Improving Load, Performance and Stress Evolutionary Testing using a Hybrid Metaheuristic Approach

Francisco Nauber Bernardo Gois,Pedro Porfírio Muniz de Farias,André Luís Vasconcelos Coelho, Thiago Monteiro Barbosa[a,b,b,a]

[a]*Servio Federal de Processamento de Dados, Avenida Pontes Vieria ,832, Fortaleza, Ceará 60130-240*
[b]*Universidade de Fortaleza, Avenida Pontes Vieria ,832, Fortaleza, Ceará 60130-240*

Abstract

Many software must respond to thousands or millions of concurrent requests. These systems must be properly tested to ensure that they can function correctly under the expected load. Load, Performance and Stress Evolutionary testing aims to find test scenarios which produce execution times violating the timing constraints specified. The purpose of this paper is proposing the use of a approach using hybrid metaheuristc in load, performance and stress test models using Genetic Algorithms, Simulated Annealing and Tabu Search Algorithms. A tool named IAdapter, a JMeter Plugin to perform evolutionary load, performance or stress tests, was developed. Two experiments were conducted to validate the proposed approach. The first experiment has been applied in an emulated component and the second one has been applied in an installed Moodle application. In both experiments, the use of a hybrid metaheuristc has obtained better fitness values.

Keywords: Evolutionary Testing, Tabu Search, Hybrid Metaheuristics

1. Introduction

Many systems must support concurrent access by hundreds or thousands of users. The failure to scale users results in catastrophic failures and unfavorable media coverage[1]. To assure the quality of these systems, performance, stress and load testing is a required testing procedure[2].

The explosive growth of the Internet has contributed to increase the need for applications to perform at warp speed. Performance problems have a bad habit of turning up late in the application life cycle, and the later you discover them, the greater the cost to fix them [3]. The use of load testing is an increasingly common practice due to the increasing number of users. In this scenario, the inadequate treatment of a workload generated by concurrent or simultaneously access, generated by system users, can result in highly critical failures and corrosion of the company's image in their customers' view [4] [1].

The Load Testing determines the responsiveness, throughput, reliability or scalability of a system under a given workload. The quality of the results of system's load tests is closely linked to the implementation of the workload strategy. The performance of many applications depends on the load applied under different conditions. In some cases, performance degradation and failures arise only in stress conditions [5] [1].

Different parts of an application should be tested on various parameters and stress conditions [6]. The correct application of a load test should cover most part of application under ordinary conditions (Load or Performance Test) or above the expected load conditions(Stress Test) [4] [7] [8].

Evolutionary testing is seen as a promising approach for verifying timing constraints [9]. The main objective of load, performance and stress evolutionary testing is to find test scenarios which produce execution times violating the specified timing constraints [10].

The purpose of this paper is propose the use of a approach using hybrid metaheuristc with Genetic Algorithms, Simulated Annealing and Tabu Search Algorithms in load, performance and stress evolutionary tests.

The remainder of the paper is organized as follows. Section 2 presents a brief introduction in load, performance and stress tests. Section 3 presents concepts about Hybrid Metaheuristcs. Section 4 presents a brief introduction about evolutionary test definitions, techniques and state of art. Section 5 presents the IAdapter

tool. The Section 6 shows the results of two experiment applied with IAdapter. Conclusions and further work are presented in Section 7.

2. Load, Performance and Stress Test

Load, performance and stress testing is typically done to locate bottlenecks in a system, to support a performance tuning effort and to collect other performance-related indicators to help stakeholders get informed about the quality of the application being tested [11] [12].

The Performance Test aims at verifying a specified system performance. This kind of test is executed by simulating hundreds or more simultaneous users over a defined time interval [13]. The purpose of this test is to demonstrate that the system reaches its performance objectives [11].

In load tests, the system is evaluated in pre-defined load levels [13]. The aim of this test is to reach the performance targets for availability, concurrency, throughput and response time of the system. Load Test is the closest to real application use [3].

Stress test verifies the system behaviour against heavy workloads [11], being executed to evaluate a system beyond its limits, validate system response in activity peaks and verify if the system is able from recover from these conditions. They differ from other kinds of testing because the system is executed on or beyond its breakpoints, forcing the application or the supporting infrastructure to fail [13] [3].

Automated tools are needed to carry out serious load, stress and performance testing. Sometimes , there is simply no practical way to provide reliable, repeatable performance tests without using some form of automation. The aim of any automated test tool is to simplify the testing process [3].

