How Can Apache Help to Teach And Learn Automatic Control?

Teresa Alvarez, David Herrero, Jesús Francisco, David Madrigal
Dpt. Automatic Control
University of Valladolid
Valladolid, Spain
tere@autom.uva.es

Abstract— **Searching for examples to encourage and motivate students in a subject is not a trivial task. Future Computer Science and Telecommunications Engineers tend to look at Automatic Control reluctantly. They do not relate their working life to it. But, if we blend examples from two areas: telecommunications and control, maybe we can change their minds. These engineers work with web servers: systems that deal with a lot of requests and must give fast and reliable service to users. CPU and memory usage could improve their performance if automatic control techniques are applied. Apache is one of the most well known web servers and it is at this point where Apache can help us.**

Web server; Apache; Automatic Control; Internet server; PID; predictive control.

I. INTRODUCTION

Automatic control combines concepts and ideas from Mathematics and Engineering. There are many different techniques that allow real systems to be controlled. For instance, we can apply control techniques to ensure that a reactor does not reach a critical point, or we can control an airplane velocity or altitude.

Future Computer Science and Telecommunications Engineers (in Spain) learn about Communications Networks and Internet and they also learn automatic control techniques. It is sometimes difficult to find systems that appeal to them from both sides: communications and control. Most control books present electrical or mechanical examples. Rerefence [1] is the only textbook that gives examples of computing systems and how control techniques can improve their performance.

Apache ([2]) is a web server developed and maintained by the contributors to the Apache Software Foundation. There are variables that are critical for the right performance. As presented in [3], we can apply automatic control techniques to achieve better server performance. It should be decided which variables will act as inputs or manipulated variables and which variables will be the outputs or controlled variables. The work in [3] is used as reference. We aimed to design a tool that could be used in the classroom and act as a pool of control techniques to experiment and compare results.

CONAPA (CONtrolling APAche) is the user's tool. It has been developed taking into account that it should be easy to use. There are different working options:

- Monitoring tool: the user watches the system's evolution.
- Simulation tool: the user can simulate the system under different situations.
- Real-time control: the user defines the controller parameters and the real web server works following the controller calculations.

It works both as an online and as a virtual laboratory. Data can be saved and loaded in different programs to analyze results. This application manages the data capture for identification purposes, the simulation environment and the control of the real system.

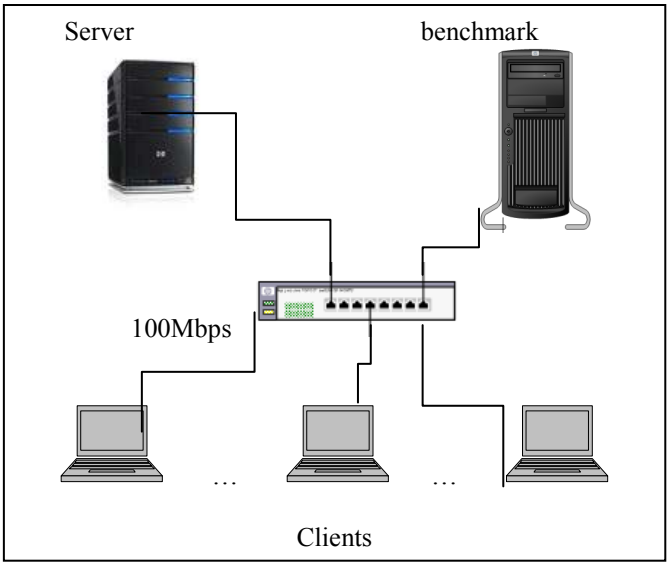

Figure 1. Working environment

The following sections describe our work. We have followed a top down approach, from general to particular. So Section II describes the system's architecture. Then the relevant variables are explained as well as how they are read and calculated from the server in Section III. Section IV

This work was supported in part by the Junta de Castilla y León under Grant VA037A08 and by the CICYT under grant DPI2007-66718-C04-02

describes the new module included in Apache. Then, a brief description of the type of controllers that are implemented is given in Section V. CONAPA is presented from the user's point of view (Section VI) and finally some conclusions and future work are outlined.

II. SYSTEM'S ARCHITECTURE

The working environment () consists of a server running the modified Apache connected through a 100 Mbps LAN to a machine running a synthetic workload generator. Thus, we can work under different traffic conditions. The server is a Pentium IV Mobile 1.8 MHz (2GB RAM memory). The benchmark runs on an Intel dual core 2.24 MHz (4GB RAM memory). Both machines run under Ubuntu (www.ubuntu.com).

The clients' machines will connect to CONAPA's web page. From there, they could work with the different options that will be explained in Section VI.

