	0	0.1	0
AUTHOR_PAGED	0	0	0	0	0	0	0.1
WORDPRESS_WEBSITE	0.05	0.05	0.05	0.05	0.05	0.05	0.05
ADD_ANONYMOUS_COMMENT	0	0.1	0	0.1	0	0	0
LOGIN_SEND_CREDENTIALS	0.15	0.1	0.1	0.1	0.1	0.1	0.1
LOG_OUT	0	0	0	0	0	0	0
ADD_COMMENT	0	0	0	0	0	0	0
ADD_POST	0	0	0	0	0	0	0
ADD_PAGE	0	0	0	0	0	0	0
ADD_CATEGORY	0	0	0	0	0	0	0
	1	1	1	1	1	1	1

Figure 4.2: Transition matrix for a Logged-in user

to another. Users interact with the WordPress Website through sessions, which are sequences of consecutive requests to bring different pages from the WordPress Website. After each request, a transition is made from the current state to another state. The next state is chosen from the transition matrix which indicates for each state which are the next possible transitions. The matrix columns indicate the current state, and the matrix lines are the states which could follow. A zero in the matrix means that there is no possible transition from current state to the state indicated by the line. A non-zero value in the matrix represents the probability to make a transition to the state indicated by the line. So when choosing the next state, only a non-zero value will be chosen. For building the transition matrix we chose bigger probabilities for widely used functionalities, like reading a blog-post or posting a comment. Figure 4.2 depicts the transition table for a registered user who can log-in and perform additional write operations.

The HTTP requests require URLs which specify what page should be brought back. Additional information is often needed, especially for users who are logged-in, so information is either encoded in POST variables, or in cookies. The information which is requested by the WordPress servers includes user information, text for publishing, IDs, options, nonces (random number issued in previous HTML content received from WordPress server), passwords and usernames. Some of this information is generated by the Workload Generator, but the IDs or nonces are gathered from the previous HTTP page just received from the WordPress server. Cookies also need to be send back to the server, exactly as they have been received. They are used to maintain the logged-in user session, and they are reset as soon as the user logs-out. If the request is successful, and the correct page is sent back, the session makes a transition to the desired state. To simulate users more realistically and knowing that real users spend a different amount of time between requests, each emulated user has a random "think time" of at least 7 seconds between requests. For avoiding fast automated requests, WordPress rejects requests sent in a very small amount of time, so the "think time" was absolutely necessary.

In case a user logs 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 previous page, and send it along with other random data generated on the fly, and needed by the Web server.

While running, the Workload Generator gathers statistics about the responses time. Statistics like response time, number of errors, number of attempts before getting the response, or other such information is stored for each user along with a timestamp, and aggregated into log files which are sent to the Master at the end of the simulation.

4.2.3 Controller

The Controller is the main command component. It has control over the Workload Generators and sends commands to them through TCP messages. The Controller can send five types of messages:

- start simulation
- stop simulation
- create users
- add users
- get logfile
- exit

For starting the simulation, the Controller indicates the username range to be used by each Workload Generator, along with the read-write ratio. The number of users is computed afterwards for each Workload Generator. Each of them uses distinct usernames to avoid overlapping. The command for creating more users has the effect of registering a certain amount of users to the WordPress server(s). This command is generally followed by either the simulation start, or the addition of more users. The command for adding more users is send during the simulation process, when the number of user is increased, so new user sessions are initiated. The command for getting the log file simply requests the log files with statistics from each Workload Generator. This operation is usually performed after the simulation has been stopped, when data is not written anymore into the log files. Afterwards, the log files are aggregated and processed by the Controller, and data is ready to be send as input for generating graphs. The exit command stops all the components, so the Controller cannot send commands to the other machines anymore.

The log files generated by each Workload Generator output on the first column the time in seconds relative to the beginning of the simulation. This includes the initialization time. On the second column is displayed the response time for the current state, and on the third column is displayed the name of the current state. The Controller takes the log files from each Workload Generator and aggregates them by the first column, computing the average response time in one second. The third column from the initial log files is ignored.

