IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 24, 2019, accepted April 18, 2019, date of publication April 29, 2019, date of current version May 9, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2913531

Apache Dynamic Update for Feedback Control of Computing Resources

MARCOS V. S. BARRETO[1], WALTER BARRA, Jr.,[2], ERICK MELO ROCHA[2], FABRÍCIO G. NOGUEIRA[3], KEVIN LUCAS MARCILLO[4], RENAN LANDAU PAIVA DE MEDEIROS[5], THIAGO W. M. ABREU[6], AND MAYSA SOUSA ALVES[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Pará, Belém 66000000, Brazil
[2]Federal University of Pará (UFPA), Professional Campus, Belém 66075-110, Brazil
[3]Federal University of Ceará (UFC), Campus of Pici, Fortaleza 60020-181, Brazil
[4]Facultad de Ingeniería Eléctrica y Computación, Escuela Superior Politécnica del Litoral, Campus Gustavo, Guayaquil 66075110, Ecuador
[5]Federal University of Amazonas (UFAM), Campus of Pici, Fortaleza 69067-005, Brazil
[6]Université de Paris, 75006 Paris, France

Corresponding author: Marcos V. S. Barreto (marcos.sadala@bol.com.br).

ABSTRACT This paper shows an environment for performing data collection in the computational system that provides the response service to ''HTTP'' requests and makes interventions in the values of the available parameters in order to control the system automatically in the closed loop. The parameters to be obtained as endogenous variables of the computational system are MaxRequestWorks and KeppAliveTimeOut, and the exogenous ones will be the amount of memory consumed and the percentage of processor occupation time. The tool developed modifies neither the source code of the response software to the clients' ''HTTP'' requests nor the operating system code of the host computer, being non-intrusive regarding original system configuration. A proportional-integral (PI) controller was designed in order to keep the average perceived time response of ''HTTP'' messages regulated. The experimental tests carried out on an Apache Web server show the improvement obtained on the system's time response.

INDEX TERMS HTTP protocol, apache web server, digital controller.

I. INTRODUCTION

Automatic adaptation to dynamic changes in the behavior of a computer system is a subject that is increasingly gaining the interest of the academia and software and service providers. It is, therefore, intended to provide a reliable service to clients at low operating cost [1]. The environmental variables of the computational systems began to be studied over the past few years in order to optimize the performance of the services proposed by each software being developed [2], [18].

Since human behavior is very unpredictable and directly influences the computational environment, as the requests to be responded are based on clients' needs, computational environments feature uncertain dynamic behavior, thus requiring that frequent reconfigurations of the system concerned are made to try to deal with variations.

Software running concurrently can lead to increased latency time. For instance, when a virus scanner scans the operating system, it causes a response delay in other software to be run. This is a small scale of a single user making requests. Now, if a web server is considered that, depending on the popularity of the website, can address many competing requests, there is a critical problem to guarantee the quality of the services to be provided [17]. For example, Table 1 shows the number of access hits in a single day for some of the most visited electronic sites in the world [3]. The ''facebook.com'' website, for example, evenly distributing the amount of hits throughout the day, has a total of 9,684 concurrent users per second, which will have an average consumption load of at least 9,684 requests per second on its servers.

TABLE 1. Most visited sites of the world.

Eletronic Address	Amount of Hits
Facebook.com	836.7 million visitors
Google.com	782.8 million visitors
Youtube.com	721.9 million visitors
Yahoo.com	469.9 million visitors
Wikipedia.com	469.6 million visitors

The associate editor coordinating the review of this manuscript and approving it for publication was Aneel Rahim.

However, an important piece of news can drastically alter the behavior of the number of information requests from the website (e.g., release of the list of students approved to Brazilian federal universities). To deal with these large variations, resource optimization software can be used to increase system robustness [14]–[16].

The optimization software depends on the environment to provide the service. Studies have been carried out to identify the computational response system to ''HTTP'' requests [4] and to develop resource optimizers through the classical control methodology [5], but in different environments. However, the signals are equivalent in an older format, thus requiring improvements to the characteristics imposed by the environment being analyzed.

In [5], a slope is observed in the data collected from the open-loop computational system that will affect the system drastically at a given time, causing a ''Denial of Service – DoS'' because the controller is reducing the number of processes available through the Web Server to respond to ''HTTP'' requests.

Thus, a study of the existence of ''drift'' and of changes in the source code of optimizer software is pertinent, so that the PI controller can operate in the computational system without promoting a DoS. To that end, the results of a study aiming at improving computing system performance by using feedback control based techniques are presented herein. The main research focus has been to develop non-intrusive software tools to provide an environment that is suitable to perform advanced performance tests in computing systems. Specifically, the study focus aimed at improving the user's perceived response time in Apache server. In order to allow the results to be of immediate use by technology information (TI) managers, the investigated control structure has been restricted to Proportional-Integral (PI) controller, the most broadly used control structure in real-world industry applications.

The main scientific contributions of this paper include:

- To propose a non-intrusive software that changes Apache server settings without service interruption.
- To design a feedback controller that relates the amount of memory consumption with the maximum number of processes available to respond to ''HTTP'' requests.
- To develop software that prevents the ''HTTP'' response service from being caused by overload.

The remaining of this paper is organized as follows: Section II: the dynamic phenomenological model of the Apache computing system is presented; Section III: the proposed simulation environment is presented; Section IV demonstrates how optimization software was developed; Section V: the system identification is performed; Section VI: the PI controller is designed to regulate the system; Section VII: the experimental results are performed and discussed; Finally, Section VIII presents the conclusion of this study.

II. PHENOMENOLOGICAL MODELING OF THE ''HTTP'' REQUEST RESPONSE OF THE COMPUTATIONAL SYSTEM

The dynamic process of ''HTTP'' request response can be split into software and hardware dynamics, respectively. As a first approximation, dynamic aspects due to the hardware subsystem can be ignored due to its almost instantaneous time response. In contrast, the dynamics of the software subsystem has a dominant effect on the time response perceived by the user. This is due to the intensive use of queuing structures upon implementation in the web server, leading to a considerable impact on the system's overall time response perceived.

By using a standard queue notation, the computational system is herein modeled by using M/M/1 queues, where ''M/M'' means that incoming requests and service time rate both conform to memoryless markovian exponential probability distributions. Number ''1'' indicates the number of simultaneous service channels, i.e. the number of processors being used for the system under analysis.

The amount of buffers or system capacity, population size and service discipline were suppressed from the queue notation because the default values were adopted, namely: infinite, infinite and FIFO (the first request incoming will be the first to be processed).

In [1], a queue scheme of a computational system of response to ''HTTP'' requests is proposed, which splits the queue into two parts: internal and external. The internal queue represents the software stage that provides the response service to requests. It has a size limit that depends on the value of ''MaxClient'' and has an occupation rate of value λ_i. The external queue has an infinite size because it is made up of multiple system queues (e.g., network connection queue, file system queue etc.) and will be occupied at a rate λ.

According to Tipper's model [1], which equates the dynamic flow of a queue, and the previously-mentioned characteristics of the queue, we have the following for the steady state:

$$\frac{d}{dt}n(t) = \lambda(t) - \mu\frac{n(t)}{1+n(t)} \tag{1}$$

where, n is total number of requests in the computational system, at time instant t and μ is the request processing rate.

