Dependability Benchmarking of Web-Servers

João Durães[1], Marco Vieira[1], Henrique Madeira[2]

[1] ISEC/CISUC - Polytechnic Institute of Coimbra
3030 Coimbra, Portugal
{jduraes, mvieira}@isec.pt
[2] DEI/CISUC - University of Coimbra
3030 Coimbra, Portugal
henrique@dei.uc.pt

Abstract. The assessment of the dependability properties of a system (dependability benchmarking) is a critical step when choosing among similar components/products. This paper presents a proposal for the benchmarking of the dependability properties of web-servers. Our benchmark is composed of the three key components: measures, workload, and faultload. We use the SPECWeb99 benchmark as starting point, adopting the workload and performance measures from this performance benchmark, and we added the faultload and new measures related to dependability. We illustrate the use of the proposed benchmark through a case-study involving two widely used web servers (Apache and Abyss) running on top of three different operating systems. The faultloads used encompass software faults, hardware faults and network faults. We show that by using the proposed dependability benchmark it is possible to observe clear differences regarding dependability properties of the web-servers.

1 Introduction

The term *dependability* translates to the quality by which a system can be relied on. This includes attributes such as availability, reliability, safety, integrity, etc. The relative importance of each attribute is dependent on the nature of the system considered. *Dependability benchmarking* is a process by which the dependability attributes of a system are assessed. The evaluation of the dependability properties of a computer system or component is a critical step when choosing among similar products. However, assessing system dependability is a very difficult problem as it is dependent on the fault probability, which in turn is dependent on many factors, either internal to the system (hardware and software) or external (environment or human made).

Dependability assessment has been addressed by using both model-based and measurement-based techniques. The former include analytical [1] and simulation [2] techniques. Measurement-based techniques include field measurement [3], fault injection [4] and robustness testing [5], just to mention a few examples.

Most of these dependability assessment techniques have been developed for mission-critical systems or for the business-critical area, and thus make assumptions about design or operating environment that affect their direct porting to more main-

stream computing systems. The goal of dependability benchmarking is thus to provide generic ways of characterizing the behavior of components and computer systems in the presence of faults, allowing for the quantification of dependability measures.

Following the well-established philosophy used in the performance benchmark world, the dependability benchmarking proposals are mainly inspired on measurement-based techniques. However, beyond existing techniques, such as fault injection and robustness testing, dependability benchmarking must provide a uniform, repeatable and cost-effective way of performing this evaluation, especially for comparative evaluation of competing or alternative systems and/or components.

A dependability benchmark can then be defined as a specification of a standard procedure to assess dependability related measures of a computer system or computer component. The main components of the dependability benchmark are [6]:

- **Workload**: represents the work the system must do during the benchmark run.
- **Faultload**: represents a set of faults and stressful conditions that emulate real faults experienced in the field.
- **Measures**: characterize the performance and dependability of the system under benchmark in the presence of the faultload when executing the workload.
- **Experimental setup and benchmark procedure**: describes the setup required to run the benchmark and the set of procedures and rules that must be followed during the benchmark execution.

This paper proposes a dependability benchmark for web-servers (the WEB-DB) with emphasis on the reliability, availability (readiness and service continuity) and integrity (non-occurrence of invalid information) attributes. To our knowledge, this is the first proposal of a dependability benchmark for this important class of systems.

The paper is organized as follows. Next section describes the related work. Section 3 presents the Web-DB dependability benchmark. Section 4 describes an example of the use of Web-DB to benchmark dependability aspects of the Apache and Abyss web servers. Section 5 concludes the paper.

2 Related work

The idea of dependability benchmarking has become popular the last few years and is currently the subject of intense research, having already led to the proposal of several dependability benchmarks for several different application domains.

In [7] it is proposed a dependability benchmark for transactional systems - the DBench-OLTP dependability benchmark. This benchmark specifies the measures and all the steps required to evaluate both the performance and key dependability features of OLTP systems. A dependability benchmark for transactional systems considering a faultload based on hardware faults is proposed by [8].

A dependability benchmark for operating systems is proposed by [9]. The goal of this benchmark is to characterize qualitatively and quantitatively the OS behavior in the presence of faults and to evaluate performance-related measures in the presence of faults. The results of a research work on the practical characterization of operating

systems (OS) behavior in the presence of software faults in OS components, such as faulty device drivers, is presented in [10].

Research work developed at Berkeley University has lead to the recent proposal of a dependability benchmark to assess human-assisted recovery processes [11].

