

► **Performance** v2.0.4

Copyright © 2005-2008 Roberto Velasco, Aritz Rabadan, Mikel Enrique, Gorka Vicente.

<b>1.</b>	<b>INTRODUCTION</b>	<b>3</b>
<b>2.</b>	<b>CONFIGURATION</b>	<b>4</b>
<b>2.1</b>	<b>SOFTWARE</b>	<b>4</b>
<b>2.2</b>	<b>MACHINES</b>	<b>4</b>
<b>2.3</b>	<b>HDIV</b>	<b>4</b>
<b>3.</b>	<b>RESULTS</b>	<b>5</b>
<b>3.1</b>	<b>AVERAGE RESPONSE TIME</b>	<b>5</b>
3.1.1	TOMCAT 5.5.20	6
3.1.2	WEBSPHERE 6.0.1	6
<b>3.2</b>	<b>CPU USE</b>	<b>7</b>
3.2.1	TOMCAT 5.5.20	7
3.2.2	WEBSPHERE 6.0.1	7
<b>3.3</b>	<b>USE OF MEMORY</b>	<b>8</b>
3.3.1	TOMCAT 5.5.20	8
3.3.2	WEBSPHERE 6.0.1	8
<b>3.4</b>	<b>RESULTS ANALYSIS</b>	<b>9</b>
3.4.1	STATE IN CLIENT VS. STATE IN SERVER	9
3.4.1.1	STATE IN CLIENT SIDE	9
3.4.1.2	STATE IN SERVER	10
<b>4.</b>	<b>CONCLUSIONS</b>	<b>11</b>
<b>5.</b>	<b>REFERENCES</b>	<b>12</b>

---

## 1. INTRODUCTION

---

HDIV has been created in order to solve most of web vulnerabilities, adding new security functionalities to applications developed in Struts 1.x, Struts 2.x, Spring MVC and JSTL in a transparent way to the programmer and without increasing development complexity.

The aim of this document is to evaluate HDIV's performance applied to different web applications over different web servers.

The different possible HDIV configurations make it necessary to study and compare each possibility and choosing the best configuration for each particular case. In this article, the 3 HDIV strategies are analyzed (Memory, Cipher and Hash) comparing their response times with applications with no integrity solution.

Base line measurements have been used to calculate upper measurements corresponding to each solution.

At the end of the document the conclusions of the study are explained.

---

## 2. CONFIGURATION

---

All the tests have been executed in a automatized environment, using the following software and hardware.

This hardware and software was configured in a test environment in a way that simulates a real production environment server activity.

### 2.1 Software

- JMeter 2.2 running on J2SDK 1.5.0\_06 with default JVM settings [1]
- Jprofiler, version 4.2.1 [2]
- Web servers (with default JVM settings):
  - Tomcat 5.5.20 and SUN's JRE, 1.5.0\_06 version
  - Websphere 6.0.1 and Websphere's JRE v6

### 2.2 Machines

- JMeter running GUI mode on Intel Pentium 4 CPU 3.2 GHz, 1 GB DDR
- Tomcat running on Intel Pentium 4 CPU 3.2 GHz, 1 GB DDR
- LAN 100 Mb/s

### 2.3 HDIV

All the HDIV strategies have been tested: Memory, Cipher and Hash.

### 3. RESULTS

To guarantee that the obtained results are valid, each test has been executed 1000 times for each configuration. Also, the tests are executed when the *response time*, *CPU use* and *consumed memory* are stabilized to avoid non reliable results.

The results obtained from an environment with no integrity solution are considered as a "base line" measurement to calculate higher performances. The upper measurements of *response time* and *CPU use* were obtained taking away the "base line" to the specific solution measurement and expressing it as a percentage compared to the "base line".