Considering Little's law and the Tipper model of the average time response, d, as well as the average number of jobs inside the web server, n_i, are given respectively by:

$$\frac{d}{dt}d(t) = -\frac{1}{T_d}d(t) + \frac{1}{T_d}\frac{n(t)}{\lambda(t)} \tag{2}$$

$$n_i(t) = n(t)\left[1 - \left(\frac{n(t)}{1+n(t)}\right)^{M_c}\right] \tag{3}$$

where, T_d is the sampling time and M_c is the value of the Apaches's ''MaxClient'' parameter, at the given operating point. To obtain the number of external requests, n_e may be

obtained by using a conservation in the form presented in (4).

$$n_e(t) = \int_0^t (\lambda(\tau) - \lambda_i(\tau))d\tau \tag{4}$$

where, the incoming rate of requests in the web server, λ_i, is obtained by using the difference between the total number of requests inside the system and the number of requests inside the web server.

$$\lambda_i(t) = \lambda(t) - \frac{d}{dt}\{n(t) - n_i(t)\} \tag{5}$$

Finally, the perceived time response is estimated by using low-pass filtering as follows

$$\frac{d}{dt}d_i(t) = -\frac{1}{T_d}d_i(t) + \frac{1}{T_d}\frac{n_i(t)}{\lambda_i(t)} \tag{6}$$

The set of differential-algebraic equations (1)-(6) provides a phenomenological, nonlinear, dynamic model for the web server system.

To relate the amount of memory used in the computational system with the number of processes in the queue of the response service to ''HTTP'' requests, it is necessary to know the amount of memory spent by each process k-*th* process, which can be represented by (7)

$$Mem = K.n_i \tag{7}$$

where the value of proportionality constant K must be estimated by tests. For the tests discussed herein, the value of *Mem* has been estimated as 0.3% out of an approximate total of 2.0 Gbytes, extracted from actual tests.

In order to gain an insight of the system behavior for different operating points, the computational system (the set of equations (1)-(6), presented above) has been used to develop simulation in MatLab/Simulink computational environment (see Fig.1). Figure 1 exhibits a block diagram developed for reproducing the behavior of the investigated system by using three levels of loading, namely $\lambda = \{50, 100, 150$, with time intervals in the ranges {(0,300), (300,600), (600,1000)}, respectively. Processor use consumption was around 28%, for the ''HTTP'' server, and sampling time was set to the value of 1.0 second.

In the ''MaxClient'', parameter value was set to 50 concurrent processes and changed in step variation of amplitude 100 over a period of 300 seconds, with duty cycle of 50% (see Fig.2).

The time response of the Apache server will depend on the number of requests that are waiting to be processed. The response time starts the counting when the process is already in the Apache server list and not in the external list in the operating system. Fig.3 shows the time response variation with parameter changes over time.

Up to 300 seconds, the ''MaxClient'' variation does not interfere, as there are few processes in the queues, that is, the request rate is less than the processing rate and the response time remains constant. In the thirtieth second there is an increase of load in the computational system, thus getting the

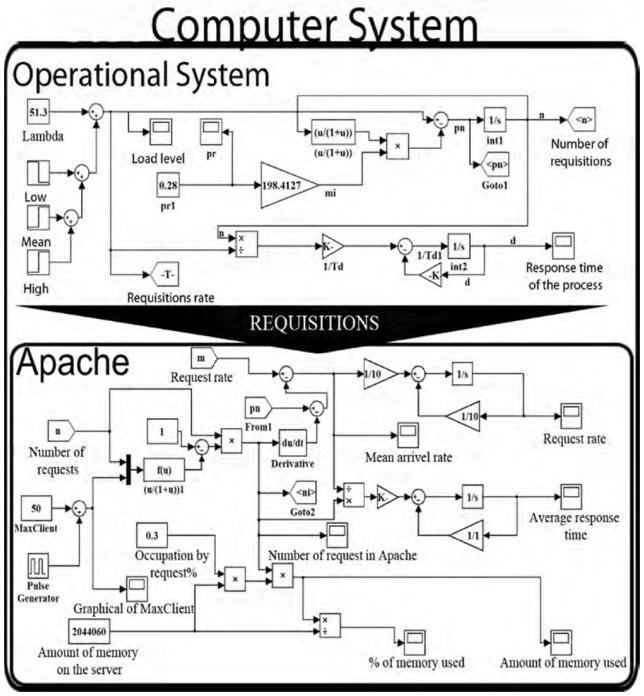

FIGURE 1. Phenomenological model simulator developed.

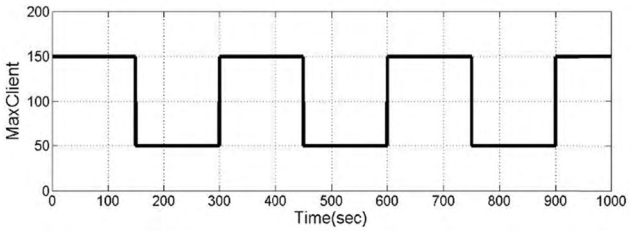

FIGURE 2. "MaxClient" variation.

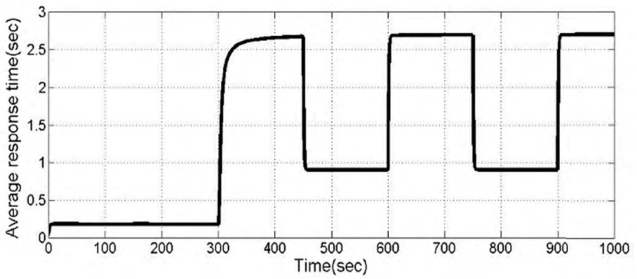

FIGURE 3. Average response time for requests in "HTTP" server.

Apache server response time to increase and having variations with the same behavior as that of ''MaxClient'', since internal server queue is occupied by processes waiting to be run.

The increase in load causes an increase in the number of processes in the internal queue (web server), Fig.4, and in the external queue (operating system). Since it was defined that the external queue would be infinite, the simulation would

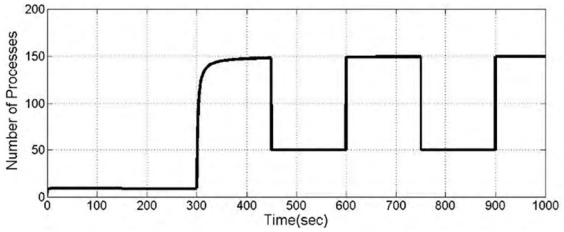

FIGURE 4. Number of Processes in "HTTP" Server.

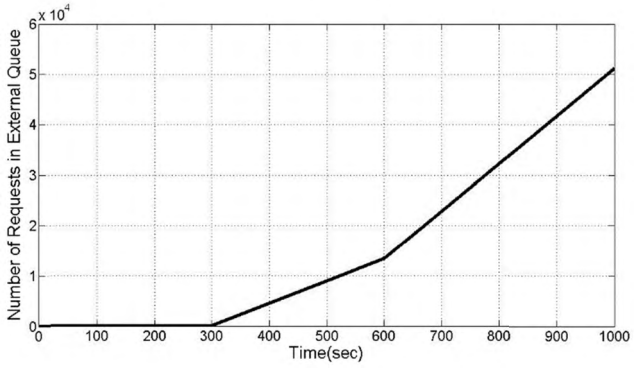

FIGURE 5. Number of requests in the external queue.

not make limitations. However, the angle of inclination of the line follows the request rate, Fig.5.

The amount of memory the Apache server will be using is that in the internal queue. So, the amount of memory used in the external queue is not being added to the simulation results. In the 0-to-300-second interval, there is a constant use of the memory, even if there is variation of "MaxClient.