The average response time is measured in milliseconds for a finer granularity, and the relative time is measured in seconds. We chose to have the relative time from the beginning of the simulation and not the real time. This decision was made in order to avoid the case when the machines are not synchronized, so the local time for each measurement would be different.

The Workload Generators compute additional statistics, and at the end of the log files are displayed additional statistics for each state. The percent, the number of transitions, number of errors, the minimum time, the maximum time, and the average time are computed for each transitioned state. The overall average time for the current Workload Generator is computed and displayed at the end of the log file.

4.3 Component Connections

Data flow between components is sent through two types of protocols: HTTP and TCP. HTTP protocol is used for sending GET methods with cookies and POST data to the WordPress Web server. HTTP messages are sent back in response with the requested pages, or error codes if necessary.

TCP protocol is used to send commands from the Controller to the Workload Generators. The Controller's commands have designated code values, so a number from 0 to 5 encodes one of the six possible commands listed above. This code could be followed by two numbers representing the user id range to be used by the current worker. A third number represented by a double value indicates the read-only percent of the requests. After each command an acknowledgement message is sent to the Controller to confirm that the message has been received and the requested operation was executed. In the case of the "Get log" command, the log files are sent through the same TCP connection and written to local log files on the Controller machine.

4.4 Workflow

After the benchmark simulation is started, the first step is to delete all the data from WordPress server's database. This is performed by a single thread with "Administrator" rights on a Workload Generator. Afterwards the Controller sends command to a Workload Generator to create a number of users. This operation will be performed similarly to previous data deletion step.

For starting the simulation each Workload Generator receives a user ID range. Their count represents the number of emulated users, so this is the number of threads which will be created. Each thread emulates a user and sends HTTP requests to the WordPress Web server placed on the system under test. The states are chosen randomly from one of the transition matrix (read-only matrix, logged-in matrix, or logged-out matrix) and for each request the response time is logged along with the current time and state name.

The Controller can send other additional messages to the Workload Generators, like adding more users. In this case an additional message will be sent in order to create more users, so a thread on a single Workload Generator will create the additional number of users if necessary. Afterwards each machine will add the corresponding users by creating more threads which emulate users, using the newly created user IDs.

When the Controller requests the end of the simulation, each thread from the Workload Generators will resume, but the machine will continue to receive new commands, the most common one being "Get logfile". In this case the log files are sent through TCP connection and aggregated on the Controller. The exit command sent by the Controller is the one which completely stops all the components. The simulation yields a number of log files which are ready to be processed.

4.5 Tools and Technologies

As mentioned before, we decided to use an existent Web server platform. We used WordPress because it is a very mature and reliable platform. Is is widely used by millions of users and therefore it went through extensive testing and constant updates. WordPress requires an Apache server for running the Web server. It listens HTTP requests coming from the Workload Generators.

WordPress is using a MySQL database as well, where it stores the data displayed on its website: user credentials, user settings, and Web blog content like blogposts, comments, pages, categories and details about each of them. WordPress as a predefined schema of tables which is created at each setup. The benchmark interacts directly with the MySQL database by using commands to delete the data at the beginning of each simulation.

The system is implemented entirely in Java. We made this choice because it offers ease of development, a good API and offers good support for TCP and HTTP protocols usage. We used Java sockets and performed GET requests to the Web servers on the system under test.

Chapter 5

Evaluation

5.1 Evaluation System

For evaluating the system we used a few machines to test if the system components are able to communicate from independent machines. We used one machine for each type of component: one for the WordPress server, one for the Workload Generator, and a third one for the Controller component. Each of them had an AMD Phenom 9600B Quad-Core processor with 2293 MHz CPU and 2GB of memory. For the last part of the tests we used two machines with Intel Core 2Duo Processor with 2.00GHz CPU and 2GB of memory. On one machine was placed the WordPress server, and on the second one the Workload Generator and the Controller.

5.2 Evaluation Results

During system evaluation we varied a number of inputs:

- number of users
- read-only ratio
- simulation length

We ran tests with a constant number of 50 users, so this is why the resulting graphs Figure 5.1, Figure 5.2 and Figure 5.3 have a constant response time, situated within a range. We then ran the benchmark with an increasing number of users, from 50 users to 80 and the response time increase can be observed in Figure 5.4. We also ran tests with a slightly larger number of users, 100 and 300, and the results are depicted in Figure 5.5 and Figure 5.6.

