Analysis and Evaluation of Web Application Performance Enhancement Techniques

Igor Jugo[1], Dragutin Kermek[2], and Ana Meštrović[1]

[1] Department of Informatics, University of Rijeka,
Radmile Matejčić 2, 51000 Rijeka, Croatia
{ijugo,amestrovic}@inf.uniri.hr
http://www.inf.uniri.hr
[2] Faculty of Organization and Informatics, University of Zagreb,
Pavlinska 2, 42000 Varazdin, Croatia
dkermek@foi.hr
http://www.foi.unizg.hr

Abstract. Performance is one of the key factors of web application success. Nowadays, users expect constant availability and immediate response following their actions. To meet those expectations, many new performance enhancement techniques have been created. We have identified almost twenty such techniques with various levels of implementation complexity. Each technique enhances one or more tiers of the application. Our goal was to measure the efficiency and effectiveness of such techniques when applied to finished products (we used three popular open source applications). We argue that it is possible to significantly enhance the performance of web applications by using even a small set of performance enhancement techniques. In this paper we analyse these techniques, describe our approach to testing and measuring their performance and present our results. Finally, we calculate the overall efficiency of each technique using weights given to each of the measured performance indicators, including the technique implementation time.

Keywords: Web application, performance, enhancement, techniques.

1 Introduction

Web applications (WAs) have become ubiquitous allowing anyone, even with only basic IT knowledge, to start an online business using a free and open sourced WA or a commercial one. There are three basic factors that have led to great importance of their performance today. First, the spread of broadband Internet connections has changed visitors expectations and tolerance to waiting for the application to respond, lowering the expected time to 1 or 2 seconds. The second factor is the increasing workload generated by the constantly growing number of Internet users. The third factor is the new usage paradigm (user content creation = write-intensive applications) that has been put forth by Web 2.0. All this has increased the pressure on performance of web applications. Our research had the following objectives: a) make a systematic overview of various techniques

S. Casteleyn, G. Rossi, and M. Winckler (Eds.): ICWE 2014, LNCS 8541, pp. 40–56, 2014.

for enhancing WA performance, b) experimentally test, measure and evaluate the effectiveness of each technique, and c) measure the effect of these techniques on WA quality characteristics. The hypotheses we set were: a) it is possible to significantly increase application performance on the same hardware basis, independently of its category (type), by using even a small subset of performance enhancement techniques and b) implementation of performance enhancement techniques has a positive effect on the quality characteristics of such applications (efficiency, availability, reliability). The first hypothesis will be confirmed by achieving a 30% or more increase in throughput, while keeping the 90% of response times under 2 seconds and the average CPU usage under 70%. The second hypothesis will be confirmed by achieving a 10% or more decrease in average CPU usage, while still enhancing the throughput for at least 30%. In order to confirm our hypotheses experimentally, we selected three well known open source applications of different categories. The applications that we selected were: a) Joomla content management system (marked APP1 in this paper), b) PhpBB online community system (APP2) and c) OsCommerce e-commerce system (APP3). We have selected these applications because they have been under development for a long period of time; they are the solution of choice for many successful web companies and are used by millions of users every day.

The paper is organized into six sections. Section Two describes the theoretical foundation of this research and situates the work in the area. Section Three presents some motivations for performance enhancement of web applications, possible approaches and an overview of our analysis of performance enhancement techniques. In Section Four we present our experiment and discuss the results. In Section Five we calculate the effectiveness of each technique which suggest implications for the problem of performance enhancement technique selection in relation to the type and workload of WA whose performance we are trying to enhance. Finally, Section Six draws conclusions and suggests further work.

2 Background

There are three basic approaches to enhancing performance of WAs. While caching and prefetching are well known from other areas of computing, response time and size minimization is a relatively new approach developed by some of the most visited applications on the World Wide Web (Web) today. In this section we will point out some of the most important work within these approaches. Caching is one of the oldest methods of performance enhancement used in various information systems. In the area of web application development, caching has been analyzed in [18], [14], [5], [2] and [24], while various caching strategies have been analyzed in [25]. Another expanding area of performance enhancement is prefetching, i.e. preparing and/or sending data that is most likely to be requested after the last request. Domenech analyzed the performance of various prefetching algorithms in [8] and created prefetching algorithms with better prediction results in [7]. One of the crucial elements of prefetching is the prediction on which content will be requested next. Prediction is usually based on the analysis

and modeling of user behavior as described [9], or by extracting usage patterns from web server logs, which has been done in [16] and in [11]. Adding a time component to better determine the optimal time for prefetching web objects was suggested in [15]. The latest approach is response time and size minimization. This approach is based on the experiences and methods developed by professionals that constructed some of the most popular applications on the Internet today, such as like Yahoo.com [26] and Flickr.com [10]. As said earlier Web 2.0 and AJAX have caused a paradigm shift in WA usage, which also brought on a change in the nature of WA workload. This was analyzed in [21],[20] and in [19]. Throughout 2009 and 2010, authors have studied the possibilities for the overall application performance enhancement [22], [6]. In this paper we analyze techniques based on all the mentioned approaches and measure their effect on the performance of WAs.