The occurrence is explained by the fact that there are not enough process numbers to be inserted in the internal queue due to the condition $\lambda < \mu$. The interval between 300 to 1000 seconds, the request rate increases, thus increasing the process number. For $\lambda = 100$ and in the 300-to-450-second interval there is a system accommodation time due to slope in the growth line of the number of processes.

For $\lambda = 150$, as from 600 seconds, the system will no longer have this behavior because it already has enough processes in the external queue to occupy the internal queue, thus causing the occupied memory percentage variation to stabilize. Refer to Fig.6.

III. "HTTP" REQUEST RESPONSE COMPUTATIONAL SYSTEM

The management of a computational system's hardware is performed by the operating system through abstractions of the codes in Machine Language, called logical resources. The logical resources under study herein are the processor and the memory.

Techniques such as multiprogramming (by processes) and virtual addressing are examples of codes that are running in

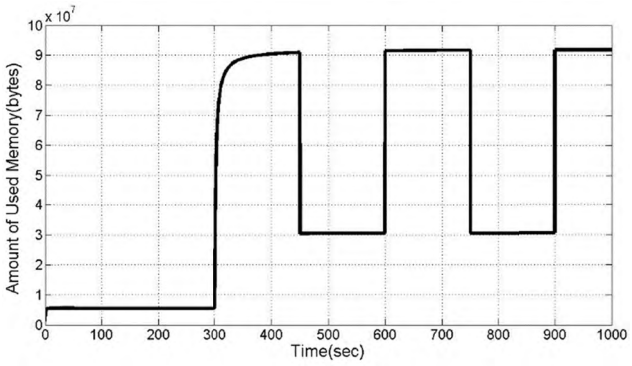

FIGURE 6. The amount of memory used by the "HTTP" server.

the machine in the operating system's software to provide the user with availability of resources.

Multiprogramming occurs when a processor is multiplexed, i.e., the processor is made available to the various processes in the host machine, whether required by the operating system or by the user. The "HTTP" request response server under analysis herein uses several multiprogramming methods. But, the multiprocessing module called "prefork", which uses processes to respond to clients, will be used herein. Fig.7 shows an architecture of the processes for the Apache server.

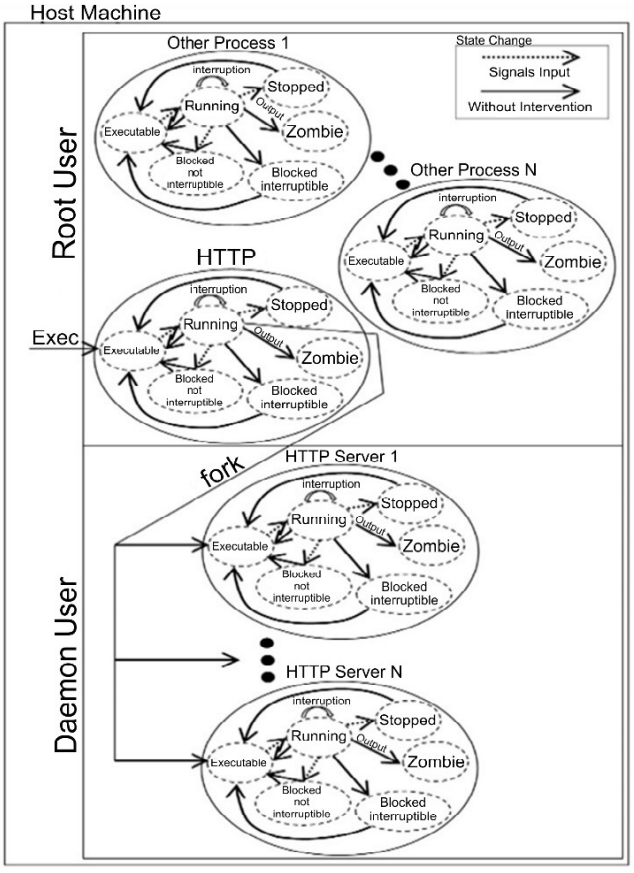

FIGURE 7. Apache server architecture in the operating system viewpoint.

There are three process states for the ''HTTP'' server: busy, standby, and inactive. The busy state is when the process is being used by a client. The standby state, a value stored in the ''KeepAliveTimeOut'' variable, is when the process is waiting for a request from the client. The inactive state is when the process ceases to have an owner and is on stand-by for a request from a new client. The ''MaxClient'' variable is the stored value of how many processes can be in place concurrently [6].

From the viewpoint of the operating system, the architecture for the server processes is more complex, since ''prefork'' uses the so-called ''fork'', which creates the clone (child) processes from a parent process. Each new server process has attributes, such as the Process Identifier - PID, which serves to identify, contextualize (priority, security, and environment) and run the planned task [7].

The parental relationship between the processes is translated by the information that is kept by the core, so that it can manage the parent process and the child processes jointly. For some operating systems, death of the parent process does not mean death of the child process; e.g., the Unix operating system, running until the end of the code or until sending a signal to its death.

The order at which the processes are run in the processor is not deterministic due to the process-switching base mechanism, scheduler, relying on numerous factors, such as ''time-slice'' or ''quantum'' and preemption [8].

There is no way to predict what the order of running the processes will be, so the same program will be run in many different ways.

Process security is based on its isolation from the other processes by confining them to limited memory address, limited actions by the privileges of its owner, and controlled interaction through calls to the operating system. Thus, for a process to interact with another, a call must be made to the operating system, which will check the permissions and the execution of the call.

Fig.7 shows the Apache server architecture in the design of the operating system for the management of existing processes. The only connection between the processes is the values of the process variables that are inherited from the parent process owned by the ''root'' user, which can interfere with the execution of the child process only by the operating system calls - the signals. The user named ''Daemon'' is responsible for providing for the services in the host machine, that is, it will respond to the client making a request.

```
root@ServerApache:/usr/local/httpd-2.4.6/bin# sudo lsof -i:6405
sudo: unable to resolve host ServerApache
COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
httpd   4117   root    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4118 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4119 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4120 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4121 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4122 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
root@ServerApache:/usr/local/httpd-2.4.6/bin#
```

FIGURE 8. Processes listening to port 6405.

Fig.8 shows an Apache server started with 6 processes to respond to requests, variable ''StartServer'' = 6, and a parent process, all of which listening to the communication port 6405 of the host machine. Fig.9 confirms the heredity of the processes.

```
root@ServerApache:/usr/local/httpd-2.4.6/bin# pstree -p 4117
httpd(4117)─┬─httpd(4118)
            ├─httpd(4119)
            ├─httpd(4120)
            ├─httpd(4121)
            └─httpd(4122)
root@ServerApache:/usr/local/httpd-2.4.6/bin#
```

FIGURE 9. Heredity amongst the httpd processes.

The ''httpd'' process of the ''root'' is responsible for managing the ''Daemon'' processes. Thus, the creation and death of the processes are governed by parameters that are stored upon server startup, process creation of the ''httpd'' process of the ''root'', and are used for determining whether a process should be killed or generated.

In case of death of the ''httpd'' process of the ''root'', the ''HTTP'' request response service will remain active. However, the management of the processes will be interrupted, not performing the task of generation or death of the ''httpd'' processes of the ''Daemon''. Fig.10 shows the ''http'' response service by listening to port 6405 without the ''httpd'' process of ''root''.

```
root@ServerApache:/usr/local/httpd-2.4.6/bin# sudo lsof -i:6405
sudo: unable to resolve host ServerApache
COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
httpd   4118 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4119 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4120 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4121 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4122 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
root@ServerApache:/usr/local/httpd-2.4.6/bin#
```

FIGURE 10. ''httpd'' response service active without the parent process.

