Automatic Feedback, Control-Based, Stress and Load Testing

Mohamad Bayan   João W. Cangussu
Department of Computer Science
University of Texas at Dallas

{msb021000,cangussu}@utdallas.edu

ABSTRACT

Stress and load testing are of special interest to many segments of the software industry. However, existing approaches present limitations with respect to automation and applicability. While some approaches present an automated solution, they lack in terms of applicability. A fully automated approach with high applicability is proposed here. The approach is based on the use of a PID controller to automatically drive the inputs and to achieve a pre-specified level of stress/load for resources of interest. The approach also allows for automatic identification of inputs impacting the resources. An experiment along with multiple simulation runs are presented as an indication of the applicability and accuracy of the proposed approach.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*

General Terms

Verification, Performance

1. INTRODUCTION

Load and stress testing are important for all segments of the software industry, but they are of special interest for segments such as embedded systems, web-servers, and many database-oriented systems. In general, embedded systems have a constrained environment with limited resources. This makes load and stress testing crucial to assess an embedded system's operation when loaded or stressed. Also, both web applications and database applications must assess how many requests can be handled within a specific response time. Stress and load test provide an alternative to verify these issues. Load Testing[2, 3] assesses how a system performs under a given "load." The rate at which transactions are submitted to the system is called the Load[2]. Stress Testing[2] requires subjecting a system to an unreasonable load with the intention of breaking it. The system is not expected to process the overload without adequate resources, but to behave (e.g., fail) in a decent manner (e.g., not corrupting or losing data).

The goal of this project is to develop an automatic stress-and-load testing technique. By automatically driving the resource usage to its limit, we can detect load-sensitive faults and verify performance issues under stress conditions. Initial experiments have shown that the proposed technique also identifies memory leaks. The automatic identification of these faults can have a large impact on the quality of the products released as well as on the reduction of the required test effort.

The proposed approach is based on the application of a feedback PID (Proportional, Integral, and Derivative) controller to drive the input and make the system achieve a specified level of resource usage. For example, if the user defines the system should be tested with a memory use of 95%, starting from an initial input value, the PID controller will automatically change the input(s) until the desired level of stress has been achieved. Stress can also be simultaneously achieved for more than one resource of interest, for example, memory and bandwidth. Experiments have shown that resources used by the PID controller itself are almost negligible and have minor or null impact on systems resources. Also, the proposed approach can properly identify inputs that have an impact on the resource(s) of interest. This is accomplished by the use of system-identification techniques[9] and does not rely on source code inspection, considerably increasing the applicability of the proposed approach.

Most of the work done with respect to stress and load testing is either domain specific or restricted to the stress of only one resource. Other approaches require the availability and analysis of source code, limiting their scope and applicability. The approach proposed here can be applied to any system, and it can control any desired resource. In addition, the automation of the complete process appears to be more feasible than existing approaches.

The remainder of this paper is organized as follow. Section 2 presents general concepts with respect to System Identification Techniques. The proposed approach is presented in Section 3 followed by experimental results in Section 4. Relevant existing methods for stress and load testing are briefly described in Section 5, which also draws some comparisons with the approach proposed here. Finally, we present our conclusions and future work in Section 6.

2. BACKGROUND

Before starting automatically driving the resource of interest to the desired stress/load level, one needs to identify the inputs that are sensitive to that specific resource. System identification techniques [9, 8] can be used for this purpose and are briefly described next.

A model capturing a system's behavior must be available if such system is to be controlled. A model that can capture the dominant behavior of aspects of any system is unlikely to be statically created due to the variety of scenarios and environments where these systems execute. However, based on the observations of a system's behavior when executing in a specific environment, a model can be dynamically customized for this scenario. Ljung states "System Identification deals with the problem of building mathematical models of dynamical systems based on observed data from the system." [9] In this scenario, the ingredients to apply System Identification techniques are present and a model can be inferred from the observations.

$$\begin{cases} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{cases} \quad (1)$$

A number of techniques, such as Least-Square and Markov Parameters [9] are available to identify state space models as given by Eqn. 1. To verify our initial experiments' applicability and accuracy, we chose the Least-Square identification procedure available in MATLAB; we present a concise description of this technique below.