3 Performance of Web Applications

The quality, and with that, the performance of WAs is primarily defined by the quality of its architecture. Badly planned elements of application architecture can limit or completely block (by becoming a bottleneck) the expected or needed levels of performance. With the new focus on the importance of application performance, production teams have begun to implement performance risk management in all phases of its lifecycle. According to the Aberdeen group research [1], companies that have adopted performance management in the form of the "Pressures, Actions, Capabilities, Enablers" (PACE) model have seen a 106% increase in availability, 11.4 times better response times and 85% problem solving rate before the problem becomes obvious to their users. However, there are already many WAs used worldwide, and with the rise in the number of users, their performance has to be enhanced to meet the desired service levels. The motivation for performance enhancement usually comes from three main directions: a) the desire to increase the number of users or profit, b) expected and planned peak workload periods and c) performance enhancement based on a business plan, or inspired by a real or expected (projected) performance problem with a goal of ensuring availability and efficiency under increased workload. Performance enhancement is always a tradeoff between the number of opposing factors like hardware and working hours investment limits, desired response times and throughput values, etc. In our research we had to balance between two limiting factors (response time and CPU usage) while trying to maximize a third one (throughput).

These factors can be visualized as given in Figure 1. The X axis shows the response time, with a maximum value of eight seconds, which was considered by some authors to be the limit of tolerable waiting time a few years ago. We believe that today, this limit for an average user is much lower so we have set the average response time for 90% of requests to 2 seconds. The Z axis displays the average CPU usage during load tests. The usage limit here has been set to 70% according to some best practices in the hardware industry (constant usage over 70%

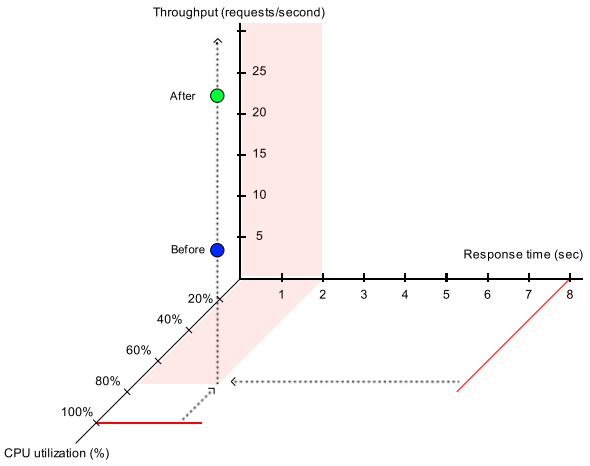

Fig. 1. Performance enhancement area

significantly raises malfunction probability and shortens the Mean Time Between Failure (MTBF), along with the Total Cost of Ownership (TCO)). In this way, we have set the margins of an area within which we can try to achieve the highest possible throughput (Y axis) by implementing various performance enhancement techniques.

Performance enhancement requires a complete team of experts from all fields of WA development - from front-end (HTML, CSS, JavaScript) to back-end (database design, SQL queries, DBMS configuration and administration) in order to plan and implement performance enhancement techniques, as well as verify changes through benchmarking. Such teams must also check the performance of application architecture at each level, as some techniques may have hidden effects that can propagate throughout the application and even cause a decrease in the performance at another level. Some techniques require additional services that must be installed on the server. This increases the complexity of application architecture and increases CPU and memory usage (if additional services are running on the same server), which has to be taken into account when doing post-implementation load testing. When trying to enhance the performance of a WA, a starting approach can be general systems theory that considers the application to be a black box with a large number of requests per second as input, and a large number of responses per second as output. When we start to decompose the black box into subsystems, we identify the ones that spend most of the time needed to generate responses. If one subsystem takes 80% of this time, then our choice is to try to enhance the performance of that subsystem, instead of another subsystem that spends just 5% of the time. After enhancing performance of one subsystem, another becomes the primary bottleneck. It is an iterative process of defining quantitative parameters that define the acceptable behavior of the system, benchmarking, identifying of bottlenecks, modifying, and

benchmarking again. Performance measuring is a process of developing measurable indicators that can be systematically tracked for supervising advances in service level goal fulfillment. With WAs, there is usually a discrepancy between the required or expected, and real performance results. Performance levels can address quality, quantity, time or service price. There are two basic approaches to performance measuring: benchmarking and profiling. While benchmarking answers the question how well the application works, profiling answers the question why does the application have such performance. In this research we have used both approaches to measure the overall performance, as well as to profile the response time structure of individual requests. There are dozens of parameters that can be measured while benchmarking a production or a staging version of an application, which are selected based on identified bottleneck or performance enhancement goals. It is usually not possible to test the application in production environment, so we use test versions, or test part(s) of a cluster, and use an artificial workload. Workload modeling is a very important part of the performance benchmarking process. To achieve the correct measurements, the artificial workload must mirror the sample of the real workload, i.e., live users of the application. This can be achieved using different data sources: real time user observations, planned interactions or log file analysis. Test workload implementation must be as realistic as possible; otherwise, incorrect data may be collected and wrong decisions could be made.