The ''Daemon'' user processes were eternally active, waiting for client requests, without increasing or decreasing the number of processes, that is, the computational resources will have a linear consumption with a regression coefficient equal to 0.

IV. RESOURCE OPTIMIZATION SOFTWARE FOR THE ''HTTP'' REQUEST RESPONSE SERVICE

The computational resources of the host machine are limited, in a way that a resource allocation methodology must be implemented to support the service in operation, thus minimizing failures at the different load operation points (number of ''http'' requests).

A tuned feedback controller is a law, an equation which herein relates the computational resources to the variables of the Apache process manager, which uses them to determine the generation or death of a ''Daemon'' user process. The variable values are stored in the file ''. . . \ conf \ extra \ httpd-mpm.conf'' encoded in accordance with the ASCII table.

When starting the service with the command ''./.../bin/httpd –k start'', the variables are loaded into the part of the main memory to be used by the ''httpd'' process of the root user. The values being stored are used as resource utilization

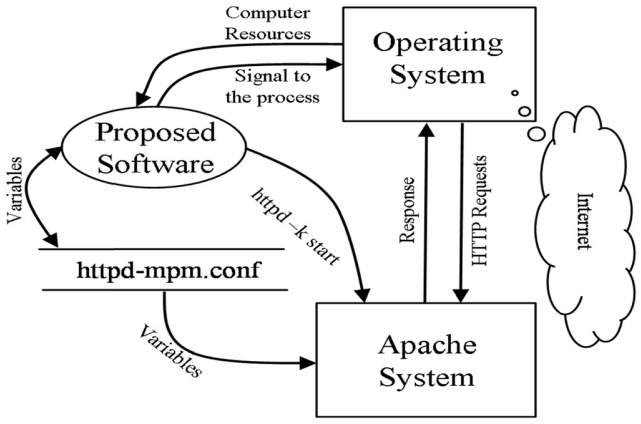

FIGURE 11. Data flow diagram.

limiters for the process manager of the "http" request response server.

The proposed software is a process manager that works on a layer above the process manager of the Apache server, thus promoting dynamics in the values described in the file "httpd-mpm.conf" in Apache 2.4.6 without interrupting the service. Fig.11 shows an overall flow diagram of the software.

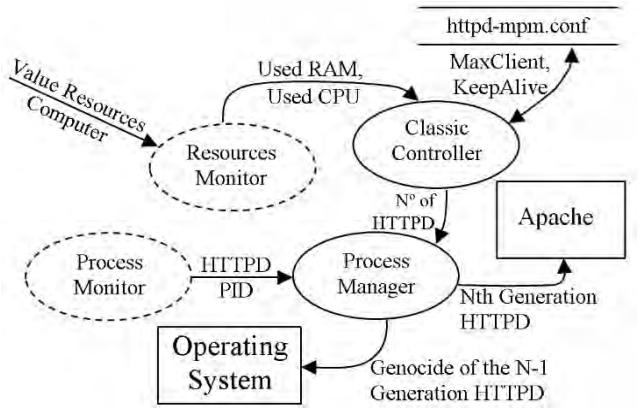

FIGURE 12. Expanded data flow diagram.

In Fig.12, the proposed software is expanded, showing how the data entering the process is processed by sending signals to the operating system, execution commands and information storage request.

```
root@ServerApache:/usr/local/httpd-2.4.6# free
             total       used       free     shared    buffers     cached
Mem:       2044060    1339436     704624          0     240480     427740
-/+ buffers/cache:     671216    1372844
Swap:      2094076          0    2094076
```

FIGURE 13. Expanded data flow diagram.

The "resource monitor" process is responsible for collecting the data from the computational resources being used. It is created from the "fork" function and processing thereof is performed in parallel. The method used for obtaining the data is the "free" (Fig.13) with change in the standard output to a buffer as a communication channel of the PIPE type. The value used to count the memory being used will be the value "used" – "buffers" – "cached", thus eliminating the behavior "drift" from the computational system.

The "classic controller" process does not run in "background", a mode at which the start and end of the scheduled routine is interfered by the user. For the test, a SISO system was used that had an output variable of the amount of memory used and with "MaxClient" input variable. The main flowchart of the "classical controller" process is shown in Fig.14, "R0" and "R1" being the parameters of the feedback controller to be tuned, "Ref" being the reference value defined by the administrator of the computational system, PU_Used_PU per memory unit used.

```
Used_PU= (float)UsedMemory/TotalMemory;
Yk = Used_PU;
 Ek_1 = Ek;
Ek = Ref - Yk;
 Uk_1 = Uk;
Uk = Uk_1 + floor(R0*Ek) + floor(R1*Ek_1);
MaxRequestWorkers = 100 + Uk;
```

FIGURE 14. Main part of the "classical controller" process.

The "MaxRequestWorks" parameter is the same as the "MaxClient" parameter. The name was changed in the Apache version used for software testing.

The initial state considered was equal to 100 and the other variables are intended for storing the system states.

The "process monitor" process relates the existing "httpd" processes with process generation, i.e. the parameters used for creating it, "MaxClient" and "KeepAliveTimeOut". The relationship is stored in a row with registry structure having the "PID" as the primary key.

The "process manager" process indicates and executes which "httpd" process of the "Daemon" user should be killed and the generation of new processes with the new configurations, based on the information sent by the "classical controller" process and the process state.

```
root@ServerApache:/usr/local/httpd-2.4.6/bin# ps aux | grep -i httpd
daemon    4118  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
daemon    4119  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
daemon    4120  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
daemon    4121  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
daemon    4122  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
root      4168  0.0  0.2 106644  5636 ?        Ss   10:33   0:00 ./httpd -k start
daemon    4169  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
daemon    4170  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
daemon    4171  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
daemon    4172  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
daemon    4173  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
root      4175  0.0  0.0  13628   964 pts/0    S+   10:33   0:00 grep --color=auto -i httpd
root@ServerApache:/usr/local/httpd-2.4.6/bin#
```

FIGURE 15. Two generations of "httpd" process.

Fig.15 shows the first action of the "process manager" process. The "PID" processes 4118, 4119, 4120, 4121 and 4122 are of the previous generation, that is, parameters no longer suitable for the condition of the host machine. The "PID" processes 4168, 4169, 4170, 4171, 44172 and 4173

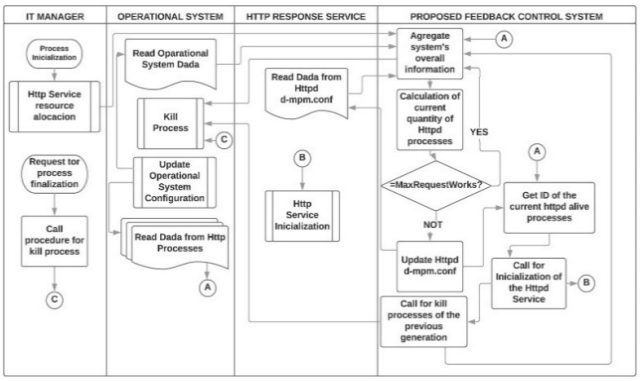

FIGURE 16. Flowchart showing the interactions between the proposed control system and its software environment.