Let $A, B, C,$ and D be the matrices in Eqn. 1 and organize them as indicated in Eqn. 2.

$$Y(t) = \begin{bmatrix} x(t+1) \\ y(t) \end{bmatrix} \quad \Theta = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad \Phi(t) = \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \quad (2)$$

Using the definitions of $Y(t)$, Θ, and $\Phi(t)$ above, we can rewrite Eqn. 1 as $Y(t) = \Theta\Phi(t)$. The observation of the system resources and the inputs/outputs of the application are used to construct the vectors $Y(t)$ and $\Phi(t)$. Then, a Least-Square approach as in Eqn. 3 can be used to compute an approximation ($\hat{\Theta}_N^{LS}$) for the matrix (Θ).

$$\hat{\Theta}_N^{LS} = \left[\frac{1}{N} \sum_{t=1}^{N} \Phi(t)\Phi^T(t) \right]^{-1} \frac{1}{N} \sum_{t=1}^{N} \Phi(t)Y^T(t) \quad (3)$$

3. PROPOSED APPROACH

As stated in Section 1, the purpose of the technique proposed here is to automate stress and load testing. Any software system may face resource saturation after running for a long time or while running with a heavy load. To detect this problem, stress and load testing becomes essential.

Figure 1 presents our proposed approach's general structure. The approach has three major components:"Input Identification," "Controller Tuning," and "Controller." The first identifies the inputs that affect the resources of interest. The "Controller Tuning", which is outside the scope of this paper, automatically computes the internal parameters of the chosen controller. Given an initial test case, the "Controller" computes the sequence of test cases to drive the resource of interested to the desired load/stress level. Description of the two blocks addressed here are given next.

3.1 Input Identification

Before we begin controlling any resource of interest, we must identify the inputs that affect the resource. For example, a system may have $\{I_1, I2, \ldots, I_{15}\}$ as the set of inputs while only inputs $\{I_4, I_7, I_8, I_{11}\}$ significantly affect resource R_1. Inputs here refer to normal inputs (determine the regular program outputs) and the load/stress test parameters (used to generate appropriate load/stress on the system). For an airline reservation system, for example, normal inputs would include start airports, start and return dates, etc.; normal outputs would include matching flights and their times,

fares, etc. The load/stress test input parameters would include number of concurrent users, their respective "think" times, etc., and the corresponding outputs would be average response time, system throughput, etc. The approach proposed here works for all types of inputs and outputs as long as corresponding data is available.

As observed later in Section 4.1, the resource cannot be controlled if the adequate set of inputs is not properly identified. The use of System Identification techniques [9] allows for identification of the subset of inputs that affect the output (the resource of interest). For explanation purposes, let us assume a system of order n with m inputs and only one output. In this case, the output part of Eqn. 1 is given by Eqn. 4

$$y(t) = [c_1 \ c_2 \ldots c_n] \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} + [d_1 \ d_2 \ldots d_m] \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_m(t) \end{bmatrix} \quad (4)$$

The method described in Section 2 computes all the matrices A, B, C, and D of the system for a given set of input and associated output data. In this discussion, we restrict the interest to matrix D. The output variable $y(t)$ is the resource under consideration and the matrix D determines which inputs $\{u_1, u_2, \ldots, u_m\}$ affect the output. That is, if $d_i = 0$, then, the input has no direct impact on the resource to be controlled. However, the use of the value 0 in the decision to include or not the input leads to many false positives (inputs that are selected by the approach but do not actually affect the output). To minimize this problem a cut off value is defined. First, matrix D is normalize to 100%, as given in Eq. 5, with respect to the summation of its absolute values. Absolute values are used to accomodate inputs that affect the resource in a positive as well as in a negative way.

$$d_j^n = \frac{d_j \times 100}{\sum_{i=1}^{m} |d_i|} \quad \text{for } j = 1, 2, \ldots, m$$

$$D^n = [d_1^n \ d_2^n \ldots d_m^n] \quad (5)$$

Now, a cut-off value z is pre-specified to determine which inputs affect the output. The input u_i is considered to affect the output if $d_i^n > z$. Clearly, z's value impacts the approach's accuracy. A small value for z would lead to cases where the input is selected but should not be considered. A large value for z would lead to drop of inputs that should be considered.

