Automated System Testing Using Dynamic and Resource Restricted Clients

Mirko Caspar, Mirko Lippmann, Wolfram Hardt
Technische Universität Chemnitz
Faculty of Computer Science
09111 Chemnitz, Germany
Email: mirko.caspar | mirko.lippmann | hardt @ cs.tu-chemnitz.de

***Abstract*—Testing on system level using a static and homogeneous architecture of clients is common practice. This paper introduces a new approach to use a heterogeneous and dynamic set of resource restricted test clients for automated testing. Due to changing resources and availability of the clients, the test case distribution needs to be recalculated dynamically during the test execution. All necessary conditions and parameters are represented by a formal model. It is shown that the algorithmic problem of DYNAMIC TESTPARTITIONING can be solved in polynomial time by a heuristic recursive algorithm. A testbench architecture is introduced and by simulation it is shown that the testbench can execute the test requirements within a small variation using a number of several hundred clients. The system can react dynamically on changing resources and availability of the test clients within several seconds. The approach is generic and can be adapted to a huge number of systems.**

I. Introduction

In this paper, a new concept and realisation for automated testing by the usage of heterogeneous, dynamic, and resource restricted clients is introduced. It can be used for any kind of system under test (SUT) where the functionalities can be encapsulated as services. With this abstract black box view, the structure of the SUT is not important. An embedded Wifi-router can also be tested like a complete cellular mobile network consisting of many complex and interacting subsystems.

The principal operation is to define a time dependent load for every service of the SUT. These loads have to be partitioned to individual test tasks and mapped to the clients by a central testbench [1]. The clients are able to generate the according load by executing the test task and accessing the SUT services. If a task is finished or interrupted, the testbench is informed so that it can recalculate the load distribution among the clients.

The basic challenge is the indeterministic behaviour of the clients during the test runtime. It can happen that clients are not available for testing due to missing resources or even due to missing communication or energy (empty battery). Thus, it is necessary to calculate the usage of the different clients for the test tasks online during the test run.

The algorithmic solution of partitioning and mapping the loads to the clients is the important and challenging problem. It is shown that the according algorithmic problem, called DYNAMIC TESTPARTITIONING, is a non-linear, discrete optimisation problem. A heuristic algorithm is introduced to solve the problem in polynomial runtime. A testbench system was developed and used to validate the approach by measured results about accuracy and performance.

978-3-9815370-2-4/DATE14/©2014 EDAA

II. Related Work

Especially in the area of software testing, many concepts and tools exist. Test robots (a.k.a. Capture & Replay tools) are the simplest approach. Parameterised tests and scripting languages allow the adaptation to more dynamic systems (e.g. session based systems) and the distribution of different kinds of tests to different clients. Several test suites are available for this kind of automated test systems as well [2] [3]. Additionally, optimisations of these approaches to special SUTs have been published [4].

In opposite to these functional tests, non-functional performance tests aim to prove the performance, scalability and resource usage of system. It is necessary to generate many parallel requests to the system under test. The automation helps to coordinate the clients that execute the requests. Adaptations to the different concepts of the SUTs leads to different approaches for the automation tools and processes [5] [6] [7].

Special approaches for the performance test of grid systems are DiPerf [8] and ServMark [9]. The idea to abstract the distributed and heterogeneous grid system to its provided services is similar to the concept of this paper.

All mentioned concepts are not able to handle dynamic client sets that base up on heterogeneous, resource restricted platforms. Additionally, the automation systems use dedicated remote technologies (e.g. Remote Procedure Calls) or communication protocols (e.g. Hypertext Transfer Protocol).

III. Modelling and Algorithm

A SUT-independent formal model needs to be defined representing all conditions and parameters for the automated generation of load. Later, a dedicated test of a concrete SUT can be derived by creating an instance of this model.

A. Model and Problem Definition

The services of the SUT and the usage of these services by clients need to be modelled. The SUT is able to provide a set $D = \{d_1, ..., d_m\}$ of services d_j. A set $S = \{s_1, ..., s_m\}$ of clients s_i is available to use different (not necessarily all) services of the SUT. Accordingly, the implementation for the usage of the service d_j by the client s_i is stated as ${}_{s_i}\tilde{d}_j$ so that the set of all client service implementations is given as $\tilde{D} \subseteq (D \times S)$. Each client implementation works in a several operational mode $k_l \in K$.