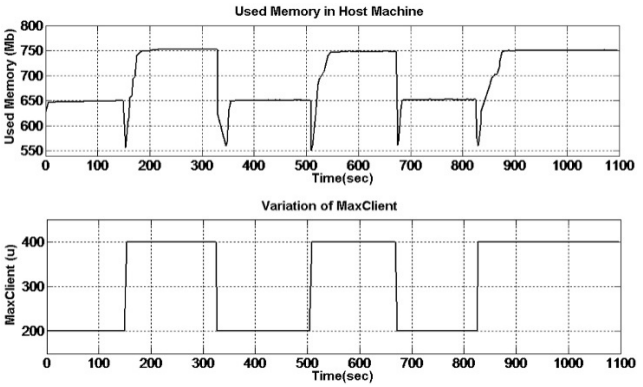

FIGURE 18. Variation and response of the "httpd" system.

are of the new generation with the parameters already set. The process that has the "owner" equal to "root" will manage the new generation in the "Apache" methodology until a new intervention of the proposed software.

In order to emphasize the interactions between the proposed control system and the underlying software environment, namely the operating system and the "HTTP" service, Fig.16 shows the corresponding process flowchart. As can be seen, the PID digital controller is executed as a task belonging to the process called "Calculation of current quantity of "HTTP" process". If a disturbance event is detected by a supervisory task, the proposed control system performs the online adjustment of the settings of the response server "HTTP".

V. IDENTIFICATION OF THE "HTTP" REQUEST RESPONSE COMPUTATIONAL SYSTEM

Mathematical modeling provides ways to develop and implement an equation that has the same behavior as that of the desired characteristics of an actual system. Thus, it is a safe way to prepare a model that is analogous to the intended characteristics. Fig.17 shows the scheme for identification thereof.

FIGURE 17. Schematics for identification.

To obtain the data for a non-parametric open-loop identification, it was necessary to develop software that varies the values of the input signal "MaxClient" and captures their effects in memory. Fig.18 shows the disturbance inserted in the system and respective response thereof.

To excite plant dynamics in order to estimate a parametric model, a pseudo-random binary sequence (PRBS) type signal was designed. The following criteria were used to choose the parameters [9]:

1. For the amplitude values of the PRBS signal, it is limited by the maximum excursion allowed to the process excitation signal, causing no interruption in the "HTTP".
2. The signal period should not be less than the system's accommodation time, otherwise it reduces the randomness character of the test.
3. The interval between the bits must be close to the shortest system time divided by 3, since the system is nonlinear.

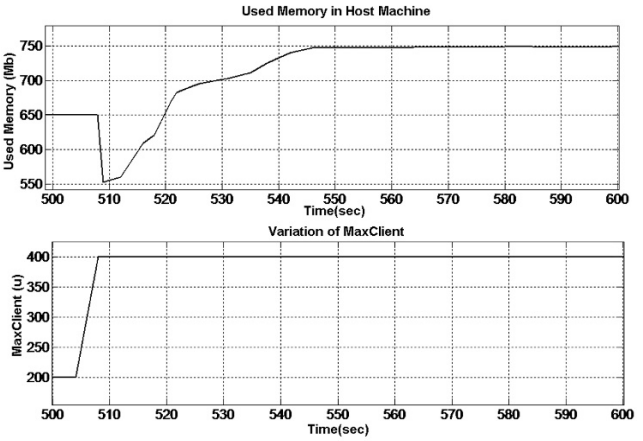

FIGURE 19. Expansion of the 4th change.

It can be seen in Fig.18 that the longest times for system accommodation are seen in the third and fifth "MaxClient" changes (from 200 to 400 processes). Fig.19 and Fig.20 are the respective extensions of the aforementioned changes to the removal of the longer system accommodation time.

The 55-second time is the longest system accommodation time which will be adopted to project the PRBS signal. For the shortest time, the 24-second time will be adopted to guarantee system response. For the amplitude value, the 200-process operating point, varying by 200 processes upwards, will be adopted, since there were no impediments in the open-loop system variation tests. Fig.21 shows the designed PRBS signal.

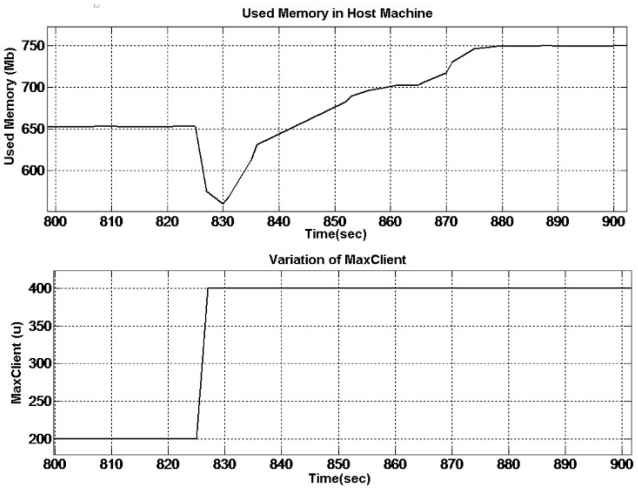

FIGURE 20. Expansion of the 5th change.

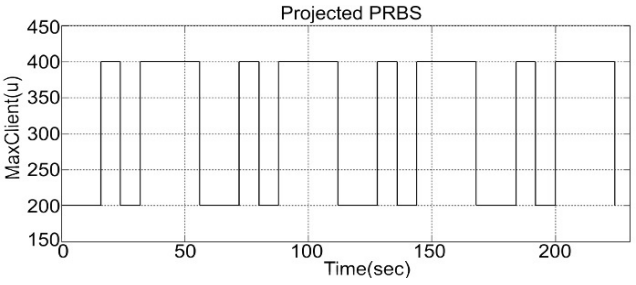

FIGURE 21. Projected PRBS signal.

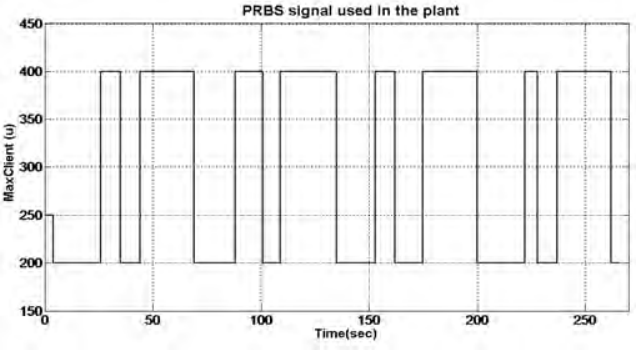

FIGURE 22. PRBS signal used in the plant.

Thus, the PRBS got an 8-second time between bits and a number of bits equal to 3, totaling the period time of 56 seconds.

Fig.22 shows the pseudo-random binary signal (conserved characteristics of the projected PRBS signal) obtained from switching the "MaxClient" between two values. Initially, the plant was in steady state, "MaxClient" equal to 250, and from the fifth second the PRBS signal is inserted.

The system's response is shown in Fig.23. It can be noted that, even for the broader levels of the PRBS signal, the system does not reach the steady state.

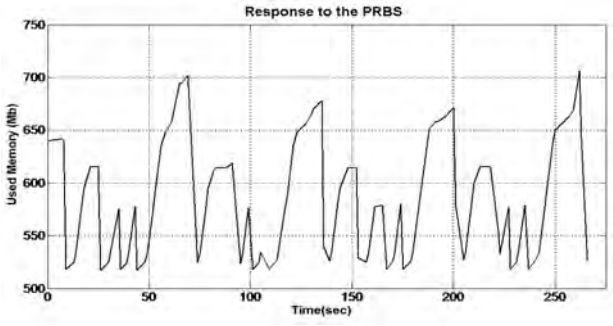

FIGURE 23. Response to the projected PRBS.