3.1.1 Validation of Automatic Input Identification

The first issue to be considered for the input identification is the number of test cases, i.e., the size of a test set T needed to make a proper selection. Use of a brute force algorithm demands an extremely large number of test cases when the number of inputs is large. Notice that not only individual values but also all possible combinations of the inputs need to be evaluated to ensure the inputs affect the resource. Eqn. 6 provides the number of combinations required by a brute force algorithm multiplied by a constant c. A single test case cannot show that an input or set of inputs affects the resource, and at least three test cases ($c = 3$) are required for that: an initial test case t_1; a test case showing a decrease in the resource usage when compared to t_1; and a test case showing an increase in the resource usage when compared to t_1. However, the value of c, according to our experiments, will range from 5 to 10.

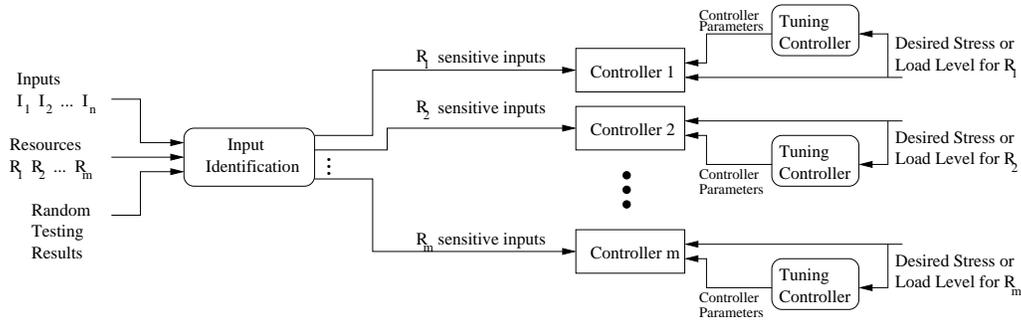$$|T| = c \times \sum_{k=0}^{n} \binom{n}{k} = c \times 2^n \quad (6)$$

Figure 1: Proposed Approach

Let us assume that a total of 15 inputs are available for an application P. To verify the proposed approach's accuracy, we executed a series of simulation runs for test sets in the range $|T| = 250, 300, \ldots, 1000, 1050$. For each size of test set, we executed a total of 100 simulation runs. The number of inputs affecting the output is randomly decided. Simulation is used here as a preliminary validation of the proposed approach, the use of real applications and corresponding test cases are being conducted.

The next step to generate simulation data is to define how the inputs will affect the output. Eqn. 7 defines how the resource y should behave with respect to changes in the selected inputs.

