Evaluating and Comparing the Impact of Software Faults on Web Servers

Naaliel Mendes, João Durães, Henrique Madeira
CISUC, Department of Informatics Engineering
University of Coimbra
{naaliel, jduraes, henrique}@dei.uc.pt

April 2010
Presentation Outline

- Research Problem
- Our Approach
- Methodology Specification
- Methodology Implementation
- Case Study
  - Performance and Dependability Results
- Conclusions
Research Problem

How a web server will behave if a software fault (ex., bug) is activated in one of their hosted web applications?

- No impact? Crash? Unresponsive?
- Abnormal resource usage? No idea?

Does it matter?

- At this moment millions of web servers are being used, many of them hosting web applications.
  - Developed under time constraint, by inexperienced programmers.

What is the problem? Why should it be solved?

- No methodology to evaluate how web servers are affected by software faults present in web applications.

Netcraf’s April-10 Survey: more than 205 millions of web sites
Our Approach

- Experimental method to evaluate and compare the impact of software faults on web servers (WS)

- How do we do that?
  - By collecting measurements on response time, response correctness, resource use, (...) of the web server when faults are injected in one of its hosted web applications.
    - State of the web server after each software fault activation:
      - Crash, Unresponsive, Wrong Results, Resource Use Penalty, No Impact.
  - Two phases: 1) Golden Run 2) Fault Injection Run
Methodology Specification

- **Generic**
  - Aimed at addressing any type of web servers and web applications.

- **Technology-independent**
  - Can be implemented by different teams using different technologies.
    - But you MUST monitor the web servers properties and follow the procedures we specify.

- Includes classical dependability benchmark elements
  - Experiments rules, Workload, Faultload, Measures, and Instrumentation Tools.
Methodology Specification

- **Target System Architecture**
  - **Web Serving System**

![Diagram of Web Serving System]

- **Server machine**
  - **WEB SERVER** *(Tomcat | SJSWS | Jetty)*
  - TPC-W Web App
  - Integrity Checker Application

- **Database machine**
  - DB2 Database
  - Windows XP Operating System

- **Client machine**
  - TPC-W Remote Browser Emulator (activate faults)
  - Integrity Checker Client
  - Windows XP Operating System
Methodology Specification

Target System Components

<table>
<thead>
<tr>
<th>System Under Benchmark (SUB)</th>
<th>Diagnostic System (DS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEB SERVER (Benchmark-target)</td>
<td>Workload Client</td>
</tr>
<tr>
<td>Workload Application (Fault Injection Target)</td>
<td>Management System (MS)</td>
</tr>
<tr>
<td>Integrity Checker Application</td>
<td>Integrity Checker Client</td>
</tr>
</tbody>
</table>

Workload
Methodology Specification

**Experiment Rules**

- **Phase 1.**
  - Golden run. It is used to determine the baseline performance of the system under benchmark (without software faults).
    - Executed three times and the result is the average of the three executions

- **Phase 2.**
  - Execution of the workload in the presence of faults to evaluate the dependability properties of the benchmark target.
    - The number of executions depends on the number of faults
Methodology Specification

- **Workload**
  - Web application that simulates the activities of a business oriented transactional web server.
    - Internet commerce application environment.
      - Multiple concurrent on-line browser sessions.
      - Involving both static and dynamic content.
      - Access and update patterns to a database.

- **Faultload**
  - Set of software faults that represents realistic bugs found in applications.
    - We can find in literature some field studies on the representativeness of software faults (e.g., [Durães, 2006]).
Methodology Specification

- **Measures**
  - **Client-side perspective:**
    - Response Time, Response Correctness, Throughput.
  - **Server-side perspective:**
    - Web Server Execution Duration, Web Server Crash (System Process), Log File Size (Number of Exceptions), Processor Use.

- **Failure Modes**
  - **State of the web server:**
    - Crash, Unresponsive, Resource Use Penalty, Wrong Results, and No Impact.

- **Mapping Measures with Failure Modes**
  - How do I know that the web server crashed?
Methodology Implementation

Diagnostic Approach

- Integrity Checker (IC) verify the performance and error propagation
  - Failure Mode: Web Server Unresponsive, Wrong Results.
  - How do we verify error propagation?
    - IC is free of software faults and provides expected responses.
      - Client knows the expected result of a request (e.g., Spain => Europe)
Diagnostic Approach

- **Web Server Processor Monitor** observe the status of the web server process, checking if it is running or not.
  - **Failure Mode: Web Server Crash.**

- **Resource Consumption Monitor** observe log file size, number of exceptions, processor use (PerfMon).
  - **Failure Mode: Resource Use Penalty.**

- **Data Analyzer.** Summarize the data from different systems source (output files with different formats).
Methodology Implementation

**Workload**

- We used a Java implementation of the TPC-W Benchmark ([http://www.tpc.org/tpcw/](http://www.tpc.org/tpcw/))
  - TPC’s benchmarks “delivers trusted results to the industry”
  - Easy to install and run, easy to inject software faults (in the servlet code). Used in other research works.

**What is TPC-W?**

- Ecommerce workload that simulates the activities of a retail store web site.
- Emulated users can browse and order books from the website.
- Simulates the same HTTP network traffic as would be seen by a real customer using a browser.
  - 10 users were emulated, multiple requests to the web server.
  - Only one instance of the TPC-W application.
Methodology Implementation

**Faultload**

To select which software faults to include in our faultload, we used a field study on software fault targeting web applications [Fonseca, 2008].

- Each fault is emulated where it could realistically exist in the source code.

- The faults were injected manually.
  - There is no software fault injection tool targeting the bytecode of Java Applications.