By using the non-recursive least squares algorithm, the digital transfer equation (8) was reached, which has a behavior similar to that of the plant being studied.

$$H(q^{-1}) = \frac{0,1521q^{-1}}{1 - 0,6732q^{-1}} \tag{8}$$

where q-1 is the backward-shift operator.

One of the validation techniques of the system domain data is the autocorrelation function and the cross correlation function, i.e., for the system model to be valid, the estimated residuals should be approximately uncorrelated. The standard deviation of 7 was used in the development of Fig.24, since the average variation of plus or minus 7 does not have a disturbing influence on the plant.

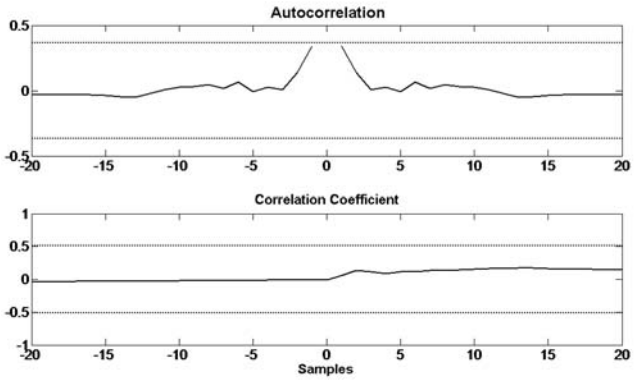

FIGURE 24. Autocorrelation and cross-correlation coefficient.

The autocorrelation coefficient informs how much the execution of a given random variable influences its neighbors. For example, if the occurrence of one variable is considered high, how much will this influence in having the next value also high? In Fig.24, it is observed that the residual value of autocorrelation close to the instant 0 is approximately a unitary pulse (not being represented in the chart, as its amplitude modifies the proportionality of the figure) and is practically null for the other delays. It can be concluded that the residual signal has a near-white frequency spectrum.

The cross-correlation coefficient is a measure of similarity between two signals due to a delay applied to one of them. Fig.24 shows the cross-correlation between the estimated

residuals and the time series. It can be seen that the residual line does not exceed the safety margin, which guarantees the near-white frequency spectrum characteristics.

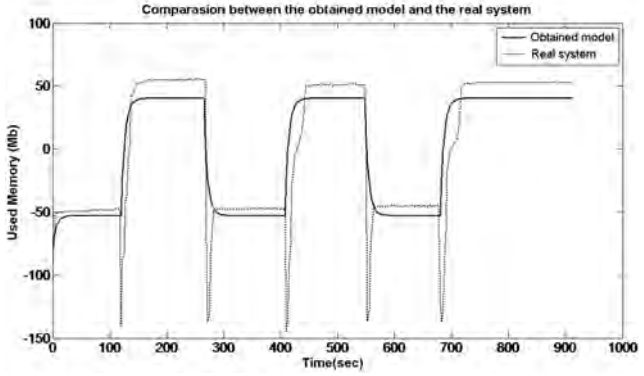

FIGURE 25. Comparison between obtained model and the real system.

Fig.25 shows the comparison between the actual model and the obtained model. The best adjustment percentage was quantified by 52.08. The validation data are from Fig.18, the averages thereof were removed.

VI. PI CONTROLLER DESIGN

The controller designed for the "HTTP" request response computational system features a PI control action adjusted by the pole placement method.

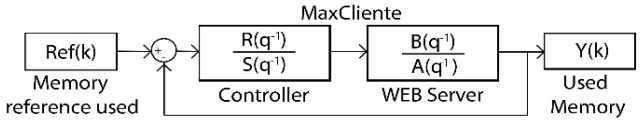

FIGURE 26. Feedback Controller Diagram.

Fig.26 shows the block diagram of the implemented digital controller. The controller's discrete transfer function is defined as [10]:

$$\frac{R(q^{-1})}{S(q^{-1})} = \frac{r_0 + r_1 q^{-1}}{1 - q^{-1}}. \tag{9}$$

The controller design method aims to find the values of r_0 and r_1. Since both controller and plant model are 1st order discrete transfer functions, the resulting closed-loop characteristic polynomial has the form:

$$A_{MF} = z^2 + (a_1 - 1 + b_0 r_0)z + (b_0 r_1 - a_1) \tag{10}$$

Therefore, in order to solve the pole placement problem, a 2nd order desired polynomial should be specified as:

$$A_{MF}^{des} = z^2 + a_1^{des} z + a_2^{des} \tag{11}$$

By equating (10) and (11), one obtains

$$\begin{bmatrix} b_0 & 0 \\ 0 & b_0 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = \begin{bmatrix} a_1^{des} + 1 - a_1 \\ a_2^{des} + a_1 \end{bmatrix} \tag{12}$$

To obtain the values of a_1^{des} and a_2^{des} , the knowledge of the plant is used in order to impose the desired behavior. The settling time (ts) is equal to 55s and by using an excess percentage of5% ($\xi = 0,7$), having [9]:

$$ts = \frac{4}{\xi w_n} = \frac{4}{0,7 * 55} = w_n = 0,1039_{rad/s} \tag{13}$$

However, the system of this work is discreet. To transform a_1^{des} and a_2^{des} into a discrete time, the following equations are used [11].

$$a_1^{des} = -2e^{-\xi w_n T_s} \cos(\sqrt{1 - \xi^2} w_n T_s) \tag{14}$$

$$a_2^{des} = e^{-2w_n T_s} \tag{15}$$

By applying the values in (14) and (15) with Ts = 8s, we obtain $a_1^{des} = -0,9266$ and $a_2^{des} = 0.1897$. By performing mathematical operations in (12):

$$\begin{bmatrix} 0,1521 & 0 \\ 0 & 0,1521 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = \begin{bmatrix} -0,9266 + 1 - (-0,6732) \\ 0.1897 + (-0,6732) \end{bmatrix}$$

Thus, the values of r0 = 4,9088 and r1 = −3,1789. The transfer equation of the tuned PI controller is seen in (16):

$$\frac{R(q^{-1})}{S(q^{-1})} = \frac{4,9088 - 3,1789 q^{-1}}{1 - q^{-1}} \tag{16}$$

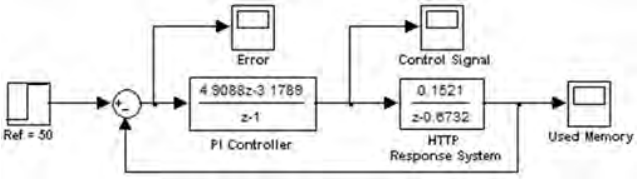

FIGURE 27. Closed-loop system simulation scheme.

Fig.27 illustrates the scheme of the closed loop simulation.

Fig.28 shows the memory used by the "HTTP" request response system. The settling time is observed to be in accordance with the plant and the overload in the established criteria.

Fig.29 and Fig.30 are, respectively, the error and control signal charts. Due to the use of the integrative portion of the PID controller, the system error was rejected as of the eighteth second; however, in the plant's settling time, the error is very close to zero, no longer interfering with the plant in the real system because a process unit of the variable "MaxClient" has a fixed size depending on the web page being requested.

The control signal does not have high relative amplitude, and it can be stated that the control effort is not high, being stabilized before the plant's settling time.