The work carried out in the context of the Special Interest Group on Dependability Benchmarking (SIGDeB), created by the IFIP WG 10.4, has resulted in a set of standardized availability classes to benchmark database and transactional servers [12].

Recent work at Sun Microsystems defined a high-level framework [13] dedicated specifically to availability benchmarking. Within this framework, two specific benchmarks have been developed. One of them [14] addresses specific aspects of a system's robustness on handling maintenance events such as the replacement of a failed hardware component or the installation of software patch. The other benchmark is related to system recovery [15].

At IBM, the Autonomic Computing initiative (see http://www.ibm.com/autonomic) is also developing benchmarks to quantify a system's level of autonomic capability, addressing four main spaces of IBM's self-management: self-configuration, self-healing, self-optimization, and self-protection [16].

3 Measuring dependability of Web-Servers: a benchmark proposal

A typical web environment consists on several clients performing their commands via a web-browser connected to the web-server through the Internet. In a simplified approach, the server is composed by three main components: the hardware platform, the operating system, and the web-server.

The WEB-DB dependability benchmark proposed in this paper uses the basic experimental setup, the workload, and the performance measures specified in the SPECWeb99 benchmark [17]. The following subsections present the WEB-DB dependability benchmark, with particular emphasis on the new components.

3.1 Experimental setup and benchmark procedure

Figure 1 presents the key elements of the experimental setup required to run the WEB-DB dependability benchmark. The key elements are the **System Under Benchmarking (SUB)** and the **Benchmark Management System (BMS)**.

The SUB consists on a web-server installation, including all hardware and software necessary to run the web-server. From the benchmark point-of-view, the SUB is the set of processing units needed to run the workload. It is important to note that the SUB is larger than (and includes) the component directly targeted by the benchmark (named **Benchmark Target - BT**), which is the web-server.

The BMS is a set of tools that control all aspects of the benchmark experiments. Its key functionalities are: submission of the workload, coordination and synchronization of the several components involved in the experiments, and collection of the raw data

needed to produce the benchmark measures (measures are computed afterwards by analyzing the information collected during the benchmark run).

The execution of the WEB-DB benchmark includes two main phases:

– **Phase 1**: this phase corresponds to the execution of the SPECWeb99 performance benchmark (see [17]), and is used to determine the *baseline performance* of the SUB. The baseline performance corresponds to the performance attainable by the SUB in normal operation conditions (that is, without artificial faults) but with the BMS tools running. A key notion implied here is that the BMS is considered as part of the workload submitted to the SUB. A second result of this phase is a measure of the intrusiveness of the BMS which is directly given by the difference of the performance of the SUB with and without the BMS. Note that, the execution of the SPECWeb99 benchmark includes 3 runs and the results reported are the average of the results from those runs.

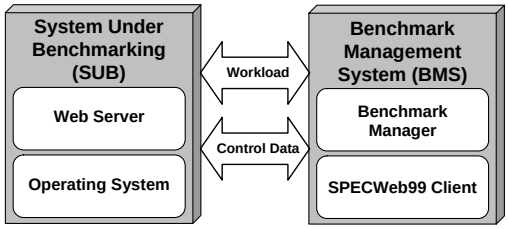

Fig. 1. Experimental setup overview

– **Phase 2**: in this phase the workload is run in the presence of the faultload to measure the impact of faults on the SUB to evaluate specific aspects of the target system dependability. As in phase 1, this phase includes 3 runs and the results reported represent the average of the results from those runs. During each run, all faults defined in the faultload are applied.

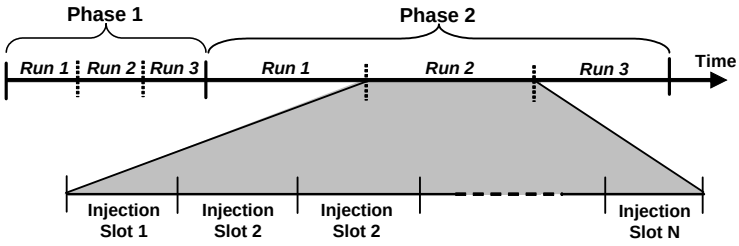

Fig. 2. Benchmark execution profile

As shown in Figure 2, each run is made of several injection slots. An injection slot can be defined as a measurement interval during which the workload is run and some faults from the faultload are injected. The execution profile of each injection slot is closely related to the class of the faults to be injected.