The benchmark is Curl-Loader ([4]). This tool is open-source and written in C. It simulates the application load and behavior of HTTP/HTTPS and FTP/FTPS clients. Information about each virtual client can be statistically analyzed.

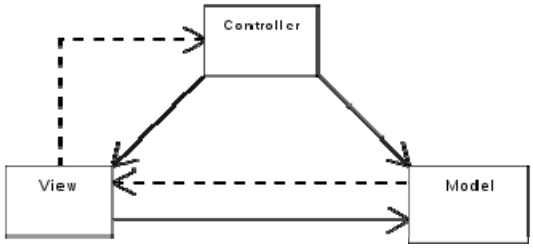

Figure 2. MVC architecture

Our application has been developed using the LAMP (Linux, Apache, Mysql and Php) architecture ([5]). The software architecture, i.e., how the system is organized inside, follows the Model-View-Controller (MVC, Figure 2) architecture (first described in [6]).

III. SYSTEM'S VARIABLES

Choosing manipulated and controlled variables is directly related with how and what we want to do. It is advisable to choose variables that have a real importance in the system and that can be easily modified and interpreted. There are several key variables in an Apache web server.

From the administrator's point of view, it is very important to keep the CPU and memory use within a desired band (from now on, these variables will be denoted CPU and MEM). If high values are reached, the system could overload or there might be a slow response to bursts in workload. CPU and MEM cannot be read directly and should be calculated (see the following subsection). These two variables are our system's outputs.

The controlled variables change their value as the result of a change in the manipulated variables. If we want CPU and MEM to follow a reference, i.e., to reach and stay around a value, we need to find which variables could affect them, and more importantly, we need these input variables to change their value dynamically.

Apache's KeepAliveTimeout (KA, [2]) directive is the number of seconds the server will wait for a subsequent request before closing a connection. The higher the value, the more server processes will be kept occupied waiting for idle clients. So, CPU and MEM are underused. If KA is very low, connections could be terminated too early. So KA should be given values that neither overload nor underload the server.

The maximum number of clients (MaxClients, MC for short) that can connect to the Apache server directly influences the use of the CPU and MEM. The more clients are connected, the higher the usage.

MC and KA cannot be modified in standard Apache. If we want to change their value dynamically, the Apache source code has to be modified (Section IV).

A. How to calculate MEM and CPU utilization on line

There are tools in Linux that calculate the use of MEM and CPU, but they group consumptions together by processes and not by application. So a script was written in Python ([10]).

The script in Figure 3 looks for information from each process that has been created. Linux stores these records in the directory /proc.

Cpu.py

```
#! /usr/bin/python
import sys
import time
import os
import pwd
import math
class PStat:
   def __init__(self, pid):
        fd = open("/proc/%d/stat" % pid)
        l = fd.readline().split()
        fd.close()
        self.pid = pid
        self.comm = l[1][1:-1]
        self.utime = int(l[13])
        self.stime = int(l[14])
    def get_time(self):
        return self.utime + self.stime
class SStat:
   def __init__(self):
        fd = open("/proc/stat")
        l = fd.readline().split()
        fd.close()
        self.uptime = int(l[4])
def get_pids():
    r = []
    for d in os.listdir("/proc"):
       try:
           r.append(int(d))
       except ValueError:
           pass
    return r
def get_pstats():
    r={}
    for pid in get_pids():
        r[pid] = PStat(pid)
    return r
def get_rss_mem(pid):
    """Return the process RSS in MB"""
    fd = open("/proc/%d/statm" % pid)
```

```
    l = fd.readline().split()[1]
    fd.close()
    return int(l) * os.sysconf('SC_PAGE_SIZE') /
1024.0 / 1024.0
total = 0
memoria = 0
tab = get_pstats()
sstat = SStat()
time.sleep(1)
tab1 = get_pstats()
sstat1 = SStat()
for pid in tab1:
    if tab1[pid].comm=="httpd":
        #load = 100.0 * tab1[pid].get_time() /
sstat1.uptime
        load = 100.0 * (tab1[pid].get_time() -
tab[pid].get_time()) / \
            (sstat1.uptime - sstat.uptime)
        memoria = memoria +get_rss_mem(pid)
        total = total + load
total=('%5.3f' % total)
memoria= ('%5.3f' % memoria)
   print str(total) + '\t' + str(memoria)
```

Figure 3. Memory and CPU utilization

IV. APACHE WEB SERVER

This section explains Apache's architecture at a basic level. Session flow and conceptual architecture are described first. Then, the necessary changes in Apache and how to program them are explained.