In the context of testing, a scenario is a sequence of steps in your application. It can represent a use case or a business function such as searching a product catalog, adding an item to a shopping cart or placing an order [12].

Load, Performance and Stress results are measured by indicators. Some researchers advocate the 90-percentile response time is a better measurement than the average/medium response time, as the 90-percentile accounts for most of the peaks, while eliminating the outliers [1].

3. WorkLoad Model

Load, Performance or Stress testing projects should start with the development of a model for user workload that an application receives. This should take into consideration various performance aspects of the application and the infrastructure that a given workload will impact. A workload is a key component of such a model.

The term Workload represents the size of the demand that will be imposed on the application under test in an execution. The metric unit used for define a Workload is dependent on the application domain, such as the length of the video in a transcoding application of multimedia files or the size of the input files to a file compression application [14] [3] [15].

Workload is also defined by the distribution of load between the identified transactions at a given time. Workload helps us study the system behavior identified in several load model. Workload model can be designed for verify predictability, repeatability and scalability of a system [14] [3].

Workload modeling is the try to create a simple and general model, which can then be used to generate synthetic workloads. The goal is typically to be able to create workloads that can be used in performance evaluation studies. Sometimes, the synthetic workload is supposed to be similar to those that occur in practice on real systems [14] [3].

There are two kinds of Workload models: descriptive and generative. The difference is that descriptive models just try to mimic the phenomena observed in the workload, whereas generative models try to emulate the process that generated the workload in the first place [13].

On descriptive models, one finds different levels of abstraction on one hand, and different levels of faithfulness to the original data on the other hand. The most strictly faithful models try to mimic the data directly using statistical distribution of data. The most common strategy used in descriptive modeling is to create a statistical model of an observed workload (Fig. 1). This model is applied to all the workload attributes, e.g. computation, memory usage, I/O behavior, communication, etc [13]. The Fig. 1 shows a simplified workflow of a descriptive model. The workflow has six phases. In first phase, the user uses the system in the production environment. In second phase, the tester collects user's data, like logs, clicks and preferences, in the system . The third phase consists in developing a model to emulate the user's behaviour. The fourth phase is made up of the execution of the test, emulation of the user's behaviour and log's gathering.

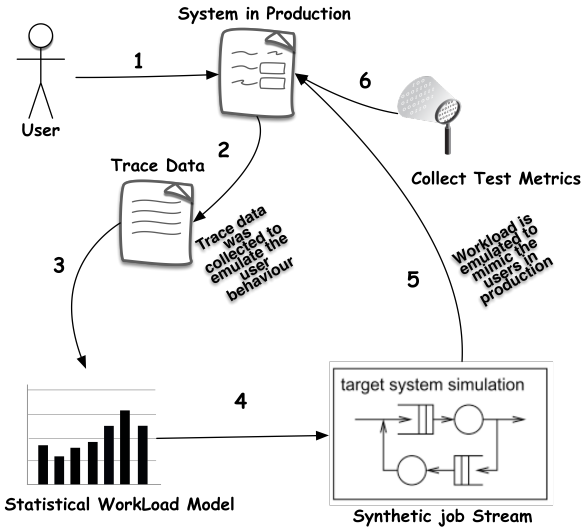


Figure 1: Workload modeling based on statistical data [13]

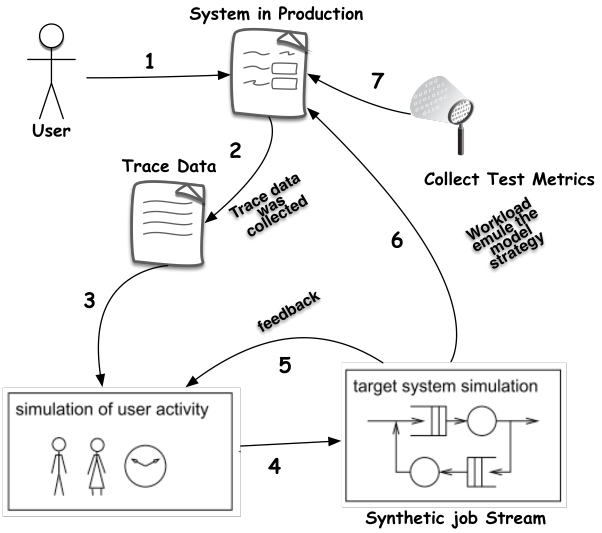


Figure 2: Workload modeling based on Generative Model [13]

Generative models are indirect, in the sense that they do not model the statistical distributions. Instead, they describes how users will behave and when they generate the workload. An important benefit of the generative approach is that it facilitates manipulations of the workload. It is often desirable to be able to change the workload conditions as part of the evaluation. Descriptive models do not offer any option regarding how to do so. But with generative models, we can modify the workload-generation process to fit the desired conditions [13]. The difference between the workflows of descriptive and generative models is that user behavior is not collected from logs, but simulated from a model that can receive feedback from the test execution (Fig. 2).