VII. OBTAINED RESULTS

By using the designed PI controller in the developed software, the system response (Fig.31) was obtained for a reference of 562 megabytes of used memory. Initially, the "HTTP" server had a process to respond to customer requests

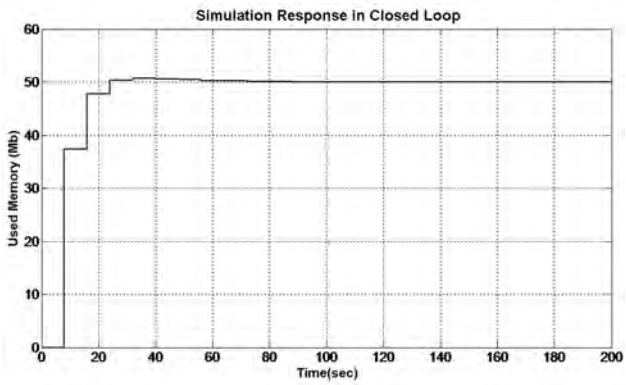

FIGURE 28. Simulation Response.

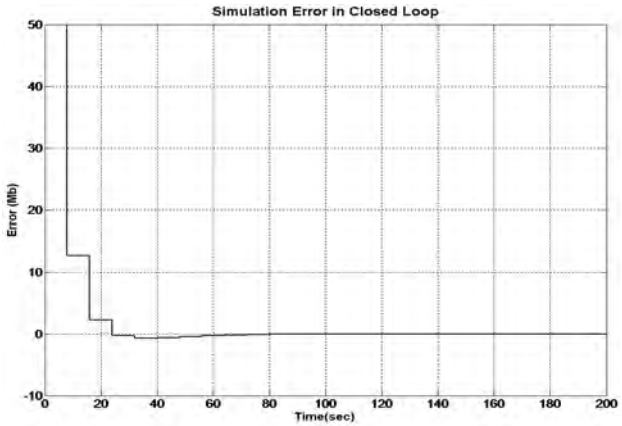

FIGURE 29. Closed-loop simulation error signal.

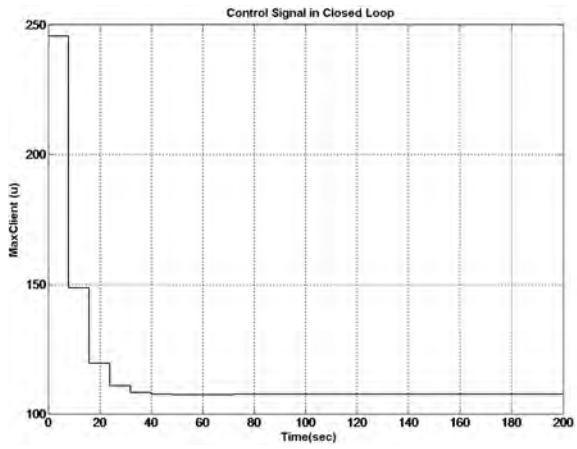

FIGURE 30. Closed-loop simulation control signal.

(time interval from 0 to about 50 seconds). As from the 50-second instant, controller action was started. It can be seen from Fig.30 that the system achieved stability approximately 55 seconds after the beginning of controller actuation and the error is less than 1% of the reference value and can be considered null [9].

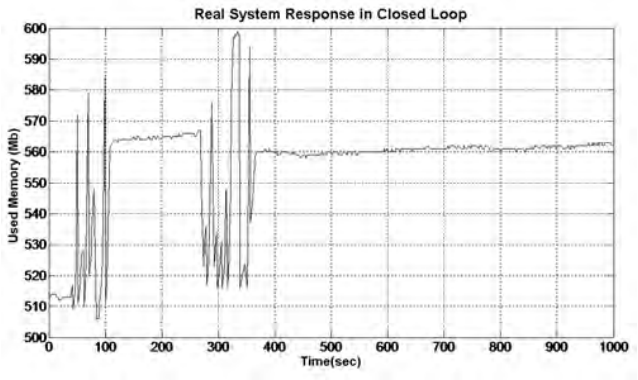

FIGURE 31. Actual closed-loop system response.

Approximately at instant 270 seconds, a disturbance was caused, the opening of a software. The controller stabilized the system approximately 100 seconds after the disturbance started. Stabilization delay is caused by the length of time the software takes to fully load the memory. Stabilization has an error of less than 1% of the reference value and is considered null.

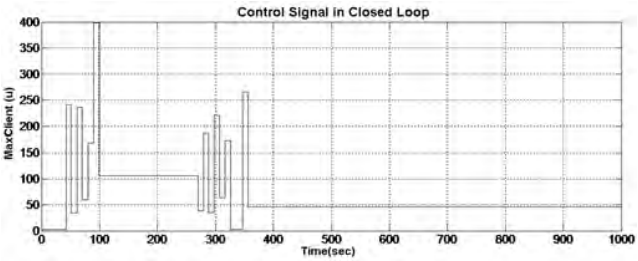

FIGURE 32. Closed-loop control signal.

Fig.32 shows the control signal that is used in the plant. At the first moment of controller operation, period of 50-270 seconds, the value of ''MaxClient'' was stabilized in approximately 110 process units; in the second moment of controller performance, period of 270-1000 seconds, the value of ''MaxClient'' was stabilized in approximately 50 process units. The reduction of processes to respond to ''HTTP'' requests is caused by the memory being occupied by an open software, thus forcing the controller to act negatively on the ''MaxClient''

The tests were run in an environment using one physical machine (Windows 7) and two virtual machines (Virtualization with VMware version 6.0.7 build-2844087). The first virtual machine hosting the ''HTTP'' request response service (Apache version 2.4.6) and the second with the ''HTTP'' request simulation system to generate the load on the system (command:./httperf –hog - -server = 172.16.15.15 –num-conns 720,000 –num-calls 100 –rate 600), with 600 requests per second, each request will make 100 calls to the web server addressed in the Internet Protocol IP 172.16.15.15, adding up to a total of approximately 1200 seconds of load.

Table 2 shows the configuration of the machines used in the tests performed herein.

TABLE 2. Testing machine configuration.

Machine	Type	Available Memory	Processor Type	Available Cores
Base System	Physical	6 GB	Intel Core I7	8
Apache Server	Virtual	2 GB	Intel Core I7	1
Load Generator	Virtual	1 GB	Intel Core I7	1

VIII. CONCLUSION

The purpose of this article, to develop non-intrusive software to the ''HTTP'' request server, has been achieved, thus enabling the IT manager to scale the maximum amount the server can use. Thus, a problem of lack of resource, main memory, for the basic routines of the operating system will be automatically controlled by the person in charge.

However, for systems that use virtualized memory, the problem of lack of main memory resources practically does not exist, since there is an extension of the main memory to the secondary memory, there being no limit to the use of memory, with a significant increase in the response time by using mechanical storage medium, a hard disk. By controlling the use of memory, one can avoid the use of virtual memory by providing a control of the server response time.

In the identification of the system, it is noticed that, depending on the state of the operating system, there is a variation in the plant's settling time. This variation in time is the result of the number of processes existing in the queue of the operating system that hosts the ''HTTP'' request response server.

The non-minimal phase of the system is caused by the running of the management of the ''httpd'' processes of generation n-1 in batch form. Generation n processes are created according to the number of requests (load) in the TCP port 80 queue of the operating system. Then, there is a steep release and occupation, which requires time, in memory.

Since this is a non-linear system, the PI controller design will succeed at the point of operation in which it was designed, that is, in the range of plus or minus 50 in amplitude and with the environment being controlled (verification of running processes).

For the operation point, the results obtained are satisfactory, since the plant output does not exceed 1% of the reference value. Although the designed standard digital PI controller has succeeded in the experimental tests reported herein, strong nonlinearities as well as time-varying system behavior may demand the investigation of more advanced control techniques, such as adaptive and/or robust control approaches. These particular issues are to be investigated and reported in future papers. Another point for future research is the influence of virtualization on the results and how other services hosted on the same machine influence each other and how the controller behaves.