Peformance Enhancement Techniques. Over the years of Web development and research, many different performance enhancement techniques (some more, some less complex to implement) have been proposed and used in practice. As the nature and complexity of the content offered online have changed [23], so have the techniques for performance enhancement. During our preliminary research of this subject we have identified many different techniques aimed at enhancing the performance of multi-tier web application architecture. Some of them are now obsolete; some are effective only with highly distributed systems; others are effective only at most extreme workloads where even 1kB makes a difference; etc. In our research we have analyzed those techniques that are currently most widely used and can be implemented on most WAs in use today - 19 techniques in total (as shown in Table 1). These techniques have been analyzed in the following manner: we defined the objective of each technique and the way in which it tries to achieve its objective, described the implementation process and gave an estimate of duration. Selected techniques have been sorted by the tier of WA architecture that they affect. As expected, the lowest number of techniques affect the business logic tier of application architecture. Although the business logic of an application can cause the bottleneck effect, it is by far the most expensive and complex problem to resolve which cannot be done by any enhancement technique. If the business logic of the application appears to be the causing the bottleneck, that is an indication of bad architecture decisions or implementation. When functioning normally, code execution takes from 5-15% of total response

Table 1. List of analyzed techniques

No code changes required	Code changes required
T1 Caching objects using HTTP expires	T2 Reducing number of HTTP requests
Compressing textual content	Positioning of HTML content objects
JavaScript minification and obfuscation	Reducing number of DNS requests
Reducing size of graphical objects	Managing redirects
RDBMS conguration tuning	Delayed content download
T3 Caching of interpreted PHP scripts	Enhancing performance of core logic
Standalone PHP application server	T4 Caching query results (Memcache)
Choice of web server	Query and database optimization
T5 Web server conguration tuning	Using stored procedures
T6 Caching pages using a proxy server	

(round trip) time and as such it is not a particularly fruitful subsystem for performance enhancement. According to [26], most of the response time is spent on content delivery (up to 80%) and query execution (database lag), so most of the analyzed techniques try to enhance the performance of these subsystems.

All the techniques displayed in Table 1 have been analyzed in detail. Due to the limited scope of this paper, we omit a detailed analysis which can be found in [13]. The analysis consists of 6 properties: 1) Tier - of the WA architecture affected by the technique, 2) Idea - basic explanation of how the technique increases performance, 3) Implementation - how is the technique implemented, 4) Time - how long it took us (on average if not stated differently) to implement this technique in the three applications used in this research. Implementation time for these techniques will vary with respect to the size and complexity of the application (e.g. amount of code, number of servers), 5) Expected results - performance characteristics the technique affects and 6) Verification - how is the performance gain measured. For the experimental part of our research we have selected six techniques with various degrees of complexity and implementation time. Those techniques have been selected based on the assumption that they will have the highest impact on the overall performance of WAs that have been used in this research. A list of analyzed techniques, as well as those that have been selected for experimental testing, can be seen in Table 1. Techniques selected for the experimental part of our research are labeled with T1-T6.

4 Performance Testing and Analysis

In the experimental part of our research we used three open-source WAs of different types/categories: portal (Joomla), community (PhpBB), and e-commerce (OsCommerce) to implement and test the effectiveness of performance enhancement techniques.