3.2 Workload

The WEB-DB dependability benchmark adopts the workload of the well-established SPECWeb99 performance benchmark [17]. This workload represents typical requests submitted to real web-servers. Its definition was based on the analysis of web-based services in several real web-sites (see [17] for more details). The workload is composed of the typical operations allowed by the HTML (GET and POST operations, both static and dynamic). The workload also reproduces common actions such as on-line registration and advertisement serving.

3.3 Measures

The WEB-DB dependability benchmark measures are computed from the information collected during the benchmark run and follow the well-established measuring philosophy used in the performance benchmark world. In fact, the measures provided by existing performance benchmarks give relative measures of performance that can be used for system comparison or for system/component improvement and tuning. It is well known that performance benchmark results do not represent an absolute measure of performance and cannot be used for planning capacity or to predict the actual performance of the system in field. In a similar way, the measures proposed for this first dependability benchmark must be understood as benchmark results that can be useful to characterize system dependability in a relative fashion (e.g., to compare two alternative systems) or to improve/tune the system dependability.

The WEB-DB measures are grouped into three categories: baseline performance measures, performance measures in presence of faults, and dependability measures.

The **baseline performance measures**, inherited from the SPECWeb99 performance benchmark, are obtained during Phase 1 and include:

- **SPEC**: this is the main SPECWeb99 metric. It measures the number of simultaneous conforming connections. SPEC defines conforming connection as a connection with an average bit rate of at least 320 kbps and less than 1% of errors.
- **THR**: reports the number of operations per second (throughput).
- **RTM**: represents the average time in milliseconds that the operations requested by the client take to complete (medium response time).

The **performance measures in the presence of faults**, which represent the penalty in the performance of the web-server caused by the faults injected in Phase 2, include:

- **SPECf**: main SPEC measure in the presence of the faultload.
- **THRf**: throughput in the presence of the faultload.
- **RTMf**: response time in the presence of the faultload.

The **dependability measures** reported are also collected in Phase 2 and include:

- **Autonomy**: this metric gives an idea about the need of external administrative intervention to repair the web-server. Administration intervention is needed when

the web-server dies or stops providing useful service. This measure is computed as: Autonomy = (100 – (No. administrative intervention / No. faults)*100).

- **Accuracy**: reports the error rate in the presence of faults. This measure is computed as: Accuracy = 100 – (No. request with errors / No. requests)*100. This measure relates to the dependability attribute of integrity.
- **Availability**: represents the time the system is available to execute the workload. It is worth noting that in the context of the WEB-DB dependability benchmark, availability is defined based on the service provided by the system. This way, the system is considered available when it is able to provide the service defined by the workload. In other words, the system is not available if the clients get no answer or get an error. For each run in Phase 2, this measure is given as a ratio between the amount of time the system is available and the total duration of that run.

3.4 Faultload

The faultload represents a set of faults and exceptional events that emulate real faults experienced by web-servers in the field. A faultload can be based on three major classes of faults: operator faults, software faults, and hardware faults.

The WEB-DB benchmark use two different faultloads: one based on software faults that emulate realistic software defects (see [18, 19]) and another based on operational faults that emulate the effects of hardware and operator faults. Of course, a general faultload that combines these two is also possible (and is in fact the best option). The following subsections present and discuss the two faultloads.

3.4.1 Faultload based on software faults

A representative faultload based on software faults is one that contains only faults that are representative of real program errors that elude traditional software testing techniques and are left undiscovered in software products after shipment. The results presented in [20, 18] identify a clear trend in the software faults that usually exist in available systems: a small set of well-defined fault types is responsible for a large part of the total software faults. Using this set of fault types as a starting point for a faultload definition, in [19] a generic faultload based on software faults for dependability benchmarking is proposed. WEB-DB follows the guidelines defined in [19] in the definition of the faultload based on software faults. Table 1 presents the fault types that compose this faultload and reproduces the statistical information regarding the representativeness of the fault types according to the field data analyzed in [18].

To emulate the software faults of Table 1, we use an implementation of the G-SWFIT technique [19]. This technique emulates the software faults by reproducing directly at low-level code the processor instruction sequences that represent programming errors. The instruction sequence inserted in the target code is equivalent to the code that would have been generated by a compiler if the emulated defect had been in fact presented in the high-level source code. A library of mutations guides the

entire process. This technique has the important advantage that does not need source code of the target, and provides a good accuracy in the emulation of faults.