A. Apache's architecture: session flow

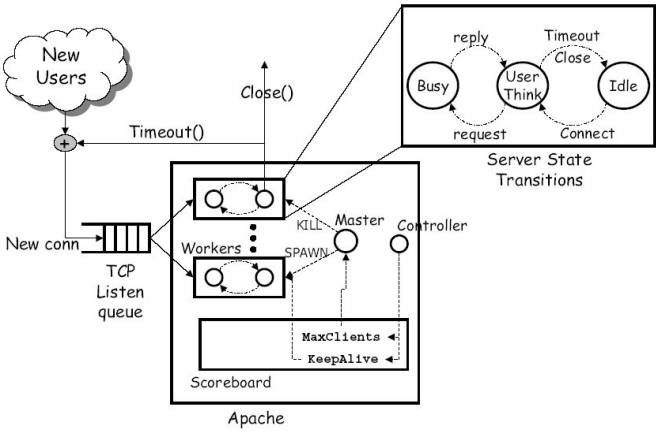

Figure 4. Apache's session flow

Although there are later versions, we are working with Apache 1.3 on Linux. This version is structured as a pool of 'worker' processes, monitorized and controlled by a master process (Figure 4). Worker processes manage the communication with the Web clients. At any time, a worker process can only deal with one connection and it is its responsibility until the connection ends. Thus, the worker process is inactive between successive petitions during the connection with the client.

MaxClients (MC, Section III) limits the size of the pool of the worker processes. KeepALive Timeout (KA, Section III) fixes the maximum time a worker process can be in the 'user think' state before the client's TCP connection is finished. If KA is too big, CPU and memory are underused. Decreasing its value, worker processes spend less time in the 'user think' state and the CPU usage increases. But, if the value is very small, TCP connections will finish before time and the benefits of persistent connections are lost.

B. Apache's conceptual architecture

Apache has a modular architecture ([6], [7], [8]) that makes it possible to increase its functionalities and customize it to our own purposes. Apache's core component defines the basic operations of a web server. The modules implement how to serve clients' requests, but the core calls the handlers in modules in a certain order that can be changed using the adequate directives.

The conceptual architecture is summarized in Figure 5. The PHP module is available to download, but it is not included in the standard installation. The mod_control.c is a new module we have written. Its main purpose is to create a socket that reads/writes KA and MC. Thus their values are updated each sampling time. More details in subsections C and D.

C. Changes in Apache source code

Configuration parameters are saved in the http.conf file. When Apache is launched, these values are read and cannot be modified until a restart of the server happens. So, MC and KA have default values that will be active all the time and cannot be changed during running time, i.e., dynamically every sampling period (the controller will calculate new values at certain time instants, sampling time and send to the Apache web server).

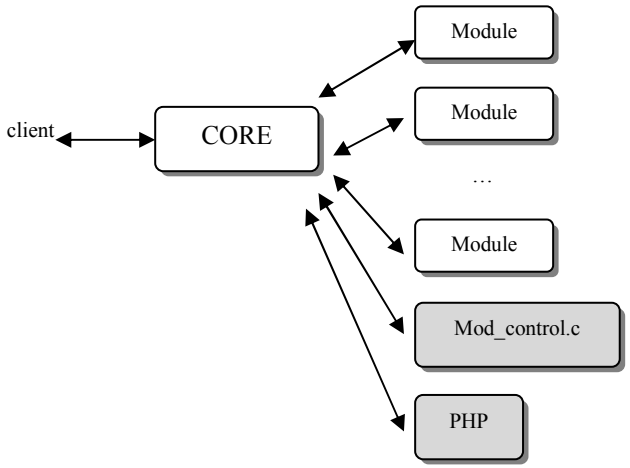

Figure 5. Apache's conceptual architecture

The first idea was to change the http.conf every sampling time and then carry out a 'graceful restart'. Thus, the parent process tells the child processes to terminate once they finish what they are doing. The new configuration file is read and the new child processes follow the new configuration. However, this approach is not very convenient because Apache is restarted every sampling time (about every 2 seconds) and only two parameters need new values. So, it was a computationally demanding approach.

The approach followed was to modify Apache's source code. Thus, MC and KA are no longer parameters, but

variables. Two new modules were written. This new source code manages the variables and the interaction with the user's interface. The following sub-sections describe these topics.

D. MC and KA as variables

MC is called ap_daemons_limit and KA is called keep_alive_timeout in Apache's source code. Their declaration was changed from integer to integer pointers. Thus, the process receiving the petitions could change their value.