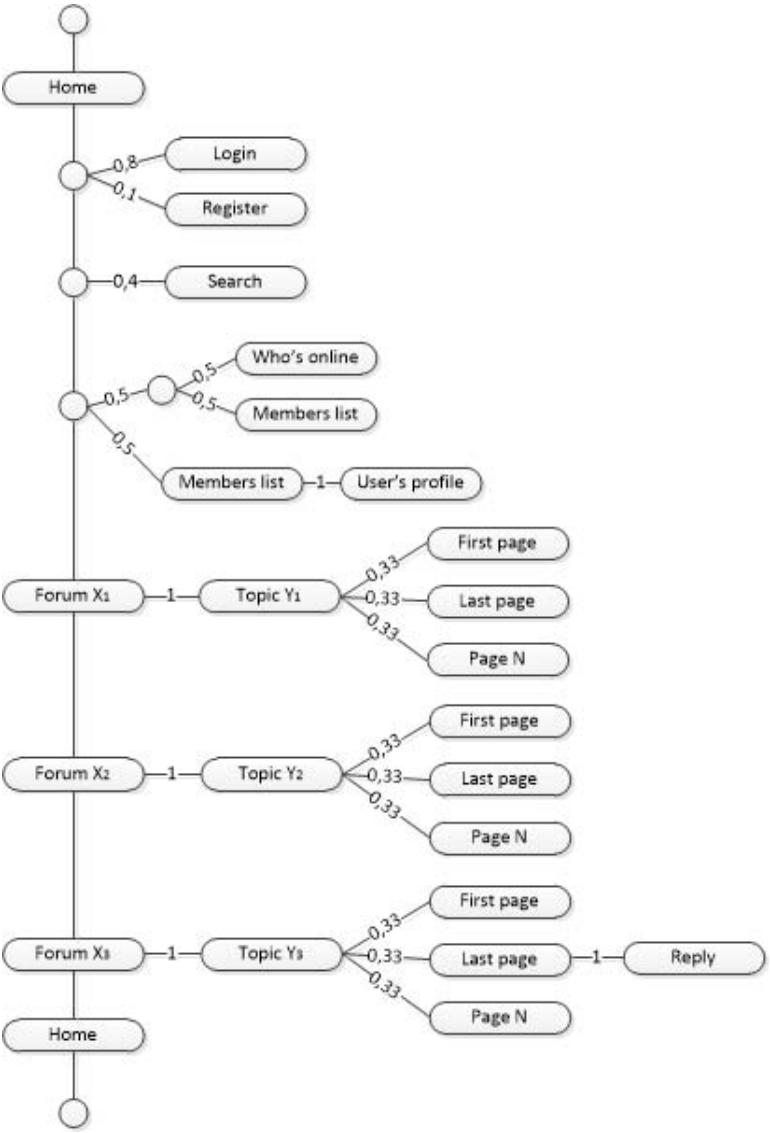

Fig. 2. Workload model for APP2 (Forum)

4.1 Preparation Phase

The first step of the research was to develop a realistic workload model (for all three applications) that will be used in all performance tests. Test workloads were modeled using various approaches: server log analysis (requires shell scripting to extract usage patterns), planned interactions (set by probability or

importance, e.g. product browsing in an e-commerce application has the highest probability) and real user auditing. More about these approaches can be found in [17]. Figure 2 displays the workload model developed for load testing APP2. All actions (HTTP requests) on the baseline will be executed each time the test is performed, while others have a probability indicated on the connecting lines. The test can be described in the following way: all users start at the home page, most of them log on to the system, while some register and some (10%) will remain as guests. About 40% of users will use the search option. Visitors then look at one of the three popular pages (members list, who's online and a users profile). Then they look at three forums (two of them are most popular while the third one is randomly selected) and look at one topic inside those forums. When accessing the topic page some users start at the first post, others go directly to the last page and some select one of the pages of the topic. A third of the visitors will post a message to the forum. Finally, they will go back to the home page (check that there are no more new posts) and leave the page. Workload models for other applications were developed in a similar fashion (ie. most popular articles on the portal, featured products in the webshop). To implement workload models and perform load tests the JMeter [3] tool was used. With complex test models we simulated many requests from the individual user which were randomized by timers (simulating pauses for reading, thinking and form submissions) and random branching (between sets of actions). To ensure randomization of URL requests, test data sources (TXT files) were prepared from the real WA databases. These TXT files are constructed by selecting a subset of primary key identifiers from the real application database and written in a tab delimited format. JMeter then reads one line from the TXT file for test run and adds identifiers to the generated HTTP requests (links). In this way we can simulate the behavior of real users (reading about products, commenting

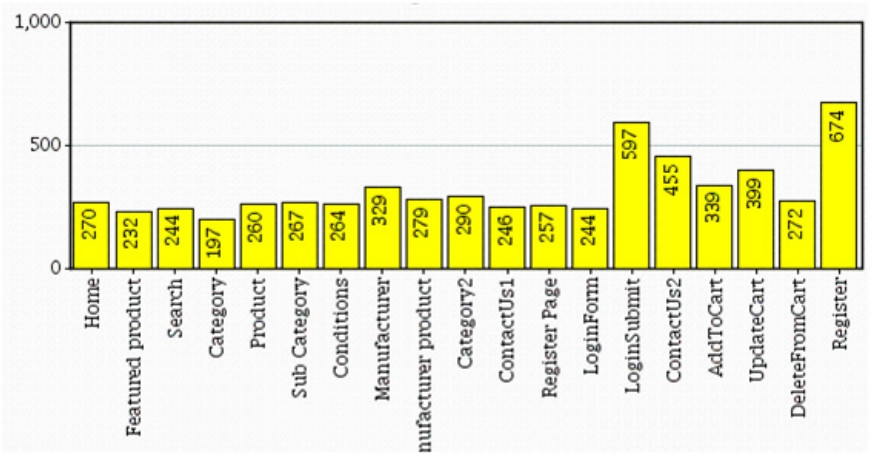

Fig. 3. JMeter aggregate graph

products, adding products to cart, making purchases, etc.). After the workload models were designed and implemented we tested them and verified that they executed correctly (reading identifiers from TXT files, adding them to HTTP requests, all response codes 200, database inserts correct, etc.). After the workload models were developed and implemented as load tests in JMeter we had to find a way to log data about hardware utilization during the tests. We decided to use Linux command line tool **vmstat** which proved to provide enough data on a fine enough time scale with a small enough footprint on server CPU usage. The next step was preparing data visualizations for the analysis phase. The first visualization was done by JMeter. After performing a test, JMeter displays data about each performed request, such as response time and server response, and calculates throughput and average response times. JMeter then offers to generate various visualizations based on this data. In relation to our first measuring constraint (90% of all requests under 2 seconds) we chose to generate an aggregate graph (shown in Figure 3) that displays average response times for each HTTP request (link) defined in the workload model. We have also used **Httpload** [12] and **ab** [4] for fast tests and to verify the results recorded by JMeter. The second visualization was done after the collected hardware utilization data was processed using an **awk** script (to remove unnecessary columns and calculate column averages) and handed over to a **gnuplot** script we developed for this research. Figure 4 demonstrates the output of our gnuplot script. Duration of the test is shown on the X axis in each chart. This visualization