One important aspect is that the software faults must not be injected in the benchmark target (the web-server). As the injection of software faults implies the modification of the target code, any conclusion drawn afterwards might not apply to the original BT. We use the notion of Fault Injection Target (FIT) which is a component of the SUB other than the BT. The best option for FIT in the WEB-DB context is the operating system itself, as the OS is indispensable in a web-server installation, and its services are required for the execution of the workload. Because the OS is very large, specific portions of it were previously selected as prime candidates for fault injection. These portions correspond to the code of the API most used by typical web-servers when executing the SPECWeb99 workload (mostly network and file management API). The resulting faultloads are specific to a given OS and not of the web-server.

Table 1. Types of software faults considered in the faultload. The fault coverage is based on the field study presented in [18], which also agrees in general with another sudy from IBM presented in [20].

Fault Types	Fault Coverage
Missing variable initialization	2.25 %
Missing variable assignment using a value	2.25 %
Missing variable assignment using an expression	3.00 %
Missing "if (cond)" surrounding statement(s)	4.32 %
Missing "AND EXPR" in expression used as branch condition	7.89 %
Missing function call	8.64 %
Missing "If (cond) { statement(s) }"	9.96 %
Missing small and localized part of the algorithm	3.19 %
Wrong value assigned to a value	2.44 %
Wrong logical expression used as branch condition	3.00 %
Wrong arithmetic expression used in parameter of function call	2.25 %
Wrong variable used in parameter of function call	1.50 %
Total	**50.69%**

Because the implementation of a G-SWFIT injector is a complex task, the tools needed to inject software faults and the fault library for the different operating systems are provided with the benchmark specification. The idea is to simply download the tool and use it in the benchmark environment.

3.4.2 Faultload based on operational faults

The faultload based on operational faults comprises hardware and operator faults. The types of faults considered have been chosen based on a estimation of the rate of occurrence, ability to emulate the effects of other types of faults, diversity of impact in the system, and portability. The faultload is composed by a set of faults from the types presented in Table 2 injected at different (and well-defined) instants.

The injection of the types of faults considered in this faultload is quite easy when compared to the injection of software faults. Simple applications can be developed to injected the hardware and operator fauls considered. The most right column of Table 2 presents some examples on how to introduce each type of fault in the SUB.

Table 2. Types of environment faults considered in the faultload.

Fault Types	Fault Injection
Network interface failure	Represents a failure in the server network card. This fault can be easily emulated by disabling the network interface at the operating system level.
Network connection failure	Represents a failure in a network connection and can be emulated by closing abruptly TCP sockets used to connect the web-server to the clients.
Abrupt web-server shutdown	Represents an operator/hardware fault that leads to the abrupt shutdown of the web-server. This fault can be emulated by killing the web-server processes at the operating system level.
Abrupt server reboot	Represents an operator fault that leads to the abrupt reboot of the server. This fault can be emulated by rebooting the operating system.

3.4.3 Injection slots execution profile

After choosing the types of faults to inject and the technique to inject those faults, we have to define the profile for the injection of faults. As mentioned before, each execution of the WEB-DB benchmark is made of three runs and each run comprises several injection slots. Figure 3 shows the execution profile for each injection slot.

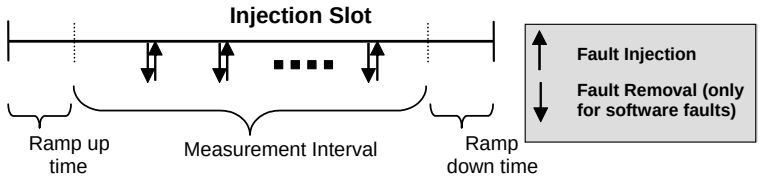

Fig. 3. Injection slot execution profile

In order to assure that each injection slot portraits a realistic scenario as much as possible, and at the same time assure that important properties such result repeatability and representativeness of results are met, the definition of the profile of the injection slot has to follow several rules. The following points summarize those rules:

- The SUB state must be explicitly restored at the beginning of each injection slot and the effects of the faults do not accumulate across different slots.
- The measurement interval starts when the system achieves the maximum processing throughput, which happens after a given time running the workload (ramp-up time). The ramp down time at the end of each injection slot represents the time the SPECWeb99 workload needs to end. Note that, no faults are injected during the SUB ramp-up and ramp-down. Ramp-up and ramp-down are 300 seconds long. These values are imposed by SPECWeb99 (see [17] for details).
- For the faultload based on software faults, faults are injected in intervals of 10 seconds and the measurement interval has a maximum duration of 20 minutes, which means that several slots may be needed to inject all the faults in the faultload, as the number of slots depends on the number of faults in the faultload. The 10 seconds injection intervals have been established based on the SPECWeb99 workload operations, which take less than one second, thus inserting each fault for a period of 10 seconds is enough time to activate the fault. Before injecting the

next fault, the previous one is undone (i.e., the original code is restored). It is worth noting that as the fault target for software faults is the operating system code, different web servers are benchmarked with exactly the same faults, and the benchmark results compare the way the different web servers behave in the presence of an unstable/faulty operating system.