Changes regarding MC:

a. include/http_conf_global.h. The variable is declared as a pointer: extern API_VAR_EXPORT int * ap_daemons_limit;

b. main/http_conf.c. Shared memory is reserved:

```
Clave = ftok("/bin/ls",33);
          if (Clave==-1)
                exit(0);
          Id_Memoria          =          shmget
(Clave,sizeof(int),0777| IPC_CREAT);
          if (Id_Memoria == -1)
                exit(0);
           ap_daemons_limit        =        (int
*)shmat(Id_Memoria,(char *)0,0);
          if (ap_daemons_limit == NULL)
                exit(0);
```

c. main/http_main.c. The variable is declared as a pointer:

d. Other files: the variable should be addressed as the pointer's content.

Changes regarding KA:

a. Include/http.h. Pointer declaration: int *keep_alive_timeout.

b. Main/http_config.c. Shared memory is reserved:

```
Clave = ftok("/bin/ln",33);
            if (Clave==-1)
                  exit(0);
           Id_Memoria = shmget (Clave,sizeof(int),0777 |
IPC_CREAT);
            if (Id_Memoria == -1)
                  exit(0);
           s->keep_alive_timeout = (int *)shmat(Id_Memoria,(char
                *)0,0);
           if (s->keep_alive_timeout == NULL)
                exit(0);
```

When the new MC was smaller than the old one, Apache did not kill the excess of child processes once they had finished their job. So we wrote a method in Apache that runs periodically. It checks whether the MC (ap_daemons_limit) is smaller than its previous value (ap_maxclients_ant). If it is, the remaining processes are destroyed. This source code is in the method static void perform_idle_maintenance(void) in http_main.c (Figure 6).

```
if(*ap_daemons_limit<ap_maxclients_ant)
     {
for(j=*ap_daemons_limit;j<ap_maxclients_ant;j++)
               {
                    int status;
            if(j>=max_daemons_limit)
                break;
            ss = &ap_scoreboard_image-
>servers[j];
            status = ss->status;
            if (ss->timeout_len)
              {
              parent_score *ps =
&ap_scoreboard_image->parent[j];
               kill(ps->pid,SIGKILL);

ap_update_child_status(j,SERVER_DEAD,(request_rec
*)NULL);
             }
          }
      ap_maxclients_ant = *ap_daemons_limit;
  }
else if(*ap_daemons_limit>ap_maxclients_ant)
  {ap_maxclients_ant = *ap_daemons_limit;}
```

Figure 6. static void perform_idle_maintenance(void)

E. Apache new module

The changes explained in the previous sub-section allow the two input variables to be modified dynamically, but we need a way to send the information in and out of Apache.

A new Apache module (mod_control.c, see pseudocode in and source code in http://www.isa.cie.uva.es/~tere/) has been written. Basically, this code opens a socket on the server side and 'tells' Apache which internal parameters can be modified dynamically.

```
MODULE mod_control
 include <libreries>
 define CONSTANTS
 var variables
 PROC open_server
   IF process_can_be_created THEN
     EXECUTE  open_connection
   END IF
 END//open_server
 PROC abrir_conexion
 IF socket_cannot_be_created THEN
    ERROR
    EXIT
 END IF
 IF not_poss_to_send_inf_kernel_about_socket
THEN
    ERROR
    EXIT
 END IF
 IF not_poss_link_socket_connection_inf THEN
    ERROR
    EXIT
 END IF
 IF socket_doen't_listen THEN
    ERROR
    EXIT
 END IF
 IF no_se_puede_tomar_la_señal_socket
    ERROR
    EXIT
 END IF
 WHILE module_in_execution
  IF connection_cannot_be_accepted
      CONTINUE
      EXIT
   END IF
 IF child_process_can_be_created THEN
    IF data_received THEN
```

```
        *ap_daemons_limit=datos;
        ap_set_keep_alive_timeout(datos)
        IF confirmation_not_send TEHN
          ERROR
          EXIT
        END IF
      END IF
    END IF
  END WHILE
  END//open_connection
/* procedure that Apache looks for to execute
the module */

  PROC main
         EXECUTE  open_server
  END//main
END //mod_control
```

Figure 7. Simplified mod_control.c pseudocode

F. php module

The php module is installed as an Apache module, so when Apache starts, php also starts. As it is installed as shared, all the advantages of the dynamic linking/loading of *Dynamic Shared Objects* (DSO) will be available. This mechanism provides a way to build a piece of program code in a special format for loading it at run-time into the address space of an executable program ([11]).