REFERENCES

[1] M. A. Kjær, ''Disturbance rejection and control in Web servers,'' Ph.D. dissertation, Dept. Autom. Control, Lund Univ., Lund, Sweden, 2009.

[2] N. Gandhi, D. M. Tilbury, Y. Diao, J. Hellerstein, and S. Parekh, ''MIMO control of an Apache web server: Modeling and controller design,'' *Proc. Amer. Control Conf.*, May 2002, pp. 4922–4927.

[3] (Feb. 07 2013). *OlharDigital*. [Online]. Available: http://olhardigital.uol.com.br/noticia/os-20-sites-mais-acessados-do-mundo/32477

[4] T. W. M. Abreu e and W. Barra, ''Parameter identification strategies applied to dynamical modeling of an apache Web server,'' (in Portuguese), *RevistaControle Automação*, vol. 23, no. 1, pp. 38–48, Jan. 2012.

[5] M. V. Barreto, F. F. BezerraFilho, W. Barra, Jr, e F. G. Nogueira, ''Feedback control for dynamic performance improvement of HTTP response in virtual machines,'' (in Portuguese), *Proc. 7th Brazilian Symp. Intell. Automat. Natal*, 2015.

[6] K. Coar and R. Bowen, *Apache Cookbook*, 1st ed. Newton, MA, USA: O'Reilly Media, 2003.

[7] P. Deitel and H. Deitel, *C++ How to Program*, 10th ed. London, U.K.: Pearson, 2016.

[8] J. A. Marques, P. Ferreira, C. Ribeiro, L. Veiga, and R. Rodrigues, *Operating Systems*. Rio de Janeiro, Brazil: LTC Press (in Portuguese), 2011.

[9] L. A. Aguirre, *Introduction to System Identification: Linear and Nonlinear Techniques*. Belo Horizonte, Brazil, UFMG Press (in Portuguese), 2007.

[10] N. S. Nise, *Control System Engineering*, 7th ed. Hoboken, NJ, USA: Wiley, 2015.

[11] K. Ogata, *Modern Control Engineering*, 5th ed. London, U.K.: Pearson, 2015.

[12] N. Matthew and R. Stones, *Beginning Linux Programming*. Birmingham, England: Wrox Press, 1996.

[13] A. S. Tanenbaum, *Modern Operating Systems*, 4th ed. London, U.K.: Pearson, 2016.

[14] H. Chen, J. Yu, C. Hang, B. Zang, and P.-C. Yew, ''Dynamic software updating using a relaxed consistency model,'' *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 679–694, Sep./Oct. 2011.

[15] Z. Zhu, X. Gao, and C. Zhang, ''Research of an adaptive PI admission control of Web service,'' in *Proc. 26th Chin. Control Decision Conf. (CCDC)*, May /Jun. 2014, pp. 2290–2293.

[16] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, ''Feedback control architecture and design methodology for service delay guarantees in Web servers,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 1014–1027, Sep. 2006.

[17] I. K. Gupta, J. Kumar, and P. Rai, ''Optimization to quality-of-service-Driven Web service composition using modified genetic algorithm,'' in *Proc. Int. Conf. Comput. Commun. Control (IC4)*, Sep. 2015, pp. 1–6.

[18] E. Casilari, F. J. Gonzblez, F. Sandoval, ''Modeling of HTTP traffic,'' *IEEE Commun. Lett.*, vol. 5, no. 6, pp. 272–274, Jun. 2001.

MARCOS V. S. BARRETO received the L.S. degree in mathematics from the Universidade do Estado do Pará, Pará, Brazil, in 2005, the T.S degree in system analysis and development from the Centro Universitário do Pará, Pará, in 2001, and the M.S. degrees in electrical engineering from the Universidade Federal do Pará, Pará, in 2007, where he is currently pursuing the Ph.D. degree in electrical engineering.

Since 2001, he has been a Professor with the Instituto Federal de Educação, Ciência e Tecnologia do Pará. His research interests include databases, control systems, process management, and computational resources.

WALTER BARRA, Jr., received the B.E., M.Sc., and Ph.D. degrees in electrical engineering from the Federal University of Pará (UFPA), Brazil, in 1991, 1997, and 2001, respectively, where he is currently a Full Professor with the Faculty of Electrical Engineering. He has experience in electrical engineering with emphasis on automation and control of electrical power systems. His current research interests include adaptive control, control and stability of electric power systems, and industrial automation.

ERICK MELO ROCHA received the bachelor's and M.S. degree in electrical engineering from the Federal University of Pará (UFPA), Brazil, in 2010 and 2012, respectively, where he is currently pursuing the Ph.D. degree.

He is currently an Assistant Professor with the Department of Electrical Engineering, UFPA. He has experience in electrical engineering with emphasis in industrial automation, industrial processes, power electronics, and control systems. His current research interests include detection and fault diagnosis using computational intelligence techniques and robust control techniques investigation applied in industrial systems.

FABRÍCIO G. NOGUEIRA received the B.E. degree in computer engineering, and the M.Sc. and Ph.D. degrees in electrical engineeringfrom the Federal University of Pará, Brazil, in 2007, 2008, and 2012, respectively. He is currently an Adjunct Professor with the Faculty of Electrical Engineering, Federal University of Ceará (UFC). He has experience in control and automation of dynamic systems. His research interest includes LPV adaptive techniques.

KEVIN LUCAS MARCILLO received the bachelor's degree in electronic and telecommunications engineering from the Escuela Superior Politécnica del Litoral (ESPOL), Ecuador, in 2015, and the master's degree in electrical engineering from the Federal University of Pará (UFPA), Brazil, in 2018. He is currently pursuing the Ph.D. degree in control and automation engineering (power electronics and microgrids) from the Department of Automation and Systems, Federal University of Santa Catarina (UFSC), Brazil.

He is currently an Invited Researcher with the Power Systems Control Laboratory (LACSPOT), UFPA, and also with the Department of Electrical and Electronics Engineering, ESPOL. He has experience in electrical engineering, with emphasis on electronics, data acquisition systems, control and automation of electrical and industrial processes, power electronics, hydraulic systems, and control systems applied in power generation. His current research interests include modeling and robust control of industrial, electronic power systems, and microgrids.

RENAN LANDAU PAIVA DE MEDEIROS received the B.E., M.Sc., and Ph.D. degrees in electrical engineering from the Federal University of Pará (UFPA), Brazil, in 2013, 2014, and 2018, respectively.

He is currently an Associate Professor with the Department of Electrical Engineering, Federal University of Amazonas (UFAM). He has experience in electrical engineering with emphasis on automation and control of industrial and electrical power systems, and multivariable robust control and application. Since 2017, he has been a full time Researcher of the (e-CONTROLS) a research group interesting in any topics of the dynamic and control systems with emphasis in electric power system control. His current research interests include nonlinear control, multivariable robust control, modeling, and design a robust control for electrical power systems.

THIAGO W. M. ABREU is currently with the Laboratoire Images, Signaux et Systèmes Intelligents (LISSI)–EA 3956, Université Paris-Est Créteil Val de Marne–Université de Paris. His research interests include wireless sensor networks, probability theory, and control systems engineering.

MAYSA SOUSA ALVES is graduated in computer engineering, in 2018. She is currently pursuing the M.S. degree in electrical engineering with the Federal University of Pará (UFPA). She has experience in control and automation applied in industrial systems.