4. Hybrid Metaheuristc

A large number of researchers have recognized the advantages and huge potential Building hybrid mathematical programming methods and metaheuristics. The main motivation to create hybrid Metaheuristics is to exploit the complementary character of different optimization strategies. In fact, choosing an adequate combination of algorithmic can be the key for achieving top performance in solving many hard optimization problems [16] [17].

There are two main categories of metaheuristc combinations: Collaborative Combinations and Integrative Combinations. The Fig. 3 presents the two main categories of Hybrid MetaHeuristc [16].

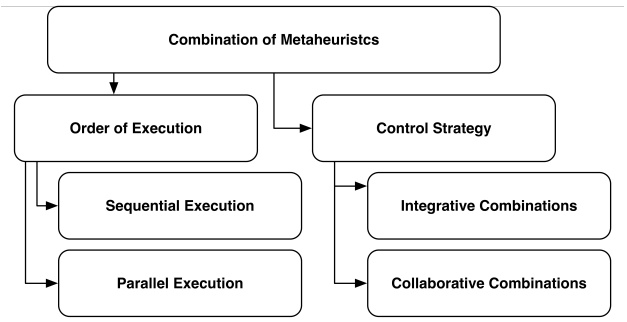


Figure 3: Categories of metaheuristc combinations [16]

Collaborative Combinations uses a approach where the algorithms exchange information, but are not part of each other. In this approach, algorithms may be executed sequentially or in parallel. The presented research work uses a type of Collaborative Combination with Sequential Execution.

5. Evolutionary Test in Load, Performance and Stress Tests

The search for the longest execution time is regarded as a discontinuous, nonlinear, optimization problem, with the input domain of the system under test as search

space [10]. The main objective of evolutionary testing in performance,stress and load tests is to find test scenarios which produce execution times violating the timing constraints specified. If a temporal error is found, the test was successful [10]. The application of evolutionary algorithms to load, performance and stress tests involves finding the best and worst case execution times (BCET, WCET) to determine if timing constraints are fulfilled [9].

Evolutionary tests uses a cost (fitnesse) function to select the best individuals. There has two measurement units normally associated with the fitnesse function in load, performance or stress test: Processor Cycles and Execution Time. The Processor Cycles approach describes a fitness function in terms of processor cycles. The Execution Time approach involves executing the application under test measuring the execution time [18] [19]. The Figure 4 shows a comparison between the presented research work and the load, performance and stress test researches presented by Afzal et. al. [18]. Afzal's work was added with some of the latest research in the area ([20] [5]). The x axis represents the type of tool used (Prototype or Functional Tool) and the y axis presents the metaheuristic used by each research (Genetic Algorithm, Tabu Search, Simulated Annnealing or a Customized Algorithm). The Figure also divides the researches by the type of function fitnesse (Execution Time or Processor Cycles). Most research is limited to making prototypes on genetic algorithms. The presented research work is distinguished from others by having a functional tool using a hybrid approach.

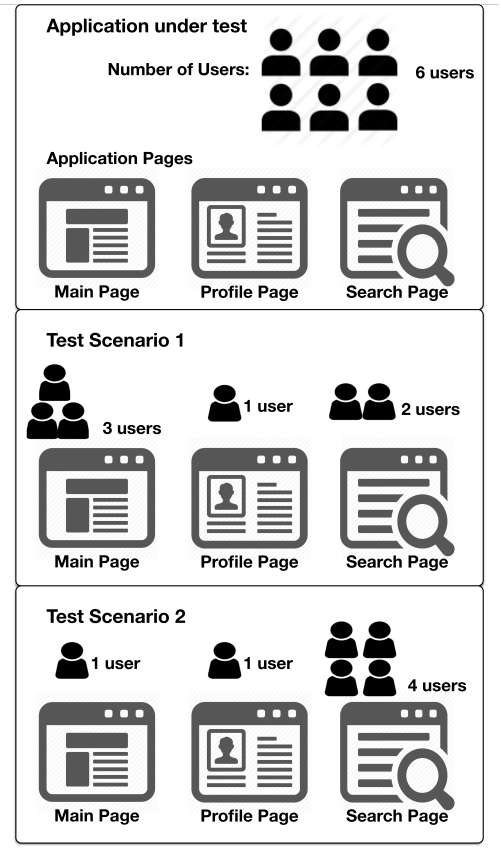
	Prototypes		Functional Tool
	Execution Time	Processor Cycles	Execution Time
Hybrid Metaheuristc			IADAPTER Gois, 2015
GA	Alander, 1996 e 1998 Sullivan, 1998 Wegener, 1997 Briand, 2005 Canfora, 2005	Wegener and Grochtmann, 1998 Mueller, 1998 Puschner, 1998 Wegener, 1999 Groß 2000,2001 and 2003 Tlili, 2006	Di Penta, 2007 Garoussi,2006 Garoussi,2008 Garoussi,2010
SA			Tracey,1998
Customized Algorithm		Pohlheim,1999	