Afterwards we varied the read-only ratio from value 0.4 in Figure 5.1, Figure 5.5 and Figure 5.6, to value 0 in Figure 5.2, and value 1 in Figure 5.3. The simulation length was 15-20 minutes, excluding the initialization step. We observed that user creation step take a quite large amount of time, so this is why the simulations with larger number of users have a bigger relative time (placed on OX axis).

The evaluation process unveiled a few issues, the biggest one being the long login time and user creation time. This increased the overall response time dramatically, and the spikes present in the graphs are caused by these long delays.

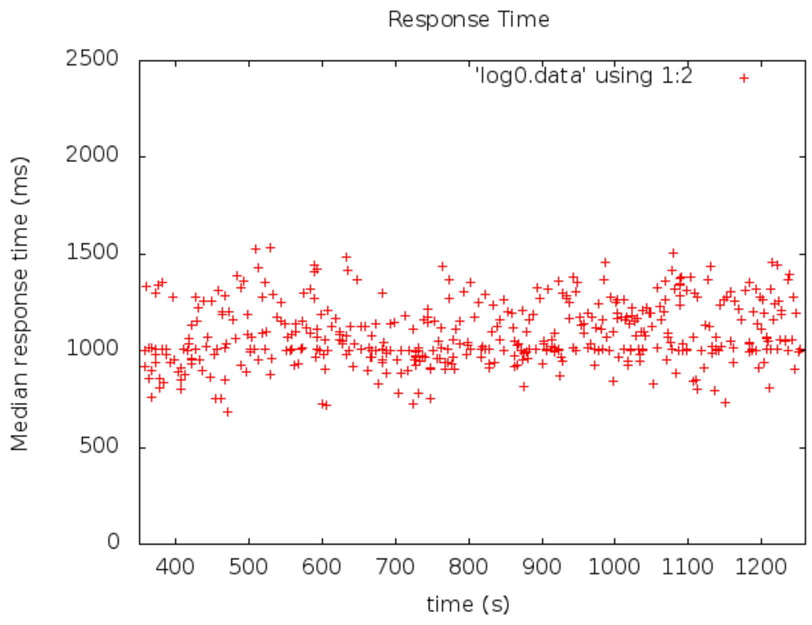

Figure 5.1: A benchmark evaluation with 0.4 Read-only ratio

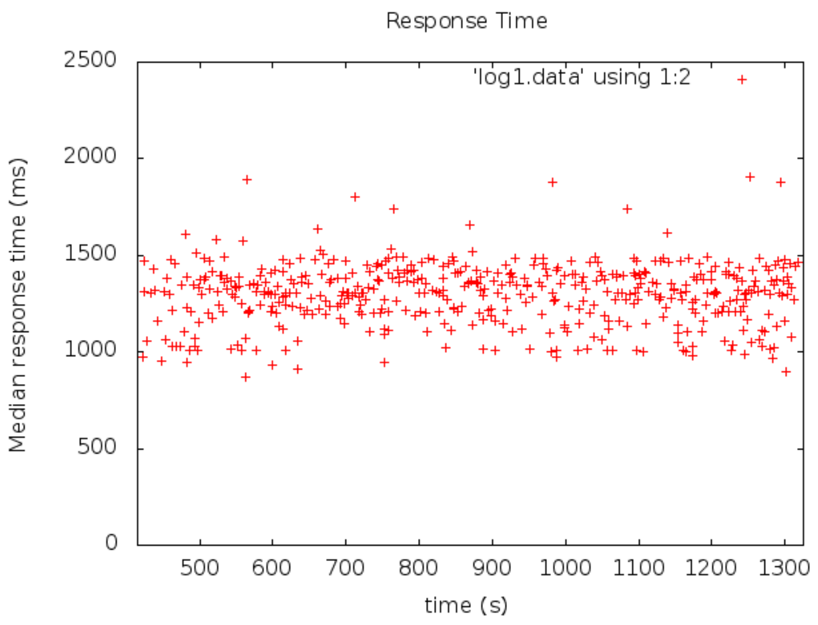

Figure 5.2: A benchmark evaluation with 0.0 Read-only ratio

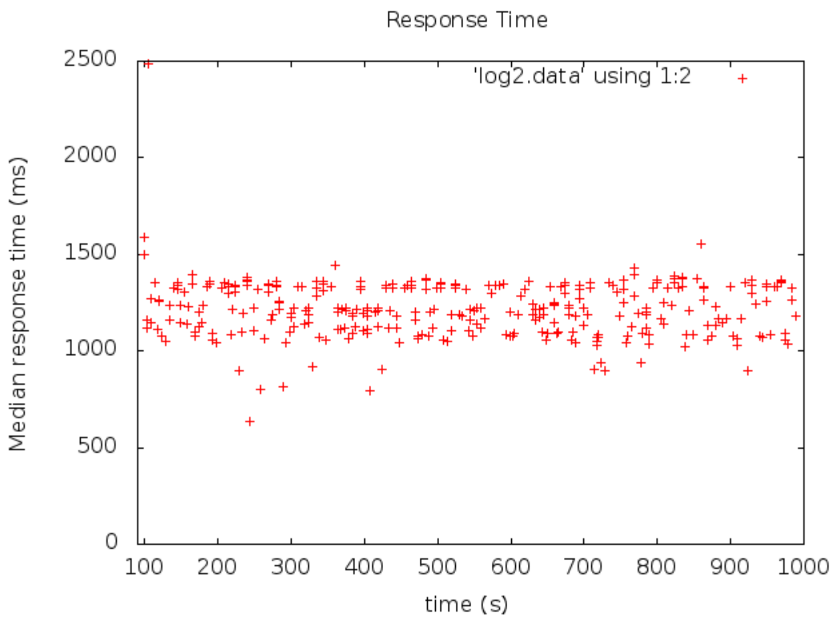

Figure 5.3: A benchmark evaluation with 1.0 Read-only ratio

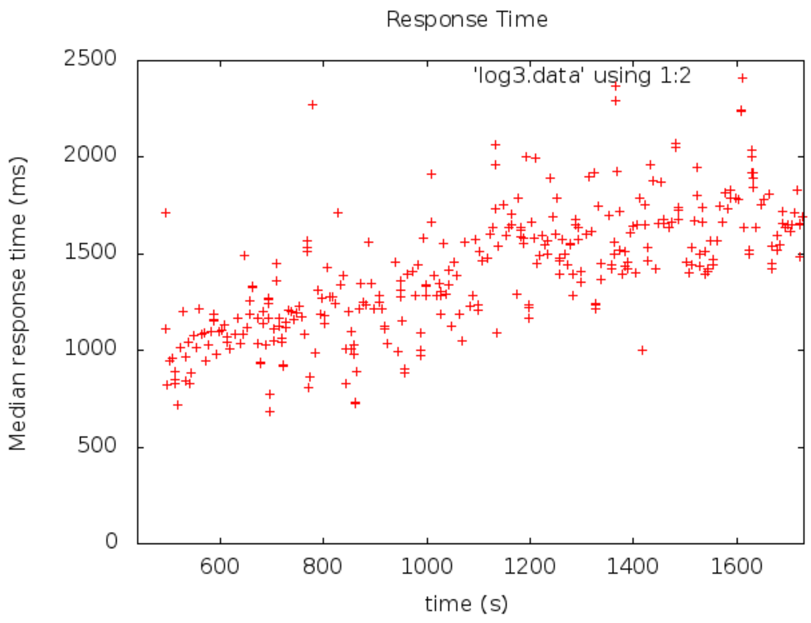

Figure 5.4: A benchmark evaluation with 0.4 Read-only ratio and increasing number of users