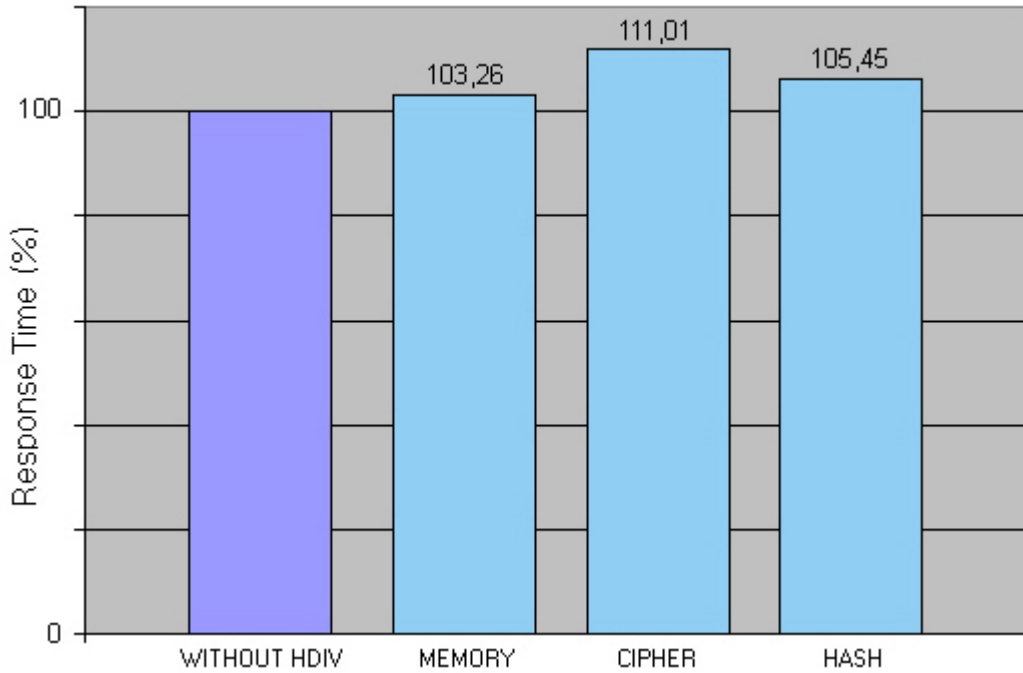
Performance Mesurements	Mesuarement Definition	Calculation Method
Average Response Time (%)	Average response time was calculated for each repetition. The average of repetitions represents the general performance mesuarement.	Calculated automatically by <i>JMeter</i> .
CPU use (%)	Use of processor (percentage of CPU's maximum potencial) in 5 seconds intervals. Average value of this parameter was calculated for each repetition. Average of repetitions represents performance general mesuarement.	Calculated automatically by <i>JProfiler</i> .
Memory use (MB)	Memory use was calculated for each repetition. Average of repetitions represents the general measurement for the performance.	Calculated automatically by <i>JProfiler</i> .

#### 3.1 Average Response Time

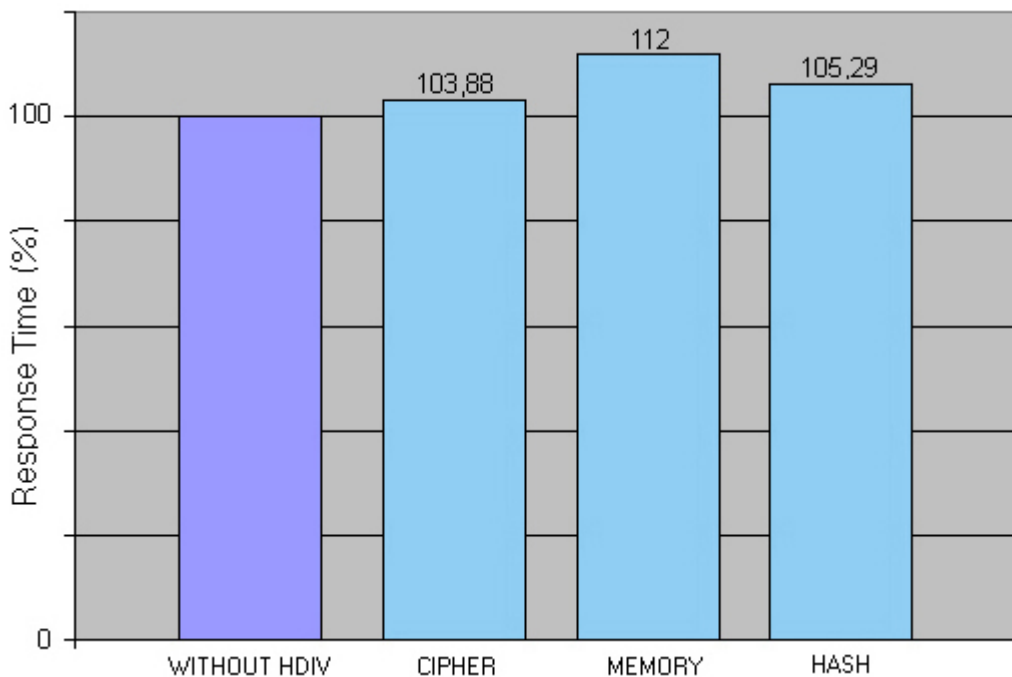
Using the average response time (in milliseconds) and the upper average response time consumed by HDIV (difference in percentage with the no integrity solution base line measurement) the average result of repetitions was obtained for each strategy or HDIV solution.

The charts below show the average response time for each HDIV strategy, being 100 the percentage of time consumed by the base line application (the one without HDIV solution).