Figure 4: Distribution of the researches over range of applied metaheuristics

6. Improving Load, Performance and Stress Evolutionary Testing using a Hybrid Metaheuristic Approach

A Load, Performance or Stress tests uses set of workloads that consists of many types of usage scenarios and different user numbers combinations. A load is typically based on an operational profile. For example, the load of an e-commerce website would contain information such as the browsing or purchasing min/average/max rate.

Figure 5: Best results obtained in 27 generations



A performance test usually lasts for several hours or even a few days and only tests a limited number of workloads. The major challenge is to find the workloads that exposes a major number of errors in the application under test [21].

The proposed solution makes it possible to create a generative model that evolves during the test. The proposed solution model uses Genetic Algorithm, Tabu Search and Simulated Annealing in two different approaches. The first approach uses the three algorithms independently and the second approach uses the three

algorithms collaboratively (Hybrid Metaheuristic approach).

In the first approach , the algorithms do not share their best individuals among themselves. Each algorithm evolves in a separate way (Fig. 6). The second approach use the algorithms in a collaborative mode (Hybrid Metaheuristic). In this approach, the three algorithms share their best individuals found (Fig. 7).

The next subsections present details about the used metaheuristcs algorithms (genotype representation and fitnesse function) and the IAdapter components.

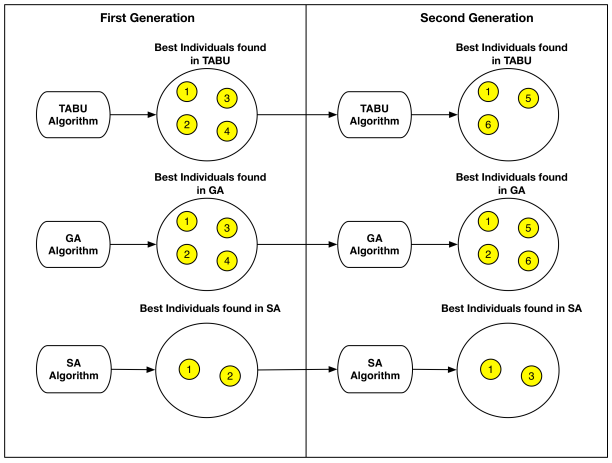


Figure 6: Use of the algorithms independently

6.1. Genotype representation

The Genotype representation is composed by a linear vector with 23 genes. The first gene represents the name of individual. The second gene presents the algorithm (Genetic Algorithm, Simulated Annealing or Tabu Search) used by the individual. The third gene represents the type of test (Load, Stress or Performance). Next genes represent 10 scenarios and their numbers of users. Each scenario is an atomic operation, the scenario must log in the application, run the task goal and undo any changes performed, returning the application to it's original state.

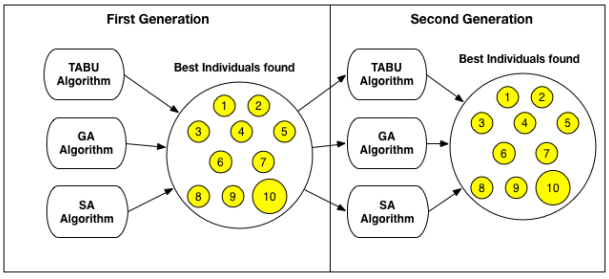


Figure 7: Use of the algorithms collaboratively

The Fig. 8 presents the genome representation and a example using the crossover operation. In the example, the genotype 1 has the Login scenario with 2 users; the Form scenario with 0 users and the Search scenario with 3 users. The genotype 2 has the Delete scenario with 10 users; the Search scenario with 0 users and the Include scenario with 5 users. After the crossover operation, We obtain a genotype with Login scenario with 2 users; the Search scenario with 0 users and the Include scenario with 5 users.