The php application will be saved in Apache's *htdocs* directory. It is composed of:

a. User's interface. Html code that will be accessed via the client's web browser.

b. Simulation code. Scripts in php

c. Control of the web server. A collection of php scripts. The client's socket is programmed here (Figure 8 and pseudocode). This part sends requests to mod_control.c when KA and MC are needed to calculate the next control signals.

d. Access to the mysql database: queries, save data, retrieve information, …

```
<HTML>
<BODY>
<FORM METHOD="post" ACTION="socket.php">
<p>MaxClientes <input type="text" name="mc"
size="30" value=""></p>
<p>KeepAlive <input type="text" name="ka"
size="30" value=""></p>
<p><input type="submit" value="Enviar datos"
name="enviar">
</FORM>
</BODY>
<HTML>
socket.php
<?php
        // Create the socket
        if(($sockd = @socket_create(AF_INET,
SOCK_STREAM,SOL_TCP))<1)
                die("Unable to create
socket:" .
socket_strerror(socket_last_error()));
if(@socket_connect($sockd,"127.0.0.1", 3490)
== FALSE)
                die("Unable to connect:" .
socket_strerror(socket_last_error()));
        $buffer =
$_POST['mc'].";".$_POST['ka'];
        if(@socket_write($sockd, $buffer) <
strlen($buffer))
                die("Unable to connect:" .
socket_strerror(socket_last_error()));
        unset($buffer);
        if(($buffer = socket_read($sockd,
100)) == FALSE)
                die("Unable to read from
socket:" .
socket_strerror(socket_last_error()));
        socket_close($sockd);
        echo $buffer;
?>
```

Figure 8. php socket

V. CONTROLLERS

This section briefly describes the controllers that students can use to test their knowledge. We have included PIDs and a predictive controller (GPC).

The system we propose here is a MIMO. There are two inputs and two outputs, although the coupling between variables is not very strong. The reasons for choosing these two methods are: PID is the most widespread controller and students of every Engineering degree learn it. Predictive control is an advanced control methodology with a relatively important influence in industry and the MIMO formulation is inherent to the controllers. So, students can compare what happens when working with a MIMO system, but considered as two SISO ones or working with the MIMO system itself.

```
SOCKET.PHP
Create socket
If error then print('error')
Connect
If error then print('error')
Collect variables
Write variables in socket
If error then print('error')
Read variables
If error then print('error'
Close socket
```

Figure 9. php pseudocode

A. PID

The PID [13] is the most common form of feedback. Its continuous formulation can be described by (8):

$$u(t) = K_p\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right) \quad (1)$$

The PID implemented in the computer follows the guidelines in [13].

B. Predictive control

Model Based Predictive Control (MBPC) ([14]) is a control strategy based on the explicit use of a model to predict the process output over a long-range time period. A receding control horizon technique is used: only the first control signal is applied (so all the changes that take place between two control signal calculations are considered). A cost function is minimized at each sampling time. It is usually defined as a

linear combination of the tracking error and the manipulated variables (the most common formulation being a quadratic cost function).

Although a non-linear model can be used for prediction, computational and robustness considerations makes the use of a linear model more adequate. Following the Generalized Predictive Controller (GPC) described in [14], a MIMO controller is described here. Considering a process with M inputs, N outputs and R measurable disturbances represented by the model:

$$A_i\left(q^{-1}\right)y_i(t) = \sum_{j=1}^{N} B_{ij}\left(q^{-1}\right)u_j(t) + \sum_{j=1}^{R} D_{ij}\left(q^{-1}\right)v_j(t) + \frac{T_i}{\Delta}\xi_i(t) \tag{2}$$

where: i =1, ..., M and A_i, B_{ij}, D_{ij} and T_i are polynomials in the q^{-1} operator, $\Delta=1-q^{-1}$ and ξ_i is white noise. The predicted future values of the controlled variables are:

$$\hat{y}_i(t+j) = \sum_{j=1}^{M}\sum_{k=1}^{j}\sigma_j g_{ij_k}\Delta u_j(t-k+j) + p_i(t+j), i = 1,\ldots,N \tag{3}$$

where g_{ij} is the step response between y_i and u_j and p_i is the free response of y_i. The coefficients σ and η take the value 1 if the corresponding input is included in the predicted output and 0 if it is not.

The control algorithm objective is the calculation of the sequence of (optimal) changes of the control variables in a control horizon Nu: $\Delta u_i(t+j)$, j=0,..., Nu_i-1 so that the predicted outputs $\hat{y}_i$ are as close as possible to the internal reference $r_i(t+j)$. This is translated into an optimization problem where a quadratic cost function of the tracking error and the manipulated variables is minimized, taking into account constraints on **Δu**, **u**, **y** and any other constrained variable that depends on **Δu**. This optimization problem can be stated as:

$$J = \sum_{i=1}^{N}\sum_{j=N1_i}^{N2_i}\eta_i\left(\gamma_i\left(\hat{y}_i(t+j) - r_i(t+j)\right)^2\right) + \sum_{i=1}^{M}\sum_{j=0}^{Nu_i-1}\sigma\left(\beta\Delta u_i(t+j)\right)^2 \tag{4}$$