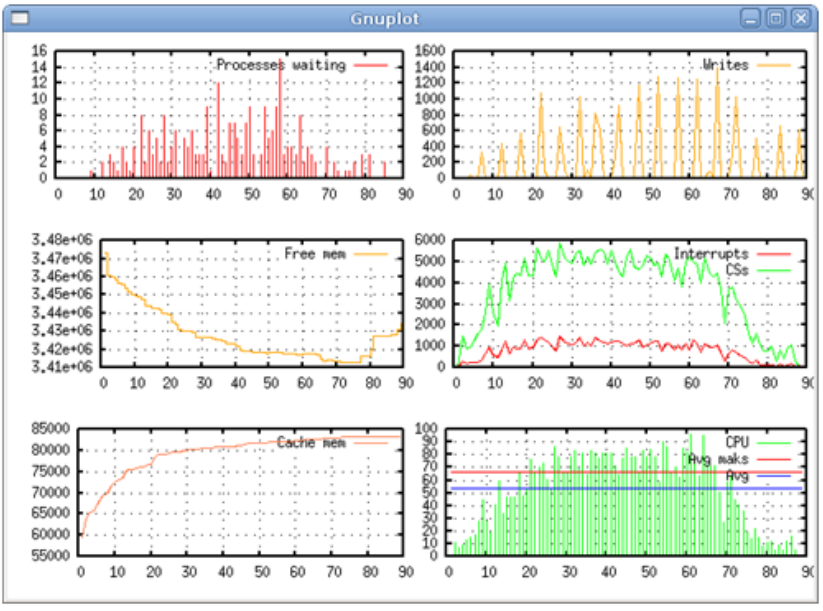

Fig. 4. Hardware utilization visualization (vmstat >awk >gnuplot)

consists of six charts representing various hardware utilization metrics received from vmstat. Some charts contain two data series. The top most graph on the left hand side displays the number of processes waiting during the test (lower number of processes waiting means lower response time), middle-left displays the amount of free memory, and bottom-left the amount of memory used for cache (more data in the cache means lower response time and higher throughput). The top right hand side graph displays the number of writes on the disc (the more files are cached, the less time is spent fetching data from the hard drive, resulting in lower response time, higher throughput and lower CPU usage). The middle-right graph contains two data sets: the number of interrupts and the number of context switches (the higher these values are, the higher CPU usage is, which results in higher response time and lower throughput). The bottom-right graph displays CPU usage during the test and 2 lines: one is the 70% limit set as one of the two main constraints, and the other is the calculated average of CPU usage during the test. The calculated average for the duration of the test had to be less than or equal to 70%. We increased the load on the test server by increasing the number of simulated users in JMeter test plans, until we came as close as possible to one or both limiting factors. Then we performed the three official tests (following the procedure described the following section). All tests were performed in an isolated LAN consisting of a load generator PC, a 100Mbit passive switch and the testbed PC. The testbed PC had the following hardware configuration: Asus IntelG43 Chipset, CPU: Intel Core2Duo 2.5Ghz, 1MB Cache, RAM 4GB DDR2, HDD WD 250GB SATA, OS Debian Linux. Network bandwidth was not a bottleneck in any of the preparation tests so we did not measure network usage.

4.2 Testing Phase

Before starting load tests we analyzed each application (file structure, number of files included per request, number of graphical objects, measured individual request response time). An important measure was the number of SQL queries performed per request. While the first two applications performed 28 and 27 queries, the third one did 83, which was a probable bottleneck. The testing phase consisted of 3 stages. First, we measured the initial performance of all three applications (after installation, using a test data set). Second, we implemented one of the selected performance enhancement techniques and repeated the measurement to determine the effect of the technique on performance (we performed this test 3 times). Then the application was restored to its initial state. This process was repeated for each technique. Third, we implemented all of the selected techniques to all three WAs and made the final performance measurement to determine final performance enhancement achieved using these techniques. In total we performed over 200 load tests. When doing performance testing, ensuring equal conditions in order to achieve the correctness of the acquired data is obligatory and was integrated in our testing procedure:

```
For APP1, APP2, APP3
  For Technique T1 - Tn
    Implement technique Tn
    Run multiple tests until as close as possible to two
    limiting factors // increment number of simulated users
    Reset()
    Restart server (clear memory, cache, etc.)
    Run a warm-up test (prime caches)
    Repeat 3 times
      Run test and gather data (JMeter, vmstat)
      Reset()
    Remove implementation of technique Tn

Reset()
  Reset database to initial state (remove inserts from the
  previous test)
  Reset server logs
```