- Two of the most representative faults types were injected in different points of the web-application source-code.
  - MIFS (Missing IF Construct plus Statements).
  - WVAV (Wrong Value Assigned to a Variable).

- 250 software faults (125 of each type) were injected.
Fault Injection 5-Step-Approach

1. Analysis of the source-code to find out a fault injection point.
2. Injection of the software fault (MIFS, WVAV).
3. Compilation of the faulty source-code.
4. Copying of the faulty web application to a specific directory.
5. Returning the source-code to the original state (free of fault) and repeat these steps.
Methodology Implementation

Examples of Fault Injection

TPCW_Database.java (Class used by TPCW servlets)

- **MIFS (Missing IF Construct Plus Statement)**
  
  ```java
  public class TPCW_Database {
    (...)
    public static synchronized Connection getConnection() {
      (...)
      if (maxConn == 0 || checkout < maxConn) {
        con = getNewConnection();
        totalConnection++;
      }
    } (...)
  }
  ```

- **WVAV (Wrong Value Assigned to a Variable)**
  
  ```java
  public class TPCW_Database {
    (...)
    private static final String jdbcPath = "jdbc:db2:tpcw";
    private static final String jdbcPath = "jdbc:db2:tpcwXXXXXXX";
  }
  ```
Methodology Implementation

Management System

Brief summarization of its execution flow:

- Starts after the initialization of the operating system
- Executes preliminary operations to ensure a fresh fault injection run start.
  - Copy the faulty web application to the web server directory that will be used in the next fault injection run
  - Clean-up temporary directory.
- Starts monitoring tools, the web server (which hosts the faulty web application), and the client applications (TPCW Client and Integrity Checker Client).
- Executes post conditions operations.
  - Normalize, organize, and analyze the output files generated during the fault injection run
- Ends by rebooting the operating system
Case Study: Experimental Setup

- **Web Servers**
  - Apache Tomcat 6.0
  - Sun Java System 7.0 (SJSWS)
  - Jetty Web Server 6.2.1

- **Workload**
  - TPC-W Web Application
    - IBM DB2 Database

- **Operating system**
  - Windows XP

- **Hardware Platform:**
  - Server-side: P4 3.0 GHz, 1.46Gb RAM, 80Gb of HD

Hardware platform and systems surrounding the BT were the same across all experiments.
Case Study: Performance Results

- **Response Time (AVG) & Throughput (Res/sec)**

  The fault activation forced a premature termination of the workload application, reducing the number of requests sent to the web server component.

  **Why did response time go down?**

  The fault activation forced a premature termination of the workload application, reducing the number of requests sent to the web server component.
Case Study: Dependability Results

- **Data Integrity**
  - No integrity data errors registered by our Integrity Checker Application.
  - Web server provided expected responses for the requests targeting the application free of fault (Integrity Checker).

- **Availability (%)**

  The high number of exceptions thrown by the web app. consumed all resources of the web server, making it to be unresponsive.
## Case Study: Resource Use Results

### Disk Use Comparison (%)

### Percentual Distribution of Log File Size

<table>
<thead>
<tr>
<th>Log File Size</th>
<th>TOMCAT</th>
<th>SJSWS</th>
<th>JETTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 1 MB</td>
<td>96.40%</td>
<td>75.20%</td>
<td>73.60%</td>
</tr>
<tr>
<td>1MB &lt; 10 MB</td>
<td>1.60%</td>
<td>22.40%</td>
<td>6.80%</td>
</tr>
<tr>
<td>10MB &lt; 40 MB</td>
<td>0.40%</td>
<td>1.20%</td>
<td>18.00%</td>
</tr>
<tr>
<td>40MB &lt; 80MB</td>
<td>0.40%</td>
<td>1.20%</td>
<td>1.20%</td>
</tr>
<tr>
<td>&gt;= 80MB</td>
<td>1.20%</td>
<td>0.00%</td>
<td>0.40%</td>
</tr>
</tbody>
</table>

Log File Size (Worst Case)
- 5 min = + 146 MB
- 1 hour = + 1.71 GB
- 1 day = + 41 GB
## Processor Use Comparison (%)

<table>
<thead>
<tr>
<th>Fault Types</th>
<th>TOMCAT</th>
<th>SJSWS</th>
<th>JETTY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AVG</td>
<td>MAX</td>
<td>AVG</td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>BASELINE</td>
<td>20.36</td>
<td>39.17</td>
<td>19.03</td>
</tr>
<tr>
<td>MIFS</td>
<td>18.79</td>
<td>49.62</td>
<td>19.29</td>
</tr>
<tr>
<td>WVAV</td>
<td>16.35</td>
<td>52.06</td>
<td>17.85</td>
</tr>
</tbody>
</table>

Web Servers have different behavior to use the processor resource. These results were collected using the same instrumentation tools.
Case Study: Failure Mode Results

- Which is the most robust of the web servers we evaluated?
Conclusions

Lessons learnt

- Software faults present in web applications affect the performance and the dependability of web servers in a different way across web servers.
  - In some cases the unavailability rate was around 2%.

- The TPC-W application tolerated many software faults.
  - However, in some cases, it generated a large number of exceptions that contributed to make the web server unresponsive.

- The manual injection of the software faults makes hard, but not impossible, the reproduction of our injection campaign.
  - We believe that the information and references we provided can guide anyone interested in reproducing our experimental setup.
Conclusions

Future Work

- Addition of new types of software faults in our faultload.
- Conduction of new case studies to demonstrate the portability and scalability of our approach in other environments.
- Advance this work and propose a dependability benchmark for web servers considering the impact of software faults in hosted applications as the main benchmark measure.
Thank you!

Increase the impact and visibility of your research with

AMBER Raw Data Repository

www.amber-project.eu/repository

Visit the web site!