$$y = \sum_{i=1}^{n} f_k(u_i) + w \qquad (7)$$

where n is the number of selected inputs ($n \leq 15$), w is some random noise introduced in the function, and $f_k(u)$ is one of the four available functions: $f_1(u) = \sqrt{u}$, $f_2(u) = a \times u$, $f_3(u) = u^2$, and $f_4(u) = -b \times u$. The values of the constants a and b are generated randomly as well as each value of k in Eqn. 7. The range of values for each input u_i is also defined randomly. A vector $U = [I_1 \ I_2 \ \ldots \ I_{15}]$ indicates the inputs used in the computation of the output y. $I_j = 0$ determines the input has no impact and $I_j = 1$ shows otherwise. After the selection using the System identification procedure, a new vector $U^s = [I_1^s \ I_2^s \ \ldots \ I_{15}^s]$ determines, in the same way, which inputs our approach selected. Now, let $U - U^s = [s_1 \ s_2 \ \ldots \ s_{15}]$. A success is represented by $s_j = 0$, a false positive by $s_j = -1$, and a false negative by $s_j = 1$.
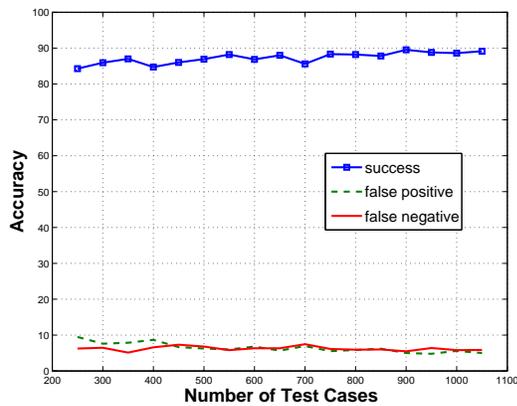

Figure 2: Results of the Automatic Identification of inputs for test sets with size $|T| = 250, 300, \ldots, 1000, 1050$.

Figure 2 depicts the results of the simulation runs. The Figure shows a small increase in accuracy (an increase in the number of successes and a decrease in the number of false positives and false negatives) as the number of test cases increases from 250 to 1050. On average, our approach achieved an accuracy of 87% when test cases were randomly generated. Though it is out of the scope of this paper, an adaptive procedure to select test cases and minimize false positives and negatives is already under investigation. Our conjecture, based on preliminary experiments, is that the accuracy will increase to around 99% with only a small increase in the number of test cases.

Figure 2 shows that the Automatic Identification of inputs is not very sensitive to the size of the test set. The lower bound of $|T| = 250$ is due to the limitations of the approach. That is, on average, our approach requires a minimum of 250 data points to compute the values of the matrices from Eqn. 2. As the number of test cases increases by 420%, the accuracy increases by less than 5%. Using Eqn. 6 (assuming $c = 5$) leads to $|T| = 163840$ test cases needed for the brute force algorithm. Comparing this with the results from Figure 2 shows an improvement of 156 times with only a small compromising of accuracy. The reduction of the number of test cases using the proposed approach is already substantial and, as stated above, the use of an adaptive technique will definitely increase accuracy which will make the approach even more appealing.

In general, while the system identification approach presents a linear growth with respect to the number of test cases required for a given number of inputs, the brute force algorithm presents an exponential growth. The brute force approach is more efficient for systems with less than 6 inputs but becomes extremely inefficient as the number of inputs increases. Except for the static source-code based identification of resource-sensitive inputs (which does not depend on test cases but has restrictions with respect to the types of resources it can handle) [12], we could not find an approach with the same goal as ours and, therefore, no comparison could be made.

3.2 Controller

There are many types of control techniques available such as the PID Control, Direct Pole Placement, Adaptive Control, and Optimal Control among others. A PID controller has been used in the experiments discussed here due to its simplicity, efficiency, and low overhead. We have deferred an investigation of alternative control methods for now.

A PID controller is used to derive test cases automatically, and consequently, achieve a pre-specified level of stress/load (the setpoint) for a specific resource. The PID controller accepts a setpoint as an input which represents the level of resource usage to be achieved by the application. The PID controller uses an initial

test case, defined by the tester, to drive the application to achieve that level of usage, independent of the initial test case value. Every time the current usage is fed back to the PID controller, the PID controller computes a new gain. In general, positive gain represents an increase in the input and negative gain represents a decrease in the input. To accurately represent the again in terms of the input, we use wrappers (which are application specific) that use the gain produced by the PID controller and the previous set of input(s) to determine a new set of input(s).

In addition to the application being tested, our approach requires a resource reporter which retrieves the current usage of the resource of interest and feeds this usage back to the PID controller.