We have performed each test three times for each implemented technique in order to get a more robust measurement and reduce the possibility of errors. Average values have been calculated from these measurements and are displayed in Table 2. Before starting our measurements, we set the throughput as the key performance indicator. Our goal was to enhance it using various performance enhancement techniques. **The limiting factors were: a) average response time for the 90% of requests had to be less than 2 seconds and b) average CPU utilization had to be kept below 70%**. First, we present the results obtained from the initial testing, then the results for each technique, and, lastly, the final testing results. These results display only the values of the aforementioned key performance characteristics. During each benchmark we collected data for 10 performance indicators which will be taken into consideration later. Second, we display the overall results that confirm our hypotheses. Third, we take into consideration all the data obtained from benchmarking and add weights to each performance indicator in order to define the effectiveness of each technique.

4.3 Results

The summary of our measurements is presented in Table 2. Let us first explain the structure of the Table. The markings in the first column are: B (for Beginning) - this row displays data about the initial performance of un-altered applications (after installation); T1-T6 (for Technique) - these rows display data about performance after each of the techniques was implemented individually; F (for Final) displays data about the performance of applications after all six techniques were implemented and finally C (for Comparison) displays data about the performance of applications under the same workload used in the B row. For each application there are three columns: The Response time (in miliseconds) is given

Table 2. Overall load testing results

-	APP1 Portal			-	APP2 Forum		-	APP3 Webshop	
	RT (ms)	T (R/s)	U (%)	RT (ms)	T (R/s)	U (%)	RT (ms)	T (R/s)	U (%)
B	834	8,3	69	791	15,2	66	794	24	66
T1	1025	8,3	68	661	**16,8**	67	794	24	66
T2	816	8,4	67	789	16,4	67	857	**27**	67
T3	793	**14,4**	68	815	**32,3**	69	1218	**35**	67
T4	858	**13,5**	68	958	**16,8**	69	993	25	67
T5	823	8,6	67	527	15,8	66	726	25	69
T6	958	8,4	69	954	17	70	1670	25	66
F	614	24,9	68	671	38,5	68	1740	35	64
C	91	8,5	21	58	16	23	442	24,1	48

LEGEND: RT= Response time, T = Throughput, U = Utilization, ms = miliseconds, R/s = Requests per second, CPU = average percent of CPU utilisation (measured by vmstat).

in the first column, the Throughput (in requests/second) in the second, and the CPU utilization (average, in %) in the third column. Although some techniques appear to increase the average response time and do not increase the overall throughput, they have a positive effect on other performance indicators that were measured and will be discussed later on. Furthermore, the first technique reduces individual response time which is not visible here, but can be observed in individual response profiling using YSlow (a Firefox profiling add-on). It can be seen that PHP script caching using APC (T3) clearly has the biggest impact on the performance of all tested applications. Storing SQL query data in memory cache using Memcache (T4) follows closely being the second most effective with two of three applications. We can also see that different techniques have different effects on each application. The comparison row (C) demonstrates the performance of applications under the same beginning workload, which was the highest possible before the implementation of all performance enhancement techniques. This data shows how the application would perform under normal daily workload, after its performance has been enhanced using the 6 mentioned techniques. In this row we can see that the throughput value has decreased significantly. This is caused by the fact that JMeter uses the number of simulated users/second (which is one of the arguments used when starting the load test) to calculate throughput. To make the comparison we had to use the load that was used to make the initial performance measurement for the WAs in their original (un-enhanced) state. Therefore, the number of simulated users/second was much smaller than it was for the final versions of WAs with all 6 techniques implemented. The result of final load testing also demonstrates that combining all of the techniques has a cumulative effect, because the maximum throughput for APP1 and APP2 is higher than the contribution of any individual technique. This was expected based on the effect of performance enhancement techniques

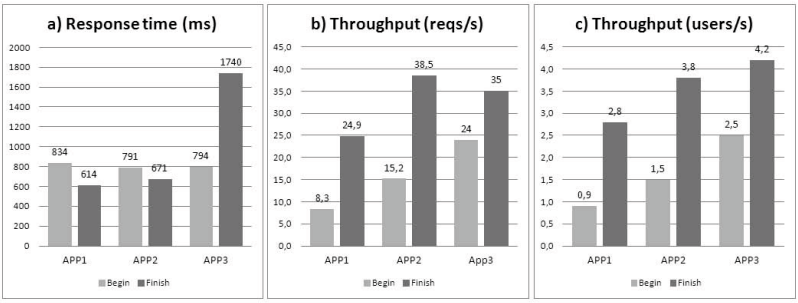

Fig. 5. Final performance enhancement results

on other performance indicators we observed (which will be described in the following section). This was not the case with APP3, whose maximum throughput was limited by database lag (due to the large number of database queries per request (an architectural problem)). To visualize the performance improvements we chose three key indicators - response time, throughput measured by number of requests/second (displayed in Column "T" in Table 2) and throughput measured by the number of users/second (not displayed in Table 2, but measured along with other indicators listed in Table 3). The differences between the values of these indicators in the first(B) and final (F) measurements are given in Figure 5 (a), (b) and c) respectively. The average response time for APP1 and APP2 was reduced even with a much higher workload while the throughput was increased by almost three times. For APP3, our hypothesis was confirmed (we achieved a 30% increase in throughput) but due to the very large number of database queries performed for each request, the increase in throughput was almost 2 times smaller than with APP1 and APP2. The increase in response time is caused by the same problem but within the limit of 2 seconds average for 90% of requests.