$$D_{m_i} \le \Delta u_i(t+j) \le D_{M_i}, j = 0,\ldots,N_{u_i}-1$$

$$U_{m_i} \le u_i(t+j) = u_i(t-1) + \sum_{i=0}^{j}\Delta u_i(t+j) \le U_{M_i}$$

$$L_{m_i} \le \hat{y}_i(t+j) \le L_{M_i}, j = N3_i,\ldots,N4_i \tag{5}$$

where the coefficients γ and β give the relative weight of every prediction error or change in the control variables, while the coefficients σ and η taking the value 1 or 0 allow a variable to be included in the index or excluded from it. When no constraints are considered, there exists an explicit solution of (4).

VI. CONAPA

Previous sections have described the architecture and the internal description of the system. Now, the user's interface is presented.

The tool is open; everyone can use it, although there are sections that require a password (Figure 10). There is only one web server, so only one student at a time can change the parameters of the controller that calculates MC and KA. The data acquisition area is also protected because the real system can only be accessed one user at a time.

The following subsections will describe CONAPA's capabilities and basic software engineering:

a. Data acquisition.
b. Simulation of the web server.
c. Control of the web server online.
d. Comparison of simulation and real system.
e. Setting the benchmark traffic generator parameters.

A. Data acquisition

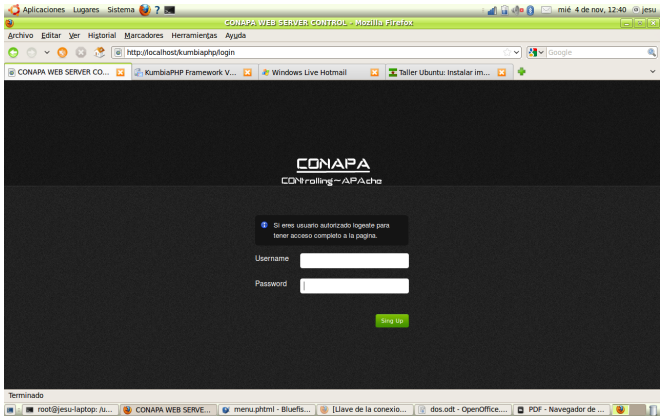

Figure 10. CONAPA's login screenshot

The user can collect data that allows the identification of a model that reflects how the real system works. As we have two inputs and two outputs, we keep constant one input and the other changes between the max and minimum value where the step duration decreases over time until 0 is reached. The procedure is repeated for the input that remained constant. This data can later be analyzed in other software such as Matlab. The period and max/min values are introduced by the user (Figure 11).

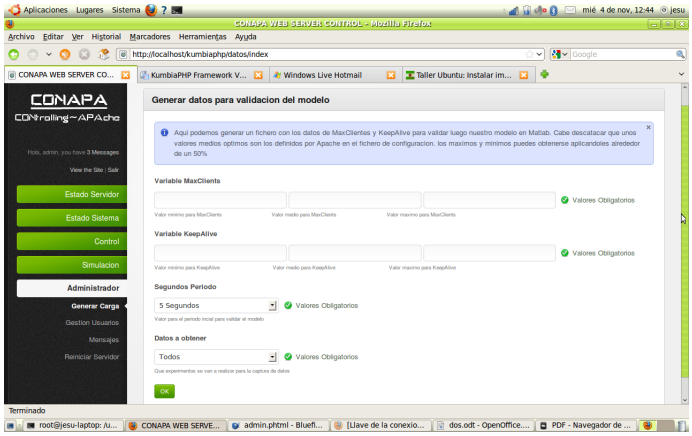

Figure 11. Setting parameters for data acquisition

Figure 12 shows the input variables (upper graphs) and the corresponding values for the outputs (lower graphs). A simple sequence diagram showing the main steps involved in the process is shown in Figure 13.

B. Web server's simulation

This area is open to all users. The type of controller can be selected and then the parameters (for instance, proportional, derivative and integral terms of the PID) should be entered. Each type of control has its own screen and parameters.

C. Web server's control

The available controllers are the same as in the previous sub-section. But, only one user can work with the real system. Each sampling time, KA, MC, MEM and CPU are read (using the socket described in Section IV) and sent to the controller.