One important aspect to be considered for stress and load testing is the simultaneous control of multiple resources[10] such as memory, disk, bandwidth. The simultaneous control does not pose any problem in a scenario where the resource sensitive input sets are completely disjoint. That is, in Figure 1, if $\{R_1\ sensitive\ inputs\} \cap \{R_2\ sensitive\ inputs\} \cap \ldots \{R_n\ sensitive\ inputs\} = \emptyset$ then each controller can be applied independently. However, when there is an overlap among these inputs, a change in one input with respect to one resource may negatively impact another resource. In some cases, both resources may not be simultaneously controllable while in other cases the overlapping inputs may be disregarded with minimum impact. The study of multi-variable control techniques is a subject of future work.

4. EXPERIMENTAL RESULTS

Some experiments have been conducted to show the applicability of the proposed approach. One experiment, presented elsewhere [5], examined the control of how much memory an application used. That experiment shows that a controller can properly change the inputs and drive memory usage to a desired level. Also, we can detect memory leaks using a controller and the allocated memory's behavior [5]. Here another experiment is presented which introduces more interesting aspects such as the control of more than one variable and a less deterministic behavior of the resource under control; response time in this case.

4.1 Experiment α: Response Time

Nowadays, web applications are the most widely used software applications. An overloaded Web site is a major problem and, once happening, the problem often cannot be addressed rapidly. To avoid overloading the application in production, Load and Stress Testing become essential. Testing a web application involves testing techniques such as Load Tests, Stress Tests, Failover Tests, Soak Tests, Performance Test, and Network Sensitivity Tests among others. However, one of the most fundamental tests is Load Test which is end-to-end performance test under anticipated production load. The test's primary objective is to determine the response times for time-critical transactions and to measure the application's ability to function correctly under load. For the purpose of this paper, response time is the interval between the receipt of the end of transmission of an inquiry message and the beginning of the transmission of a response message. Slow response time will distract users and make it more annoying and difficult to interact with a site. Having that in mind makes it important to load test the server and determine the number of clients the server can handle without exceeding a pre-determined response time.

In this experiment, we considered a remote client server application (emulating a web server). Response time was our resource of interest. The system consisted of a Server that accepted client requests, processed the requests, and sent responses to the clients.

Our goal was to load test the server by adding clients until we achieved a specified response time (Setpoint). In other words, if the Setpoint was 5ms, our goal was to determine how many clients the server could handle without exceeding the 5ms response time.

There were two types of clients, Regular Clients and VIP Clients. The server served VIP Clients faster than Regular Clients. On average, the server took twice as much time to process a Regular Client than a VIP Client. In this experiment each client (both VIP and Regular) sent requests continuously to the server. Two variables affected the Average Response Time: (i) Total number of clients in the system (totalClients); and (ii) the percentage of regular clients in the system (perRegClient). Average Response Time increased as the number of Clients increased. Also, the Average Response Time increased as the percentage of Regular Clients increased since Regular Clients took more time to be processed. Results achieved when controlling one variable are discussed in Section 4.1.1, results achieved when controlling one variable of two inputs are discussed in Section 4.1.2, and results achieved when controlling two variables of two inputs are discussed in Section 4.1.3.

4.1.1 Controlling one variable

In this first experiment, only Regular Clients are considered (all clients are processed equally by the server). Since we only have Regular Clients in the system, the perRegClient variable will always be 100. Therefore, total number of clients (totalClients) is the only variable to control.
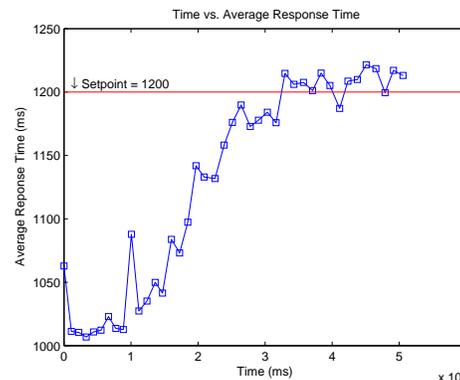

Figure 3: Application of a PID controller to achieve a certain average response time for the remote client server application. Only regular clients are considered and the total number of clients is the only input variable used by the controller.