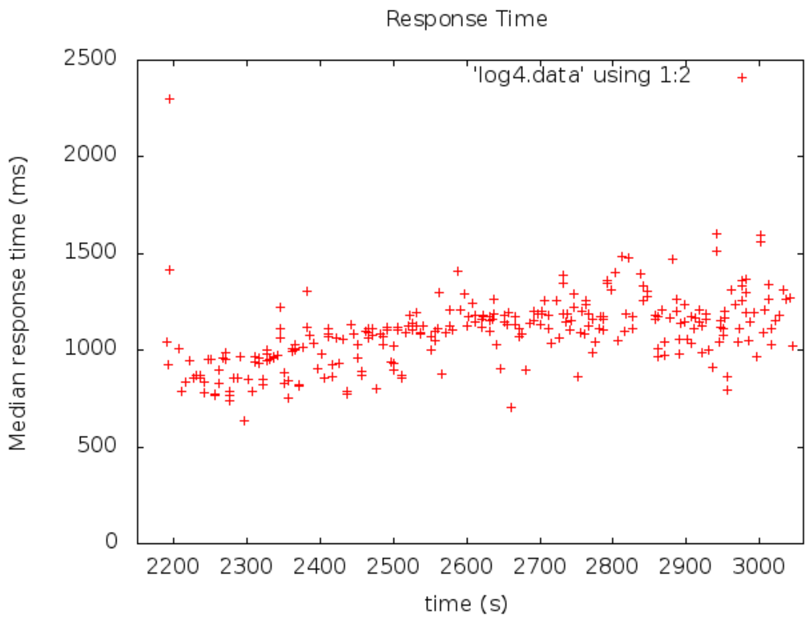

Figure 5.5: A benchmark evaluation with 0.4 Read-only ratio and a larger number of users

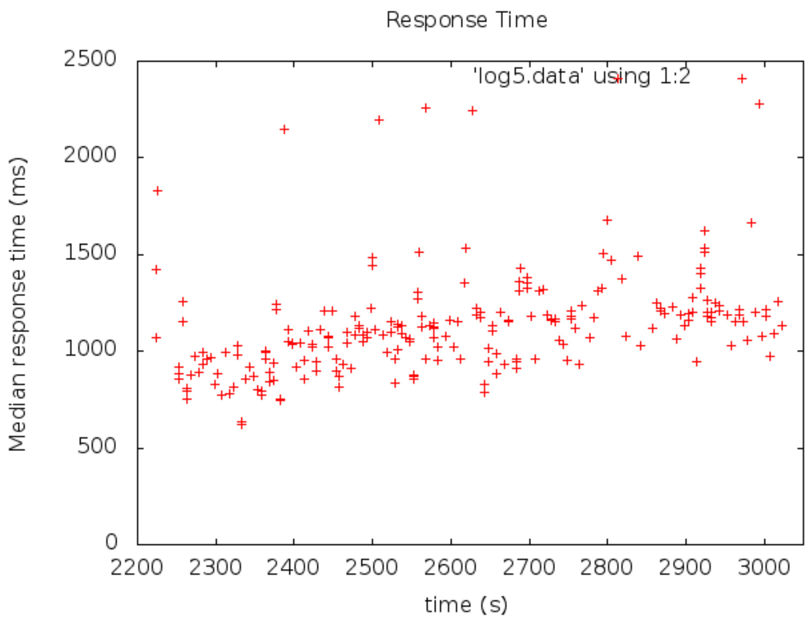

Figure 5.6: A benchmark evaluation with 0.4 Read-only ratio and a larger number of users

Chapter 6

Conclusion

6.1 Future Work

The current WordPressBench application is very simple, and performs only the basic functionalities. There are a lot of additions and bug fixes that could be implemented to improve the application and some of them are described below.

The system was first designed with a graphic interface, but due to lack of time it was not implemented anymore. A graphic interface would definitely be very useful because it would ease the input provisioning. The interface must have input controls for setting the number of users and the user should be allowed to set this variable during the simulation time as well. The interface should also allow to set the read-only ratio using percentage of values between 0 and 1.

Another desirable improvement concerns the display as well and represents the display of measurements in real-time. It is not an easy task, because this implies a periodic transmission of log files between Workload Generator and Controller and permanent aggregation of the log files by the Controller. In order to create the impression of real-time simulation, these operations have to be performed very often.

The transmission of messages between Controller and Workload Generators is done through TCP connections. They have initialization overhead and require more connection time, and the messages are not sent very frequently. To improve this issue, the connection could be replaced with a UDP connection in order to exchange short and fast datagrams.