- For the faultload based on operational faults, four injection slots are needed (i.e., one slot for each type of fault). Several faults from a given type are injected in each slot. The duration of the measurement interval is 20 minutes. The time between injections depends on the type of the faults and is the following: 20 seconds for network connection failures, 40 seconds for network card failures, 10 seconds for abrupt web-server shutdowns, and 10 minutes for abrupt server reboots.
- After the injection of a fault an error diagnostic procedure has to be executed to evaluate the effects of the fault and to check if a recovery procedure is required (if an error is detected). This recovery procedure is controlled by the BMS and represents the external intervention needed to restore the service of the web server.

4 Apache vs Abyss: an example of dependability benchmarking

Our case-study is composed of the two web-servers: Apache (www.apache.com) and Abyss (www.aprelium.com/abyssws). Each web-server was installed over three different operating systems: Windows 2000 (SP4), Windows XP (SP1) and Windows 2003 Server. This resulted in a total of six different SUB configurations. It is worth noting that the benchmark proposal is in no way tied to a specific platform. The benchmark specification remains the same across different OSes and web-servers: all that is needed to use the benchmark on other platforms is to port the benchmark tools (we are currently working on a port for the Linux platform).

4.1 Experimental Setup

Each SUB configuration involves two computers connected via a 100 Mbps direct Ethernet connection. The SUB is entirely contained in one of the computers (the *server* computer). The BMS is composed of the following tools:

- The SPECWeb client, responsible for submitting the workload to the SUB and collecting measurements related to performance (placed in the client computer).
- The fault injector, responsible for all fault injection tasks. This tool is running in the server computer. The fault injector is also responsible for the assessment of the web-server process status. If the web-server process dies unexpectedly, or if it hangs or otherwise stops providing service, then it is restated by the fault injector. These events are useful to obtain the autonomy results.
- The availability evaluator, which continuously checks if the web-server is providing service by submitting a “GET” request and analyzing the result.

Fig 4 represents a client-server pair. All the computers used in our experiments have exactly the same configuration (1.6 GHz Pentium IV, 512 Mb RAM, 7200 rpm IDE hard disk, 100Mbit network interface). The software configuration of the client computers is the same for all SUBs (the SPECWeb client and the availability evaluator running on top of Windows XP). The server computers have six different configurations (i.e., six different SUBs: three OSes and two Web servers).

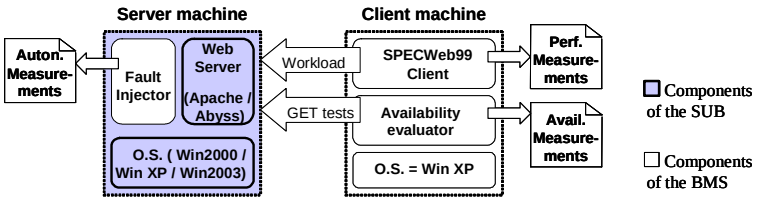

Fig. 4. Experimental setup.

Because some of the BMS tools are also running on the same machine as the web-server, there is a small performance penalty. To maintain our conclusions meaningful, we consider that the execution of the BMS tools is part of the workload subjected to the SUB, i.e., we want to observe the behavior variation induced by the presence of faults and not by presence of the BMS tools. This means that the normal behavior of the SUB must be assessed with the BMS running but with no fault being injected (we call this the *baseline performance*). To that effect, our fault injector has a profile-only mode that executes all the tasks related to fault injection except the injection itself. Later, the behavior of the web-servers in presence of faults will be compared to the baseline performance. The workload execution with and without the BMS tools have shown a performance intrusion less than 4%. We obtained the maximum performance attainable on our hardware configuration according to two guidelines:

- There was no specific effort towards custom performance optimization. Two factors contribute for *less than maximum* performance of our setups, when comparing to published results for similar hardware setups: the network interface has a maximum bandwidth of 100Mbits, and we used interpreted *perl* CGI-scripts instead of compiled scripts.
- The maximum workload subjected to each SUB was such that the *conformance* (as defined by SPECWeb99) is 100%, which means that every connection that was submitted to that SUB was fulfilled in a timely fashion (see SPECWeb99 specifications [17]). We also imposed that no errors are reported. The rationale behind the 100% conformance and 0% errors requirements is that we are interested in observing a 100% clean (no errors) SUB behavior. Any errors that occur during the benchmark experiments can then be assigned to the presence of the faultloads.