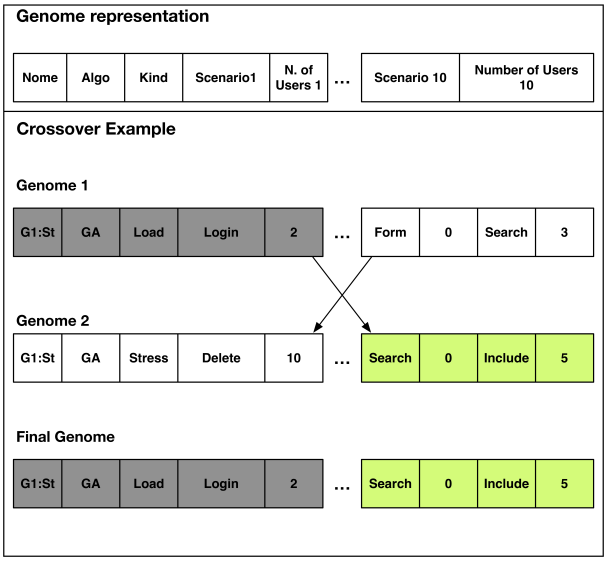


Figure 8: Genotype representation and crossover example

The Fig. 9 shows the strategy used by the IAdapter to obtain the genotype of the neighbours for the Tabu Search and Simulated Annealing algorithms. The neighbours are obtained by the modification of a single cromossome (scenario or number of users) in the genotype.

6.2. Initial population

The strategy use by the plugin to instantiate the initial population is to generate 50% of the individuals randomically and 50% of the initial population are distributed in three ranges of values:

- 30% of the maximum allowed users in the test ;
- 60% of the maximum allowed users in the test; and

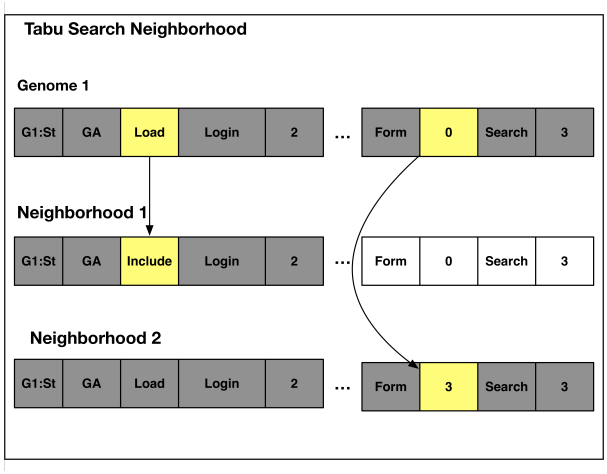


Figure 9: Tabu Search and Simulated Annealing neighbour strategy

- 90% of the maximum allowed users in the test.

6.3. Objective (Fitnesse) Function

The proposed solution was designed to be used with the independent testing teams in various situations where the team has no direct access to the environment where the application under test was installed. Therefore, The IAdapter uses a measurement approach to the definition of the fitnesse function. The fitnesse function applied to IAdapter solution is governed by the following equation:

$$\begin{aligned} fit = 90percentileweigth * 90percentiletime \\ +80percentileweigth * 80percentiletime \\ +70percentileweigth * 70percentiletime + \\ maxResponseWeigth * maxResponseTime + \\ numberOfUsersWeigth * numberOfUsers - penalty \end{aligned} \quad (1)$$

The proposed solution's fitnesse function uses a series of adaptable user-defined weights (90percentileweigth, 80percentileweigth, 70percentileweigth, maxResponseWeigth and numberOfUsersWeigth). These weights make it possible to customize the search plugin functionality. The penalty is applied when a application under test responds in a longer time than the level of service.

7. IAdapter

IAdapter is a JMeter Plugin to perform evolutionary load, performance or stress tests. JMeter is a desktop application, designed to test and measure the performance and functional behavior of applications [22]. The IAdapter plugin implements the solution proposed in the section 6

The JMeter have components organized in a hierarchical manner. The IAdapter plugin provides three main components: WorkLoadThreadGroup, WorkLoadSaver, and WorkLoadController.

The WorkLoadThreadGroup is a component that creates an initial population and configure the algorithms used in IAdapter . The Fig. 10 presents the main screen of the WorkLoadThreadGroup component. The component has a name ❶, a set of configuration tabs ❷, a list of individuals by generation ❸, a button to generate an initial population ❹ and a button to export the results ❺.