In Figure 3 the system started with an initial test case of totalClients=5 and the desired Average Response Time was 1200ms (setpoint). As time advanced and the PID Controller produced positive gains, the number of Clients increased until the Average Response time reached the setpoint of 1200ms. It took 18 clients to reach the setpoint of 1200ms. Unlike the memory usage experiment, response time for a remote client server application is less deterministic. That is, while memory usage can be completely controlled and, in most cases, driven to the exactly specified level and remain there, the same does not occur to response time. The server, though dedicated, still has to handle OS-related requests or backups and this affects the response time. That is, the average response time will oscillate around a value, but it will not always be the same value. Figure 3 clearly shows this behavior. After creating 18 clients, the average response time oscillates around the setpoint (this represents a asymptotically stable system[10]).
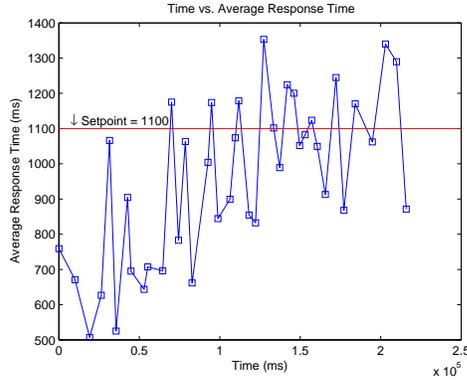
Figure 4: Application of a PID controller to achieve a certain average response time for the remote client server application. Both Regular and VIP clients are considered and the total number of clients is the only input variable used by the controller.

In this experiment, only one variable, the total number of regular clients, has been used to control the system and achieve the desired average response time. However, most applications have several inputs affecting the resource. Next, experiments where two variables affect the average response time (resource) are presented. They are a clear indication of the necessity to properly identify and control all the inputs that significantly affect the resource to be stressed.

4.1.2 Controlling one variable of two inputs

In this experiment, we considered both Regular and VIP clients. As described earlier, the server serves VIP Clients faster than Regular Clients. Therefore, not only the total number of clients but also the percentage of Regular Clients affects the Average Response Time. To demonstrate the need to identify and control all the inputs affecting the resource, only the variable totalClients (total number of Clients) is used to reach the setpoint while perRegClients (percentage of Regular Clients) changes randomly. That is, after all the existing clients have reported their average response time to the controller, the wrapper accepts the gain produced by the PID controller and the total number of currently running clients (totalClients) to determine the new number of clients (totalClients). However, the perRegClients is generated randomly.

The system in Figure 4 started with 2 clients (totalClients) and perRegClients set to 50% (1 Regular Client). As time advanced, the total number of clients increased due to the gains produced by the PID; but the number of Regular and VIP clients changed randomly. Since only one of the variables that affected the resource was under control, the behavior of the average response time could not achieve a reasonable level of stability and the oscillations around the setpoint were too large as seen in Figure 4. This is a clear indication that both variables need to be controlled. One could argue that one variable could be controlled while all others are kept constant, however this can significantly decrease the effectiveness of a Test Suite leading to many important combinations of relevant input variables not being tested.

4.1.3 Controlling two variables of two inputs

The difference between this experiment and the one described in the previous section is that the two variables (totalClients– Total number of Clients and perRegClients–percentage of Regular Clients) affecting response time are used to control the system and achieve the Setpoint.
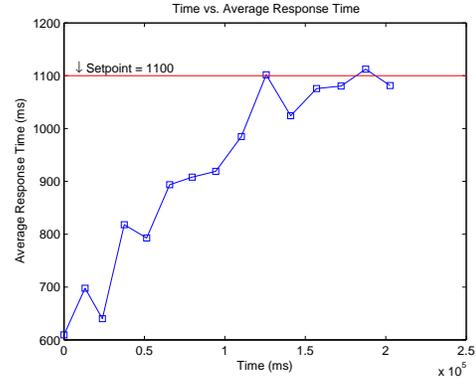

Figure 5: Application of a PID controller to achieve a certain average response time for the remote client server application. Both Regular and VIP clients are considered; total number of clients and percentage of Regular Clients are both used by the controller.