4.2 Benchmarking Results and Discussion

A complete benchmark experiment for a given SUB is composed of three complete executions for each faultload, following the SPECWeb99 style. The values for each benchmark measures are the average of the values obtained in each run for each

measure. Table 3 presents results obtained for each faultload and the resulting average. To facilitate the comparison of the behavior of the SUB with and without faults, we also included in table 3 the baseline performance for each SUB (row "baseline").

Before analyzing the results, it is important to recall that the benchmark measures characterize the SUB. However, the SUB is the whole server (platform + operating system + web server), which is larger than the component the Web-DB is meant to characterize (called benchmark target - BT), which is the web server. The comparative analysis of different web servers is achieved by comparing the results obtained in more than one SUB, where the only difference between each SUB is the BT. For example, we can only compare the results obtained for Apache and Abyss when each web-server is running on the same platform and operating system.

Table 3. Benchmark measures results. The Autonomy (AUT%), Availability (AVL%) and Accuracy (ACR%) are given in percentage, the Throughput in the presence of fault (THRf) is given in operation per second, and the Response Time in the presence of fault (RTMf) is given in miliseconds. The SPECf measure has the same units as SPEC.

		Apache						Abyss					
		AUT%	AVL%	SPECf	THRf	RTMf	ACR%	AUT%	AVL%	SPECf	THRf	RTMf	ACR%
2000	(Baseline)	(100)	(100)	(31)	(90)	345,9	(100)	(100)	(100)	(28)	(82.7)	(344.4)	(100)
	Software	92,63	96,72	10,64	83,65	362,18	94,63	90,7	95,61	4,97	75,96	359,69	90,07
	Operation.	95,32	93,84	17	74,83	402,23	99,79	98,02	97,09	15,67	75,96	367,75	99,48
	Average	93,98	95,28	13,82	79,24	382,2	97,21	94,36	96,35	10,32	75,96	363,7	94,78
XP	(Baseline)	(100)	(100)	(26)	(74.5)	(348.9)	(100)	(100)	(100)	(25)	(73.3)	(343.4)	(100)
	Software	93,68	97,84	13,8	71,67	357,12	95,56	93,51	96,62	10,74	68,69	356,68	89,42
	Operation.	97,28	98,04	22,33	71,59	362,33	99,63	98,43	98	16,67	67,74	367,4	99,57
	Average	95,48	97,94	18,07	71,63	359,7	97,6	95,97	97,31	13,71	68,22	362	94,5
2003	(Baseline)	(100)	(100)	(30)	(82.4)	(363.9)	(100)	(100)	(100)	(24)	(70)	(345.8)	(100)
	Software	94,72	97,43	13,79	78,82	371,48	95,5	93,55	96,69	10,4	66,09	355,08	91,49
	Operation.	98,81	97,81	8,75	79,59	374,77	99,07	98,94	98,37	15,42	66,26	362,27	99,6
	Average	96,77	97,62	11,27	79,21	373,1	97,29	96,25	97,53	12,91	66,18	358,7	95,55

Because the benchmark comprises six different measures, it is inevitable that some of the measures favor one of the BT, while the others measures favor others BT. Different benchmark-users may intend to deploy the web-servers in different environments. As such, the relative weight of each benchmark measure may have different relative weights across different benchmark users (e.g., user *A* may consider the availability more important than user *B*). In this case-study we assumed a general-purpose web-server scenario and assigned equal relevance to all six benchmark measures. To facilitate the identification of the SUB with the best dependability results, the values more favorable are presented with a gray background. The results presented in table 3 suggest that Apache presents better dependability properties that Abyss. Indeed, five out of six benchmark measures have their best values on the Apache side. For space reasons we cannot show the results in a form of charts, which would show the differences between Apache and Abyss much more clearly.

The benchmark is primarily intended for web-server comparison. However, accepting the faultloads as representative of real faults, the results can also be used to analyze and compare the properties of the OSes or the entire SUB (OS + Web-server). If we focus on one server, we can obtain information regarding which operat-

ing system provides better dependability properties for that server. According to our results and taking into account both faultloads and all six measures, Windows XP seems to provide the best platform for Apache and Windows 2003 the best for Abyss. The differences between XP and 2003 are smaller than the differences between 2000 and any of the other two OSes. On the other hand, if we focus on a given operating system we can obtain which of the two observed web-servers provides the most dependable behavior. We can also extract information regarding which of the six SUB presents the best dependability properties: the combination Apache/XP seems to be the one where the service degradation caused by faults is less noticeable.