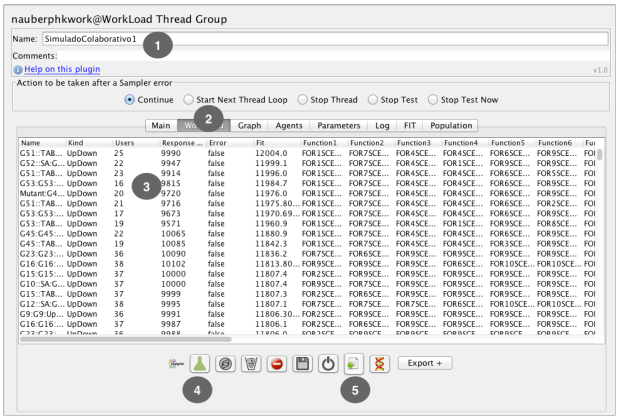


Figure 10: WorkLoadThreadGroup component

The WorkLoadSaver component is responsible for saving all data in the database. The operation of the component only requires its inclusion in the test script.

The WorkLoadController represents a scenario of test. The WorkLoadController represents a scenario of test. All actions necessary to test a application should be included in this component. All instance of the component need to login in the application under test and return the application to it's original state.

8. Experiments

This section presents two experiments. The first one has been applied in an emulated component and The second experiment has been applied in an installed Moodle application. The experiments used this fitnesse function:

Listing 1: SimulateConcurrentAcess class

```
1  public class SimulateConcurrentAccess {
2    @Test
3    public void test() {
4      synchronized (StaticClass.class) {
5        for (int i = 0; i <= 1000; i++) {
6          StaticClass.x += i;
7        }
8        StaticClass.x = 0;
9      }
10   }
```

$$
\begin{aligned}
fit = 0.9 * 90percentiletime \\
+0.1 * 80percentiletime \\
+0.1 * 70percentiletime+ \\
0.1 * maxResponseTime+ \\
0.2 * numberOfUsers - penalty
\end{aligned}
\tag{2}
$$

The fitnesse function used in the experiments intended to find individuals with the highest percentile of 90%, followed by individuals with higher percentile time of 80% and 70%, maximum response time and number of users.

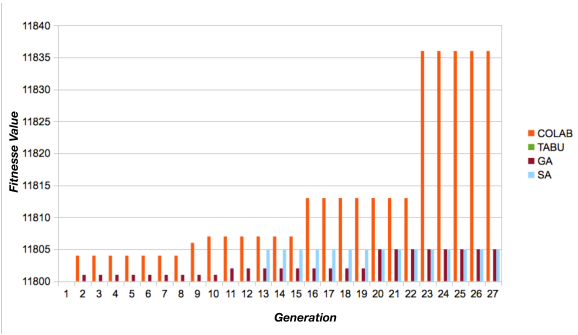
The first experiment has implemented 27 generations and the second experiment has performed 6 generations, with 300 executions by generation (100 times for each algorithm), generating 300 new individuals. The experiments had used a initial population of 100 individuals. The Genetic Algorithm used the top 10 individuals from each generation to the crossover operation. The Tabu List has been configured with the size of 10 individuals and expire every 2 generations. The mutation operation was applied to 10% of the population on each generation.

8.1. First Experiment- Emulated Class Test

The first experiment aimed to apply performance, load and stress testing in a simulated component. The purpose of using a simulated component is able to perform a greater number of generations in a shorter time available and eliminate variables such as the use of databases and application servers. The first experiment used a test class named SimulateConcurrentAccess. These class have a static variable named x and a set of methods that uses the variable in a synchronized context (Listing 1).

Fig.12 presents the best results in 27 generations applied in the first experiment . The Figure shows the results obtained with the algorithms with and without collaboration. The x axis represents the generation number and the y axis represents the best fitnesse value obtained until the current generation. The results of the experiment showed that the use of cooperation between the three algorithms resulted in find individuals with better fitnesse values.

Figure 11: Best results obtained in 27 generations



The table 1 presents the results obtained by the Hybrid Metaheuristc (HM), Genetic Algorithm (GA), Simulated Annealing (SA) and TABU Search (TS) from 27 generations in the first experiment. The values are the maximum value of the fitnesse obtained in each algorithm.

The signed-rank Wilcoxon non-parametrical procedure was used for comparing the results. The procedure showed that there was a significant improvement in the results with the collaborative approach.