### 3.1.1 Tomcat 5.5.20



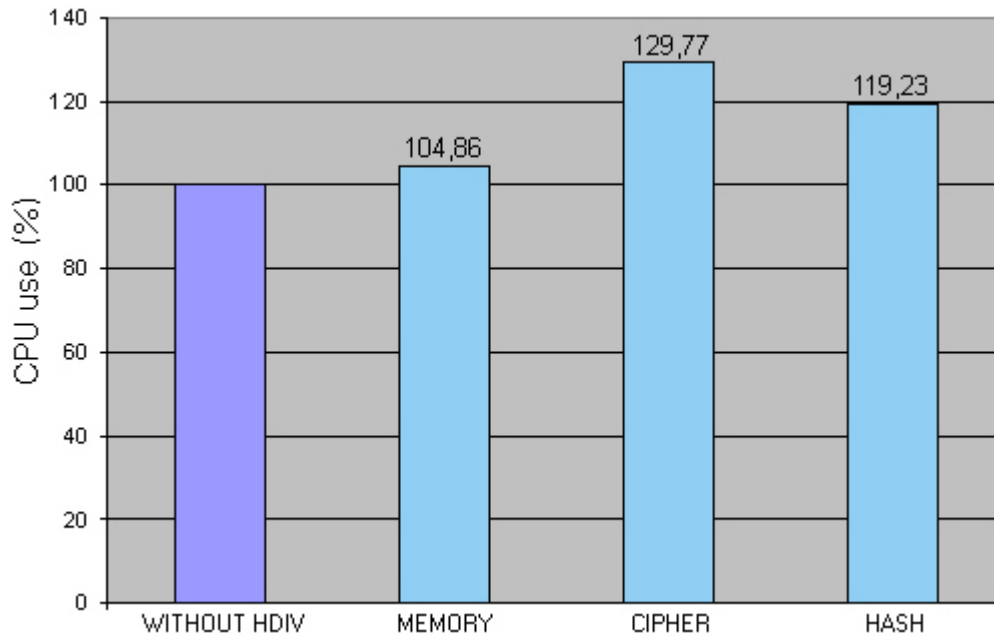
### 3.1.2 Websphere 6.0.1



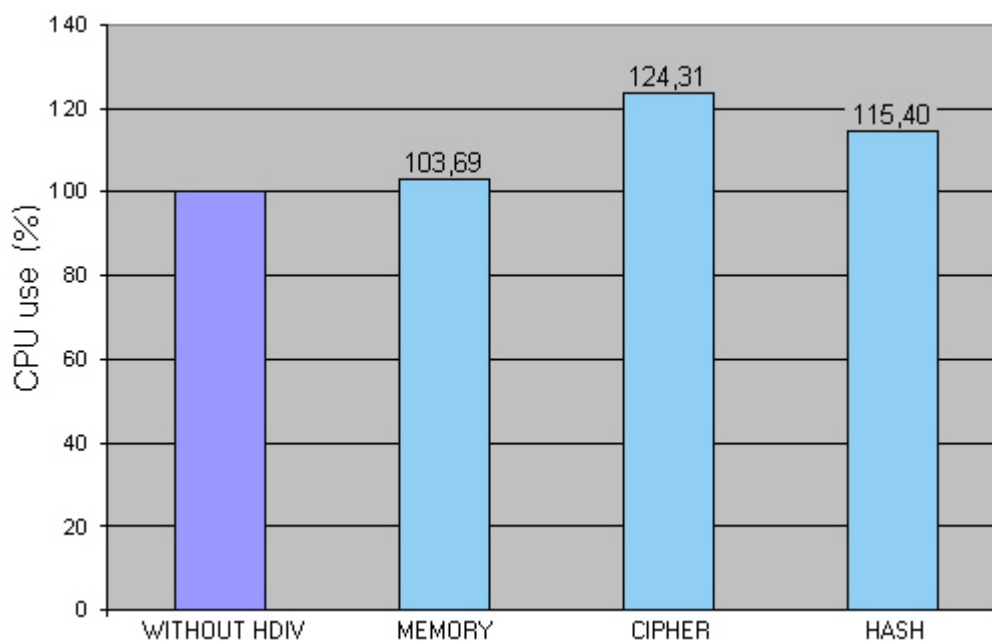
### 3.2 CPU use

The charts below show the average CPU use that was obtained during the tests for each strategy, being 100 the percentage of CPU consumed by the application without HDIV.