Concerning software faults in the operating system, Apache shows a clear advantage when compared to Abyss. The only measure where Apache does not behaves better than Abyss is the response time in presence of faults (RTMf), in particular when the underlying OS is the 2003. However, all the other measures favor Apache.

When considering the faultload of operational faults, the differences between both servers are not as clear as when using the faultload of software faults. Apache presents a better overall behavior regarding Autonomy, SPECf and THRf, but Abyss is superior when considering Availability, Accuracy and RTMf. The benchmark user may weight each measure according to the operational environment where the web-server is to be deployed in order to better distinguish the behavior of the servers.

When considering both faultloads (global results), again it becomes clear the difference between Apache and Abyss. Although Abyss is better according to Availability and Response Time, Apache is superior regarding all other four measures.

The following remarks are worth noting:

- The faultload based of software faults is the one that causes the larger number of process hangs and aborts leading to lower Autonomy results. This agrees with the idea that correctness of OS is essential to the correct behavior of the applications.
- Abyss and Apache appear to have different internal strategies dealing with network mechanisms. Indeed, faults related to network (network connection failure, network interface failure) cause larger behavior differences than other fault types. The difference in network fault tolerance is also indirectly behind the differences between the Availability results: when Apache dies unexpectedly, its next instance has some trouble restoring the listening sockets; on the other hand, Abyss can restore its sockets more quickly in the same circumstances.

4.3 Benchmarking Execution Effort

The time involved in the execution of our benchmark for a given SUB is the sum of the time required for the setup procedures and the execution itself. The setup procedures are: OS and web-server installation, SPECWeb client, benchmark tools installation, and the faultload generation. The setup and configuration of the OS + web-server + tools took us less than one hour. It is not expected that other OSes and servers take significantly more than it took us to configure our setup. It is important to refer that we are interested in obtaining a basic, working web-server machine, not a customized specific-purpose portal; therefore the configuration procedures are relatively simple.

The only faultload that may require a specific preparation is the faultload based on software faults as it is specific to each OS (see [19]). The generation of this faultload using G-SWFIT [18] is an automated process that requires only simple parameter tuning. The entire process takes less than half an hour, provided that the tools for the generation are available. In our case we have an implementation of the G-SWFIT technique for the windows family which will be available for download on the web.

The execution of the benchmark is mainly dependent on the faultload size. Taking into account the three runs imposed by the SPECWeb99 rules and considering all slots needed to execute the complete faultload, the total benchmark execution time is 31 hours and 45 minutes (the worst-case scenarios are the SUBs involving Windows XP which has a faultload of software faults of nearly 3000 faults) Adding the setup timing, the total time required by the benchmark is a less than one and a half day.

5 Conclusions

In this paper we presented a proposal for the dependability benchmarking of web-servers. Given the central role that web-based services play today, the existence of a dependability benchmark aimed at the characterization of web-servers is a valuable tool when planning a web-based information system. The results of the benchmark are especially useful when comparing several web-servers to decide which one is best suited to include in a larger information system.

The benchmark components were specifically designed to be representative of the typical web-based services: the measurements address the both user and system-administrator perspectives and target all the key properties of the service expected from typical web-servers. The workload is composed of a standard widely accepted performance benchmark which represents the typical requests submitted to web-servers. The faultload addresses typical faults existing in the operational environment of web-servers including software, hardware, operator and network faults. A very important aspect of the benchmark is the fact that it was specifically designed to be independent from internal knowledge about the benchmark targets.

We illustrated the use of the benchmark through a case-study involving two web-servers running on top of three different operating systems. The results show clear differences between the two web-servers and confirm that the benchmark can be used to differentiate the dependability properties of web-servers. This is a valuable help whenever there is the need to choose between several web-servers. Given the current industry trend to used components-off-the shelf to build larger systems, a tool to evaluate the dependability properties of such components is indispensable.

References

[1] K. S. Trivedi, B. Haverkort, A. Rindos and V. Mainkar, “Methods and Tools for Reliability and Performability: Problems and Perspectives”, in Proc. 7th Int. Conf. on Techniques and Tools for Computer Perf. Eval., LNCS, 794, Springer-Verlag, Vienna, Austria, 1994.