The system in Figure 5 started with 15 clients (totalClients) with perRegClients set to 13.33% (2 Regular Clients). After all the existing clients had reported their average response time to the controller, the controller calculated the gain. The wrapper accepted the gain produced by the PID controller as an input in addition to totalClients and perRegClients. The wrapper determined the new number of clients (totalClients) and the new percentage of Regular Clients (perRegClients).

So, if 10 clients were running with 40% regular Clients (4 Reg and 6 VIP) and the PID controller produced a gain of 50%, then, the wrapper would have added 5 clients to set totalClients to 15 and increased perRegClients to 60%. So, the system would have ended with 9 Regular Clients and 6 VIP Clients. However, if the PID controller produced a gain of -30% then the controller would have kept only 7 clients and decreased perRegClients to 12%. So, the system would have ended with 1 Regular Client and 6 VIP Clients. As can be seen from Figure 5, when both variables affecting response time were controlled, the system was able to achieve the desired average response time and the oscillations around this value were considerably small when compared to Figure 4 where only one variable was controlled.

5. RELATED WORK

Both black box and white box approaches have been proposed for Load and Stress Testing. One example of a white box approach is provided by Yang and Pollock [12]. Their technique identifies the load sensitive parts in sequential programs based on a static analysis of the code. Another white box approach for Stress Testing is given by Grosso [7]. It uses Genetic Algorithms to generate test cases to identify buffer overflow threats. Grosso's approach deals only with buffer overflow and not resource saturation. The approach proposed here presents a higher applicability because it does not depend on the availability of source code and it can be used to potentially control any type of resources. White box techniques are, in general, restricted to certain types of resources. In terms of accuracy, if the system is controllable and stable or asymptotically stable [10], the use of control theory guarantees that the stress/load level will be achieved. If the system is uncontrollable and unstable, no technique can be accurate. However, the proposed approach depends on the proper identification of the inputs affecting the resource. As described in Section 3.1, the use of system

identification techniques [9] have shown to be reasonably accurate to identify such inputs. Further improvement of the identification procedure is already under investigation.

Different black box testing techniques have been proposed for Load and Stress testing. One particular method, proposed by Avritzer and Larson, is called Deterministic Markov State Testing [4]. Their approach is limited to applications that can be modeled by using the Markov Chain because the input data is assumed to arrive according to a Poisson distribution and is serviced in an exponential distribution. Avritzer and Larson's approach is similar to the approach proposed here where they both concentrate on resources available. However, their approach is limited to systems that can be modeled by Markov Chain and it generates the test case before starting the system test. The approach we offer here is more general and it generates the test cases automatically. Briand [6] developed a method based on Genetic Algorithms to analyze real-time architectures and to determine whether deadlines can be missed. The proposed method generates test cases, concentrating on seeding times for aperiodic tasks, such that completion times of a specific task's execution are as close as possible to their deadlines. Our approach has different goals. Our main concern is in resource saturation, whereas Briand's method focuses on missing deadlines.

To address Stress Testing in multimedia systems, Zhang [13] developed a technique and a tool to automatically generate test cases. The same goal is shared with our approach: resource saturation and testing with different loads. Zhang's technique is applicable where the specifications consist of a temporal-event Petri net and some temporal formulas for describing the relationships among media objects. The system is statically divided into a set of objects where the resource usage of each object is known statically. Not just any system can be divided into such objects and not all systems can be or are specified using Petri nets. In other words, Zhang's algorithm has applicability limitations. If a system can be divided into different objects with know resources requirements and these objects can be easily specified in petri nets then it may be easier to use Zhang's technique. However, our approach is applicable to any system.