#### 3.2.1 Tomcat 5.5.20



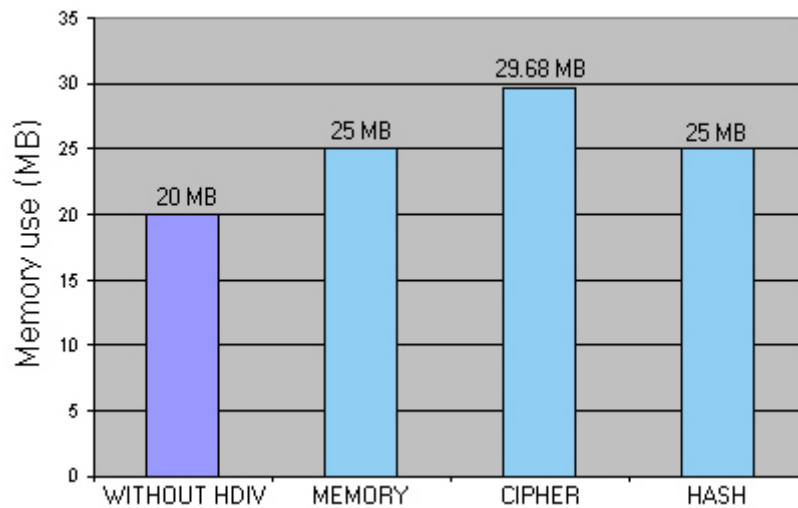
#### 3.2.2 Websphere 6.0.1



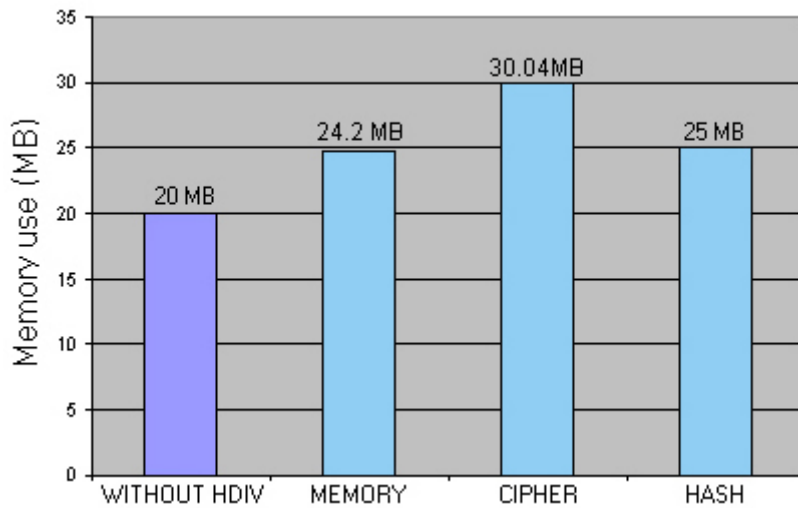
### 3.3 Use of Memory

The charts below show the average use of memory obtained in the test servers.

#### 3.3.1 Tomcat 5.5.20



#### 3.3.2 Websphere 6.0.1



## 3.4 Results Analysis

In both test environments (Tomcat and Websphere) the results have been very similar for each strategy. So we can state that the environment has no influence in HDIV.

According to the results, Memory is the best strategy. If we applied it to an unsecured application it only increases its response time in a 3,5%. The other two test parameters as well (CPU use and memory use) get a remarkable result. This strategy provides data integrity and confidentiality to the application.

The second best option based on the average response time, CPU use and memory use is the Hash strategy. It requires a 5,3% more time to response to requests than an unsecured application. This solution provides data integrity.

As the third option we have Cipher strategy which adds a 11,5% of time to the average response time of an unsecured web application. It provides data integrity and confidentiality.

### 3.4.1 State in client VS. State in server

Once we have analyzed the obtained results, let's see the advantages and disadvantages of storing the state in the client (Cipher strategy) or in the server (Memory strategy).

Hash strategy combines the functionalities of the Memory and Cipher strategies storing state information in both sides.

#### 3.4.1.1 State in client side

Advantages:

- No memory consumption between requests
- No concurrency issues
- No browser back problem
- Restart availability
- Cluster friendlier

Disadvantages:

- Higher bandwidth
- Higher CPU
- Higher client side memory usage

### 3.4.1.2 State in server

Advantages:

- Bandwidth usage is low
- CPU processing time usage is low

Disadvantages:

- Memory consumption on the server per user in a session
- Memory consumption of several views to support the back-button
- Clustering – moving the session to another node of the cluster

---

## 4. CONCLUSIONS

---

As we have seen in this study the amount of time added by HDIV to an average response time of an unsecured application, the CPU use and the memory use depend on the chosen strategy.

The extra time added by HDIV must be analyzed and evaluated by system administrators to determine if it is admissible for the application, as well as the importance or necessity of data confidentiality of the application.

So, this study proves that HDIV's consume of resources is minimum compared with an unsecured web application. It only increases its response time in a 10% in the worst case. So, it is an ideal way of avoiding data integrity vulnerabilities as well as providing data confidentiality with very low cost.

---

## 5. REFERENCES

---

- [1]. Apache JMeter  
<http://jakarta.apache.org/jmeter/>
  
- [2]. ej-technologies Jprofiler  
<http://www.jprofiler.com>