5 Calculating Technique Effectiveness

Although the results obtained show that it is possible to significantly enhance the performance of various types of WAs by using just a small subset of performance enhancement techniques, we were interested in defining an overall "quality indicator" of used techniques for each application type, and checking whether they appear in the same order with all WAs. This would mean that there is a uniform top list of techniques that can be implemented on any web application. To precisely determine the efficiency and effectiveness of a technique, we took all the recorded indicators into consideration. The full list of indicators whose values have been recorded during testing is displayed in Table 3. The most important indicator (after the three previously mentioned) is the time needed to implement the technique, which we recorded for each technique. Each indicator is given a

Table 3. List of indicators with appointed weights

No	Name	Weight	Proportional	Acquired from
1	Response time	5	Inversely	JMeter
2	Throughput (requests/sec)	5	Directly	JMeter
3	Throughput (users/sec)	5	Directly	JMeter
4	Utilization (cpu)	2	Inversely	vmstat
5	Processes waiting	4	Inversely	vmstat
6	RAM usage	2	Directly	vmstat
7	Cache mem. usage	3	Directly	vmstat
8	Disc usage (writes)	3	Inversely	vmstat
9	Context switches	4	Inversely	vmstat
10	Implementation time	5	Inversely	Measured / Estimated

weight (range 1-5), marking its importance in the overall performance gains. The weights were given according to our perception of each indicators importance in the overall performance enhancement. The column titled "Proportional" indicates whether the measured indicator is directly or inversely proportional to the technique effectiveness. Directly proportional means the higher the measured value, the better, while inversely proportional means the lower the value, the better, e.g. an increase of requests per second is directly proportional while the implementation time is inversely proportional. We measured the time needed for one developer to implement the technique (change server configuration, change application configuration, change code of the application) in each application. To determine the "quality indicator" of a technique we used the following procedure and equations.

```
For each WA (APP1, APP2, APP3)
 For each technique (T1-T6)
    For each of 10 performance indicators
       Calculate the effect E of indicator n using the value of
  indicator Vn in relation to the indicators maximum value change
 (Cnmax) and minimum indicators value change (Cnmin). Depending on
  whether the indicator is inversely proportional or directly
  proportional formulas (1) or (2) are used (respectively)
```

$$E_n = C_{n_{max}} - V_n / C_{n_{max}} - C_{n_{min}} \, . \tag{1}$$

$$E_n = V_n - C_{n_{min}} / C_{n_{max}} - C_{n_{min}} \, . \tag{2}$$

```
Calculate indicator weight using (3)
```

$$W_i = W_n / W_{max} \, . \tag{3}$$

```
Sum up to get the technique efficiency using (4)
```

$$\sum T_i = \sum_{i=1}^{10} W_i * E_i \ . \tag{4}$$

In this way, we calculated the top list of performance enhancement techniques with respect to type/category of the WA. The results are displayed in Table 4. Technique 1 (Reducing number of HTTP requests) was not implemented on APP 3 (marked "N/A" in Table 4.) because there were not enough graphical objects that could have been merged into a single larger one. It is clear that the order (ranking) of techniques is different for each type of application. A few important conclusions can be made from these calculations and will be used as problem guidelines in our future work. First, we don't have a framework that defines which techniques to use for each type of WA. It is clear that the subset of techniques to be used for performance enhancement must be tailored to the specific application. Secondly, there are a number of factors that influence the decisions about the techniques to be used such as: the goals of performance enhancement (what aspect of performance are we trying to enhance), the type of content the WA delivers (e.g. text, graphic objects, large files, video, etc.) and the specific workload. In our future work we will repeat these measurements on a larger number of (various types of) WAs and try to develop and verify such a framework for identifying a subset of techniques that yields the best results based on these factors.

Table 4. Overall ranking of performance enhancement techniques effectiveness

No	Technque	APP1	APP2	APP3
1	Reducing number of HTTP requests	5	3	N/A
2	Caching objects using HTTP Expires	3	1	3
3	Caching of interpreted PHP scripts (APC)	2	2	1
4	Caching query results (memcache)	1	6	4
5	Web server conguration tuning	6	5	4
6	Caching objects using proxy server (Squid)	4	6	2