8.2. Second Experiment- Moodle Application Test

The second experiment uses a Moodle application installed in a machine with 500 Gb of hard disk and 8 Gb of memory.The study used six application scenarios:

- PostDeleteMessage- This scenario post and delete messages in the moodle application.
- MyHome- This scenario access the user's homepage of the application.
- Login- This scenario are responsible by the user authentication of the application.
- Notifications- This scenario enter in the notification page of each user.
- Start Page- Initial start page of the application.
- Badge- This scenario enter in the Badge page.

The maximum tolerated response time in test was 30 seconds. Any individuals that obtained a time longer than the stipulated maximum time suffered penalties.

Table 1: Fitnesse function maximum value by algorithm

GEN	HM	TS	GA	SA
1	11238	11238	11238	11238
2	11804	11596	11801	10677
3	11787	8932	8411	10869
4	11723	9753	9611	10760
5	8164	9780	10738	4794
6	11802	9781	11086	6120
7	9985	5782	11272	11798
8	11803	11749	10084	11309
9	11806	7284	11633	10766
10	11807	9386	11717	4557
11	11802	9653	11802	11151
12	11807	10594	11793	9434
13	11802	10848	10382	11805
14	11801	11551	7219	10237
15	11807	1701	7189	9338
16	11813	6203	11758	5321
17	11805	10720	10805	11748
18	9600	6371	11698	7818
19	11733	8160	11648	11509
20	9589	9428	11805	4813
21	11800	9463	11798	10801
22	11805	11799	11804	6029
23	11836	11655	11800	3579
24	11805	11512	11803	5761
25	11804	11573	11802	9680
26	11800	11575	11403	9388
27	11805	10691	11745	9465

The whole process of stress and performance tests, which took three days and about 1800 executions, was carried out without the need for monitoring of a test designer. The tool have selected automatically the next scenarios to be run up to the limit of six generations previously established.

The Table 2 presents the maximum fitnesse value obtained by the Hybrid Metaheuristc (HM), Genetic Algorithm (GA), Simulated Annealing (SA) and TABU Search (TS) in each generation.

The small number of samples of the experiment is insufficient to give a statistical significance with Wilcoxon procedure. However, it is noted that in 4 of 6 generations, the collaborative approach presented the best values. The experiment succeeded in finding 29 individuals where the maximum time expected by the application was obtained. The Table. **??** has a example of the six individuals with the highest fit values in the second experiment. The Table shows the fitnesse value (Fit),

Table 2: Results obtained from the second experiment

GEN	HM	TS	GA	SA
1	32242	32242	32242	32242
2	34599	32443	26290	35635
3	35800	34896	34584	34248
4	35782	34912	32689	25753
5	35611	31833	34631	8366
6	35362	35041	33397	9706

the name of scenario (Scenario), the number of users (N.Users), the percentiles of 90%,80% and 70% (90per, 80 per and 70per) in seconds.

Table 3: Example of individuals obtained in the second experiment

Ind	Fit	Scenario	N.Users	90per	80per	70per
1	35800	MyHome	31	30	29	10
		Badges	4			
2	35795	MyHome	30	30	29	10
		Notifications	2			
		Badges	2			
3	35782	MyHome	32	30	29	10
		Badges	3			
4	35773	MyHome	22	30	29	10
		Notifications	6			
		Badges	9			
5	35771	MyHome	28	30	29	9
		Badges	6			
6	35683	MyHome	27	30	29	8
		Badges	10			

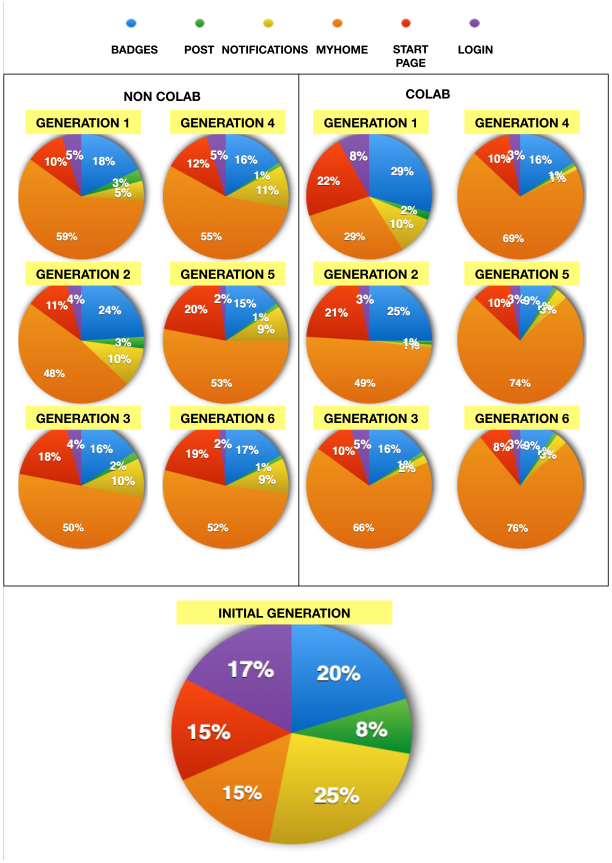
The Fig.