The transition matrix is not generated very realistically. The current matrix contains probabilities based on empirical user behavior. An good improvement which would add realism to the simulation is to have a matrix based on measurements on real user behavior. The probabilities to go to certain states should be measured and put in relation with the read-only ratio.

During the evaluation process we discovered that the login state takes a longer amount of time, leading to spikes in graphics and a response time increase in certain states. If this issue is solved, the results would be more precise and the overall simulation would take less time.

User creation is another step which takes a larger amount of time than expected, and if fixed would have the same results as described above. This problem causes a delayed start of the simulation which can be observed in graphs on OX axis, where the relative time has large values, especially at simulations with larger amounts of users.

Finally, we consider that our evaluation process was a very basic one, and more extensive testing is needed. Using multiple machines for the Workload Generators would be a very good way of testing the scalability, as well as the Controller's capacity to deal with a lot of commands and log files. Flash crowds were one of the system's purpose, but we did not have enough time to simulate them either, so this could be one of the future tasks.

6.2 Conclusion

The need of proper benchmarks has increased in recent years, because the existent ones are far from bringing the desired results. They have many limitations, like the lack configurability, realism, and scalability.

We designed WordPressBench to surpass the issues of present benchmarks. WordPressBench is a Web-hosting benchmark for testing various Web-server platforms. We designed the system with realistic user simulation in mind, achieved by using a real WordPress website. We aimed to bring scalability by adopting a master-slave architecture which allows the simulation of large number of users. Flexibility was the third goal, achieved by allowing the user to have control over the simulation. The user can set the number of users in real-time, the read-only ratio, or the simulation time.

WordPressBench is based on WordPress platform and emulates users by generating user requests on the fly, using pre-defined transition matrices. The system contains several Workload Generators which send HTTP requests to the WordPress website placed on the system under test. The Controller is the central component, sending commands to the Workload Generators and aggregating the log files.

The Workload Generators chose their states randomly from the transition matrices. They generate their URLs using data retrieved from previous requests and send them along with POST methods and cookies. There are three types of transition matrices depending on the user type: anonymous, registered, or read-only. The Workload Generators measure the response time at a certain moment in time, relative to the start of the simulation, and write these time measurements into log files. The files are sent to the Controller after the simulation resumes.

We evaluated the system by varying the number of users, the read-only ratio and the simulation time. The average response time per second at each second in time were plotted. We observed that the delay in login time and user creation creates spikes in graphs and delays the simulation start. We leave these issues to be solved in the future, along with a few other improvements.

To improve the current system we suggest a graphic interface which would permit the user to control the simulation process. A great feature of the graphic interface would be the display of results in real-time, or at least at very short time increments. We also propose the replacement of TCP communication with UDP datagrams. We also propose generating the transition matrix more scientifically, using precise measurements.

Bibliography

[1] TPC-W specification available at tpc.org/tpcw/TPC-W_Wh.pdf.

[2] RUBBoS website jmob.ow2.org/rubbos.html.

[3] RUBiS documentation rubis.ow2.org/download/rubisva_v1.0.pdf.

[4] WikiBench documentation available at wikibench.eu/wp-content/uploads/2010/10/van-baaren-thesis.pdf.

[5] WordPress official website wordpress.org/.

[6] WordPress platform for download at wordpress.org/download/.

[7] Daniel A. Menascé, *TPC-W: A Benchmark for E-commerce*, IEEE Internet Computer, May/June 2002.

[8] Wayne D. Smith, *TPC-W*: Benchmarking An Ecommerce Solution*, Revision 1.2. Intel Corporation.

[9] Cristiana Amza, Alan L.Cox, Willy Zwaenepoel, *Conflict-Aware Scheduling for Dynamic Content Applications*, Fifth USENIX Symposium on Internet Technologies and Systems, 2003.

[10] Swaminathan Sivasubramanian, Guillaume Pierre, Gustavo Alonso, Maarten van Steen, *Analysis of Caching and Replication Strategies for Web Applications*, Internet Computing, IEEE , 2007.

[11] Harold W. Cain, Ravi Rajwar Morris Marden, Mikko H. Lipasti, *An Architectural Evaluation of Java TPC-W*, Seventh International Symposium on High-Performance Computer Architecture, 2001.