When considering performance, load, and stress testing of web applications, a vast number of existing tool. For example, load-testing tools evaluate the performance of a specific web site on the basis of actual users' behavior; they capture real user requests for further replay where a set of test parameters can be modified. Httperf [11] developed at Hewlett-Packard and Mercury Load Runner [1] are some examples. All these tools try to achieve load generation and performance measurement by simulating client requests to the application under test. They all require user configuration for specifying the number of clients to simulate, and the test cases to run. After each run the tester has to manually revisit the test cases and add or modify them to achieve the test load required. The approach we discuss here automates the process of test loading the application. It also automatically identifies the resource sensitive inputs. To achieve this, we require only that testers provide an initial test case.

6. CONCLUSIONS

Stress and Load are important testing techniques. In this paper, we propose a new approach based on the use of feedback control theory. The approach, though of special interest for embedded systems and web servers, is not restricted to any one type of system or application. We offer a more general approach. Existing approaches present limitations in terms of applicability and resources to be controlled. The approach proposed here can be used to automatically control any type of resource (as long as it can be mea-

sured) and can also accurately identify resource-sensitive inputs in an automated manner. The results of experiments controlling memory usage and response time indicate the proposed approach offers both applicability and accuracy. Further, our remote client-server application results show a need to properly identify all the inputs affecting the resource of interest.

Two major problems, automatic identification of resource sensitive inputs and automatic stress/load testing for a resource, have been addressed in this paper. However, many other issues remain to be investigated. First, a set of actual applications with multiple inputs have been collected for examination to further validate the automatic identification of resource sensitive inputs. Second, tuning techniques appropriated for software systems need to be investigated. Finally, different types of control as well as multi-variable control techniques must be examined.

7. REFERENCES

[1] Loadrunner by mercury interactive.
http://www.mercuryinteractive.com/products/loadrunner/.

[2] *Software System Testing and Quality Assurance*. Van Nostrand Reinhold, 1984.

[3] *Testing Object-Oriented Systems - Models, Patterns, and Tools*. Addison-Wesley, 1999.

[4] Alberto Avritzer and Brian Larson. Load testing software using deterministic state testing. In *Proceeding of the International Symposium on Software Testing and Analysis*, pages 82–88. ACM, June 1993.

[5] Mohamad Bayan and Joao W. Cangussu. Automatic stress and load testing for embedded systems. In *3rd International Workshop on Software Cybernetics - 30th Annual IEEE International Computer Software and Applications Conference (COMPSAC 2006)*, Chicago, IL, Sept. 18-21 2006. IEEE.

[6] Lionel C. Briand, Yvan Labiche, and Marwa Shousha. Stress testing real-time systems with genetic algorithms. In *Proceedings of the Genetic And Evolutionary Computation Conference*, pages 1021–1028, Washington, DC, June 2005. ACM.

[7] Concettina Del Grosso, Giuliano Antoniol, Massimiliano Di Penta, Philippe Galinier, and Ettore Merlo. Improving network applications security: a new heuristic to generate stress testing data. In *Proceedings of the Genetic and evolutionary computation Conference*, pages 1037–1043, Washington, DC, 2005. ACM.

[8] Jer-Nan Juang. *Applied System Identification*. Prentice Hall, Englewood Cliffs, New Jersey, first edition, 1993.

[9] Lennart Ljung. *System Identification: Theory for the user*. Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[10] David G. Luenberger. *Introduction to Dynamic Systems: Theory, models and applications*. John Wiley & Sons, New York, 1979.

[11] David Mosberger and Tai Jin. httperf a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, 1998.

[12] Cheer-Sun D. Yang and Lori L. Pollock. Towards a structural load testing tool. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 201–228, San Diego, CA, 1996.

[13] Jian Zhang and S. C. Cheung. Automated test case generation for the stress testing of multimedia systems. *Software, Practice & Experience*, 32(15):1411–1435, 2002.