6 Conclusion

Static websites are rapidly being replaced by web applications, and the ever increasing number of Internet users demands more and more functionality while expecting lower and lower response time. Web 2.0 has brought about a paradigm shift which changed the structure of workload, moving it from read-intensive to write-intensive. Therefore, the performance of WAs has become one of the focal points of interest of both scientists and professionals in this field. The goal of performance enhancement has to be set before any of the techniques are implemented or tests performed. This goal depends on the problem perceived in

the performance of an application and can be aimed at any aspect of its performance (e.g. minimizing CPU or memory usage). In this research, our goal was to maximize throughput and lower response time of different finished systems (web applications) on the same hardware basis. Performance measurement itself is a complex process that requires careful monitoring of the real workload, identification of bottlenecks, planning and modelling test workloads, identifying key characteristics, goals, technical knowledge on all elements of the content creation and delivery system, etc. We have proved that it is possible, in a controlled environment at least, to significantly enhance the performance of WAs using just a small set of performance enhancement techniques with a total implementation time ranging from 10 to 50 working hours for applications running on one multiple-role (e.g. web, proxy, application) server. We found that the results of each technique vary from application to application and that further research is needed to develop a generalised framework that would take into consideration all the factors mentioned above (goals, content type, system architecture, etc.) and suggest what techniques would be best suitable for a selected application.

References

1. Aberdeen Group: Application Performance Management, http://www.aberdeen.com/Aberdeen-Library/5807/RA-application-performance-management.aspx
2. Amza, C., Soundararajan, G., Cecchet, E.: Transparent Caching with strong consistency in dynamic content web sites. ICS Boston (2005)
3. Apache JMeter, http://jakarta.apache.org/jmeter
4. Apache Benchmark Tool, http://httpd.apache.org/docs/2.0/programs/ab.html
5. Bahn, H.: Web cache management based on the expected cost of web objects. Information and Software Technology 47, 609–621 (2005)
6. Bogardi-Meszoly, A., Levendovszky, T.: A novel algorithm for performance prediction of web-based software system. Performance Evaluation 68, 45–57 (2011)
7. Domenech, J., Pont, A., Sahuquillo, J., Gil, J.A.: A user-focused evaluation of web prefetching algorithms. Journal of Computer Communications 30, 2213–2224 (2007)
8. Domenech, J., Pont, A., Sahuquillo, J., Gil, J.A.: Web prefetching performance metrics: a survey. Performance Evaluation 63, 988–1004 (2006)
9. Georgakis, H.: User behavior modeling and content based speculative web page prefetching. Data and Knowledge Engineering 59, 770–788 (2006)
10. Henderson, C.: Building Scalable Web Sites. OReilly, Sebastopol (2006)
11. Huang, Y., Hsu, J.: Mining web logs to improve hit ratios of prefetching and caching. Knowledge-Based Systems 21, 149–169 (2008)
12. Http Load Tool, http://www.acme.com/software/httpload/
13. Jugo, I.: Analysis and evaluation of techniques for web application performance enhancement, Master of Science Thesis, in Croatian (2010)
14. Khayari, R.: Design and evaluation of web proxies by leveraging self- similarity of web traffic. Computer Networks 50, 1952–1973 (2006)
15. Lam, K., Ngan, C.: Temporal prefetching of dynamic web pages. Information Systems 31, 149–169 (2006)

16. Liu, H., Keelj, V.: Combined mining of Web server logs and web contents for classifying user navigation patterns and predicting users future requests. Data and Knowledge Engineering 61, 304–330 (2007)
17. Meier, J.D., Farre, C., Banside, P., Barber, S., Rea, D.: Performance Testing Guidance for Web Applications. Microsoft Press, Redmond (2007)
18. Na, Y.J., Leem, C.S., Ko, I.S.: ACASH: an adaptive web caching method based on the heterogenity of web object and reference characteristics. Information Sciences 176, 1695–1711 (2006)
19. Nagpurkar, P., et al.: Workload characterization of selected J2EE-based Web 2.0 applications. In: 4th International Symposium on Workload Characterization, pp. 109–118. IEEE Press, Seattle (2008)
20. Ohara, M., Nagpurkar, P., Ueda, Y., Ishizaki, K.: The Data-centricity of Web 2.0 Workloads and its impact on server performance. In: IEEE International Symposium on Performance Analysis of Systems and Software, pp. 133–142. IEEE Press, Bostin (2009)
21. Pea-Ortiz, R., Sahuquillo, J., Pont, A., Gil, J.A.: Dweb model: Representing Web 2.0 dynamism. Computer Communications 32, 1118–1128 (2009)
22. Ravi, J., Yu, Z., Shi, W.: A survey on dynamic Web content generation and delivery techniques. Network and Computer Applications 32, 943–960 (2009)
23. Sadre, R., Haverkort, B.R.: Changes in the web from 2000 to 2007. In: De Turck, F., Kellerer, W., Kormentzas, G. (eds.) DSOM 2008. LNCS, vol. 5273, pp. 136–148. Springer, Heidelberg (2008)
24. Sajeev, G., Sebastian, M.: Analyzing the Long Range Dependence and Object Popularity in Evaluating the Performance of Web Caching. Information Technology and Web Engineering 4(3), 25–37 (2009)
25. Sivasubramanian, S., Pierre, G., van Steen, M., Alonso, G.: Analysis of Caching and Replication Strategies for Web Applications. Internet Computing 11(1), 60–66 (2007)
26. Souders, S.: High Performance Web Sites. O'Reilly, Sebastopol (2007)