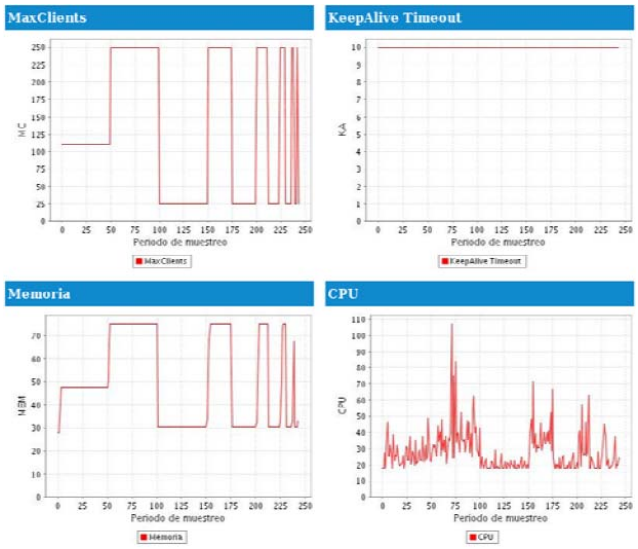

Figure 12. Data to export for identification

New values for MC and KA are calculated and sent to the server. Figure 14 shows the results of an experiment where the reference for the CPU was 5% and MEM was 40%. The controller was a PID.

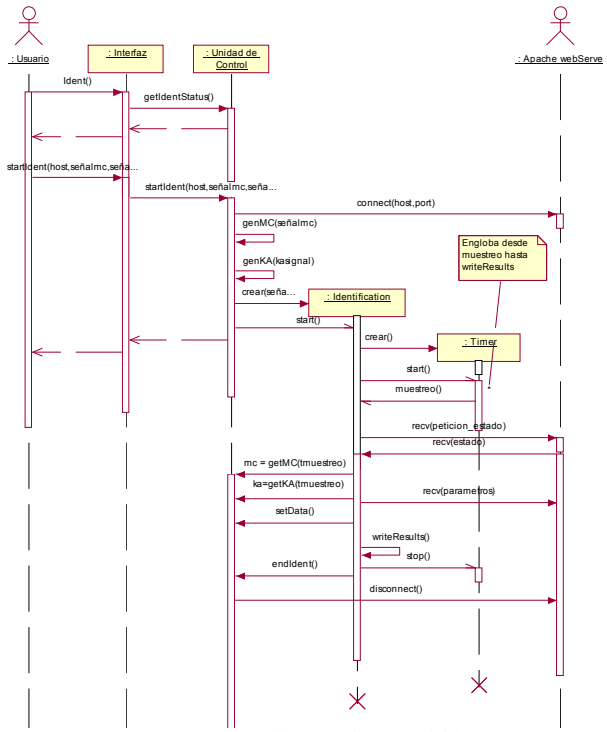

Figure 13. Sequence diagram, data acquisition

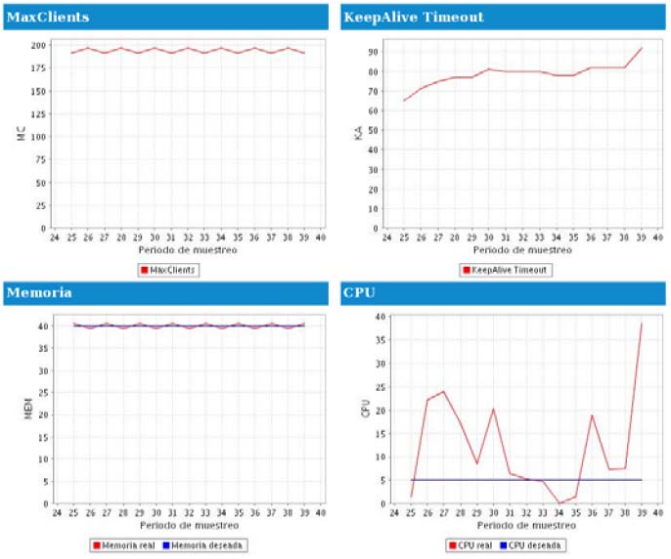

Figure 14. Real control

D. Comparison of simulated and real system

When this option is chosen, the real system is controlled and we apply the same tuning parameters to the simulation model. Thus, we compare whether the model and the real system have the same trends. This is quite useful if we are teaching our students the differences between simulation and the real world.

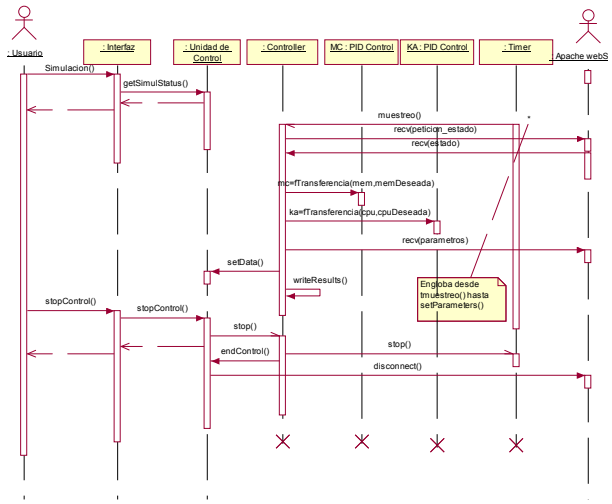

Figure 15. Simulation versus real system

Figure 15 shows the sequence diagram for this case when the controller is a PID.