[2] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson and J. Karlsson, “Fault Injection into VHDL Models: The MEFISTO Tool”, in Predictably Dependable Computing Systems (B. Randell, J.-C. Laprie, H. Kopetz and B. Littlewood, Eds.), Springer, Berlin, Germany, 1995.
[3] J. Gray, (Ed.), “The Benchmark Handbook for Database and Transaction Processing Systems”, San Francisco, CA, USA, Morgan Kaufmann Publishers, 1993, 592 p.
[4] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie and D. Powell, “Fault Injection and Dependability Evaluation of Fault-Tolerant Systems”, IEEE Trans. on Comp, vol.42, no.8, 1993.
[5] P. Koopman and J. DeVale, “Comparing the Robustness of POSIX Operating Systems”, in *Proc. 29th Int. Symp. Fault-Tolerant Computing (FTCS-29)*, Madison, USA, 1999.
[6] P. Koopman and H. Madeira, “Dependability Benchmarking & Prediction: A Grand Challenge Technology Problem”, 1st IEEE Int. Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems; Phoenix, Arizona, USA, November 30, 1999.
[7] M. Vieira and H. Madeira, “A Dependability Benchmark for OLTP Application Environments”, 29th Int. Conf. on Very Large Data Bases (VLDB-03), Berlin, Germany, 2003.
[8] K. Buchacker and O. Tschaeche, “TPC Benchmark-c version 5.2 Dependability Benchmark Extensions”, http://www.faumachine.org/papers/tpcc-depend.pdf, 2003.
[9] A. Kalakech, K. Kanoun, Y. Crouzet and A. Arlat. “Benchmarking the Dependability of Windows NT, 2000 and XP,” in Proc. Int. Conf. on Dependable Systems and Networks (DSN 2004), Florence, Italy, IEEE CS Press, 2004.
[10] J. Durães and H. Madeira, “Characterization of Operating Systems Behaviour in the Presence of Faulty Drivers Through Software Fault Emulation”, in Proc. 2002 Pacific Rim Int. Symp. on Dependable Computing (PRDC-2002), Tsukuba, Japan, 2002.
[11] A. Brown, L. Chung, W. Kakes, C. Ling, D. A. Patterson, "Dependability Benchmarking of Human-Assisted Recovery Processes", Dependable Computing and Communications, DSN 2004, Florence, Italy, June, 2004
[12] D. Wilson, B. Murphy and L. Spainhower. “Progress on Deining Standardized Classes of Computing the Dependability of Computer Systems,” in Proc. DSN 2002 Workshop on Dependability Benchmarking, pp. F1-5, Washington, D.C., USA, 2002.
[13] J. Zhu, J. Mauro, I. Pramanick. “R3 - A Framwork for Availability Benchmarking,” in Proc. Int. Conf. on Dependable Systems and Networks (DSN 2003), San Francisco,,2003.
[14] Ji J. Zhu, J. Mauro, and I. Pramanick, “Robustness Benchmarking for Hardware Maintenance Events”, in Proc. Int. Conf. on Dependable Systems and Networks (DSN 2003), pp. 115-122, San Francisco, CA, USA, IEEE CS Press, 2003.
[15] J. Mauro, J. Zhu, I. Pramanick. “The System Recovery Benchmark,” in Proc. 2004 Pacific Rim Int. Symp. on Dependable Computing, Papeete, Polynesia, IEEE CS Press, 2004.
[16] S. Lightstone, J. Hellerstein, W. Tetzlaff, P. Janson, E. Lassettre, C. Norton, B. Rajaraman, and L. Spainhower. "Towards Benchmarking Autonomic Computing Maturity", 1st IEEE Conf. on Industrial Automatics (INDIN-2003), Banff, Canada, August 2003.
[17] SPEC - Standard Performance Evaluation Corporation, “SPECweb99 Release 1.02 (Design Document)”, http://www.spec.org/web99/, July 2000.
[18] J. Durães and H. Madeira, “Definition of Software Fault Emulation Operators: a Field Data Study”, in Proc. of the International Conference on Dependable Systems and Networks, DSN2003, San Francisco, CA, June 22 - 25, 2003.
[19] J. Durães and H. Madeira, “Generic Faultloads Based on Software Faults for Dependability Benchmarking”, accepted for publication at the Int. Conf. on Dependable Systems and Networks, Dependable Computing and Comm., DSN-04, Florence, Italy, June, 2004.
[20] J. Christmansson and R. Chillarege, “Generation of an Error Set that Emulates Software Faults”, in Proc. of the 26th IEEE Fault Tolerant Computing Symposium, FTCS-26, Sendai, Japan, pp. 304-313, June 1996.