9. Conclusion

This paper presented a approach of use Hybrid Metaheuristc in load, performance and stress testing. Two experiments were performed to validate the solution. The first experiment has been applied in an emulated component and the second experiment has been applied in an installed Moodle application. The collaborative approach has obtained better fit values in both experiments.

The main contributions of the research are: The presentation of an approach that uses a hybrid metauristc to perform load, performance and stress tests; The development of a JMeter plugin to evolutionary tests and the automation of the load, performance or stress test execution process.

Figure 12: Best results obtained in 27 generations



Among the future work of the research, we can highlight the use of new combinatorial optimization algorithms such as Very large-scale neighborhood search.

Reference

[1] Z. Jiang, Automated analysis of load testing results, Ph.D. thesis, 2010.

[2] Z. Jiang, A. Hassan, Automated performance analysis of load tests, . . . , 2009. ICSM 2009. IEEE . . . (2009).

[3] I. Molyneaux, The Art of Application Performance Testing, "O'Reilly Media, Inc.", 2009.

[4] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, G. Weber, Realistic load testing of Web applications, in: Conference on Software Maintenance and Reengineering (CSMR'06).

[5] V. Garousi, A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation, IEEE Transactions on Software Engineering 36 (2010) 778–797.

[6] C. Babbar, N. Bajpai, D. Sarmah, Web Application Performance Analysis based on Component Load Testing, International Journal of Technology . . . (2011).

[7] A. Luiz, C. Freitas, O. Prof, R. Vieira, Ontologias para Teste de Desempenho de Software, Ph.D. thesis, Pontifícia Universidade Católica do Rio Grande do Sul, 2011.

[8] I. D. S. Fé, P. d. A. dos Santos, Os custos dos Testes de Desempenho e Estresse (2004).

[9] W. Afzal, R. Torkar, R. Feldt, A systematic review of search-based testing for non-functional system properties, Information and Software Technology 51 (2009) 957–976.

[10] M. O. Sullivan, S. Vössner, J. Wegener, D.-b. Ag, Testing Temporal Correctness of Real-Time Systems — A New Approach Using Genetic Algorithms and Cluster Analysis — (????) 1–20.

[11] C. Sandler, T. Badgett, T. Thomas, The Art of Software Testing (2004) 200.

[12] M. Corporation, Performance Testing Guidance for Web Applications, 2007.

[13] G. a. Di Lucca, A. R. Fasolino, Testing Web-based applications: The state of the art and future trends, Information and Software Technology 48 (2006) 1172–1186.

[14] D. G. Feitelson, Workload Modeling for Computer Systems Performance Evaluation, Cambridge University Press, 2013.

[15] M. C. Gonçalves, Um Processo de Inferência de Desempenho para Apoiar o Planejamento da Capacidade de Aplicações na Nuvem Um Processo de Inferência de Desempenho para Apoiar o Planejamento da Capacidade de Aplicações na Nuvem (2014).

[16] J. Puchinger, R. Raidl, Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization : A Survey and Classification, Artificial Intelligence and Knowledge Engineering Applications a Bioinspired Approach 3562 (2005) 41–53.

[17] C. Blum, Hybrid metaheuristics in combinatorial optimization: A tutorial, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7505 LNCS (2012) 1–10.

[18] A systematic review of search-based testing for non-functional system properties, Information and Software Technology 51 (2009) 957–976.

[19] N. J. Tracey, A search-based automated test-data generation framework for safety-critical software, Ph.D. thesis, Citeseer, 2000.

[20] V. Garousi, Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms, August, 2006.

[21] C. Barna, M. Litoiu, H. Ghanbari, Autonomic load-testing framework, . . . international conference on Autonomic . . . (2011) 91–100.

[22] D. Nevedrov, Using JMeter to Performance Test Web Services (2007) 1–11.