E. Setting the benchmark traffic generator parameters

Different traffic conditions cause the behavior of the Apache web server to change. The tuning of the controller can be good for a traffic situation but not so adequate for another.

If we can define different scenarios, we can check the goodness of the controllers and the range they work in within acceptable performance.

This option is rather interesting for students because they can learn to think as a web administrator and deal with day to day situations. So, if they work administering networks in the future, they can be prepared to respond under pressure and changing network traffic loads.

VII. WHY IS CONAPA USEFUL?

It cannot be argued that there are many tools that present different components and characteristics of automatic control. There are virtual laboratories, online laboratories, add-ons to existing software: they are useful and have been developed for certain students and requirements.

CONAPA was built for students who follow courses where automatic control is not traditionally the main objective. But, if they are going to be good engineers they need a basic formation/knowledge and understanding of fundamental control ideas. If anyone studies Telecommunications or Computer Science, it is clear that they prefer networks and computers to a conventional mechanical system. It is at this point where problems arise. As stated in the introduction, most control books and applications deal with classic examples: electrical, mechanical, … So, we thought it would be interesting to present a system which these future engineers felt close to and that the tool could provide them with notions and knowledge in other valuable fields. CONAPA can be used for:

- Learning basic identification methods.
- Linear simulation.
- Controlling the Apache web server in simulation.
- Controlling the REAL Apache web server.
- Setting different traffic conditions and visualizing results: congestion, underutilization, overexploitation …

It is also useful for understanding Apache itself, how it works and how it can be modified to fulfill our necessities.

VIII. CONCLUSIONS

The paper has presented a tool that works both as a virtual and an online laboratory. The real system is a web server. Students can freely work with the simulation area, check their controllers and compare their tunings with the one that is being applied to the real system. The control of the web server is restricted because only one person at a time can change the parameters: we only have one plant and many potential users.

CONAPA is useful for those students whose degrees are not closely related to mechanical or traditional process control environments. They can see that automatic control can be applied to plants that have the same problems as these more conventional systems, but they feel more related to them.

The paper has given details on how to modify Apache's source code because there is little information about it and this was the hardest part of the work. In the future, we plan to migrate to a Windows operating system and to a higher Apache release.

REFERENCES

[1] J.L. Hellerstein, Y. Diao, S. Parekh and D.M. Tilbury, Feedback of computing systems. John Wiley & Sons, 2004.

[2] Apache Web server, home page: http://www.apache.org, as of 2009.

[3] Y. Diao, N. Gandhi, J.L. Hellerstein, S.Parekh and D.M. Tibury, "MIMO Control of an Apache Web Server: Modeling and Controller Design". American Control Conference 2002. Alaska, USA.

[4] R. Iakobashvili and M. Moser. "Welcome to curl-loader". http://curlloader.sourceforge.net/, as of 2009.

[5] D. Dougherty, "LAMP: The Open Source Web Platform", ONLamp, 2001.

[6] A. Goldberg and D. Robson, Smalltalk-80 The Language and its Implementation. Addison-Wesley, 1983.

[7] O.A. Dragoi, "The conceptual architecture of the Apache web server", web page http://www.cs.ucsb.edu/~tve/cs290i-sp01/papers/Concept_Apache_Arch.htm , as of 2009 (published 1999).

[8] K. Coar, "Writing modules for Apache", O'Reilly Open Source Conference, California, USA, 1999.

[9] P. Kamthan, "Apache web server customization", web page http://www.irt.org/articles/js180/index.htm, as of 2009 (published 1999).

[10] Python Web page, http://www.python.org/, as of 2009.

[11] R.S. Engelschall, "Apache 1.3. Dyanmic Shared Object (DSO)", web page: http://httpd.apache.org/docs/1.3/dso.html, as of 2009 (published 1998).

[12] http://www.cs.ucsb.edu/~tve/cs290iwi02/papers/Concept_Apache_Arch.htm

[13] K.J. Aström and T. Hägglund, *Advanced PID control*. ISA. NC, 2006.

[14] Clarke, D.W., Mohtadi, C., and Tuffs, P.S.: 'Generalised predictive control-Part I: The basic algorithm and Part II: Extensions and Interpretations', *Automatica*, 1987, 23, (2), pp. 137-160.