Based on this sets, the relation *"using"* $\rhd$ can be defined:

$$\begin{aligned} \rhd &\subseteq ((\tilde{D} \times K) \times D) \\ {}_{s_i}\tilde{d}_{j\,k_l} &\blacktriangleright d_j \end{aligned}$$

The meaning of each element $\blacktriangleright$ is that client s_i uses its service implementation $\tilde{d}_j$ in the operational mode k_l to access the SUT service d_j.

As mentioned, it is necessary to quantify this *"using"*. The according value is called *"load"* l and is assigned to each element of $\rhd$:

$$load : \rhd \to \mathbb{N} \quad , \quad ({}_{s_i}\tilde{d}_{j\,k_l} \blacktriangleright d_j) \mapsto l$$

This description of services, clients, and loads are used to derive the requirements for a concrete test. It is the aim of a single test run to charge the different services of the SUT with a predefined load. The load is fluctuating in time and generated by the available clients and their according implementations.

Hence, a single test scenario F can be defined by a set of time depending functions:

$$F = \{f_1(t), f_2(t), ..., f_n(t)\} \quad , \quad f_i : T \to \mathbb{N}$$

Each function f_i describes the load that has to be generated on the service i at the given time t.

The overall load, defined by the function set F, needs to be generated totally by all clients. Several constraints needs to be considered due to the mentioned restrictions of the clients.

First, a client has restricted resources and cannot generate unlimited amounts of load for one or several services. It is given that ${}^t l_{s,i}$ is the load that is generated by client s on service i at the time t. There needs to be a maximum load $a_{s,i}$ describing the restrictions of the client so that at no point in time the actual load is greater than the maximum.

Furthermore, the generation of load for one service can restrict the usage of other services on the client. A typical example is GPRS[1] where the data link is not available during a voice call of the device. In the worst case the client device is not available for test execution, e.g. due to an empty battery.

To represent these aspects, a constraint matrix $\mathcal{G}$ is defined. Each element maps the time t and the actual load l for each client to 1 or 0 representing the (un)availability of it:

$$\mathcal{G} = \begin{pmatrix} g_{1,1}(t,(l_{1,1},...,l_{1,n})) & \dots & g_{1,n}(t,(l_{1,1},...,l_{1,n})) \\ \vdots & \ddots & \vdots \\ g_{m,1}(t,(l_{1,1},...,l_{1,n})) & \dots & g_{m,n}(t,(l_{1,1},...,l_{1,n})) \end{pmatrix}$$

$$g_{s,i} : T \times \mathbb{N}^n \to \{0,1\}, (t,(l_{s,1},...,l_{s,n})) \mapsto g_{s,i}$$

The automation of the test execution requires an algorithm that separates and maps the overall load to the available clients. The mapping needs to be as optimal as possible following predefined rules. These rules influence the usage of the clients in a significant way. The rules are modelled by cost functions in a cost matrix $\mathcal{C}$. It is similar to the constraint matrix $\mathcal{G}$ where each matrix element c is now defined as:

$$c_{s,i} : T \times \mathbb{N}^n \to \mathbb{R} \quad , \quad (t,(l_{s,1},...,l_{s,n})) \mapsto c_{s,i}$$

For each client s, the time t and the loads l_i of the different services i are mapped to a cost value.

Based on these representations of client capabilities, the algorithmic problem DYNAMIC TESTPARTITIONING can be described as an optimisation problem. The allocation of $l_{s,i}$ is sought so that the following objective function and the condition hold:

$$K = \sum_{s=1}^{m}\sum_{i=1}^{n} \left(c_{s,i}\left(t,(l_{s,1},...l_{s,n})\right) \cdot g_{s,i}\left(t,(l_{s,1},...l_{s,n})\right)\right)$$

$$\forall i \in [1,n] : \left| \sum_{s=1}^{m} \left(l_{s,i} \cdot g_{s,i}(t,(l_{s,1},...l_{s,n}))\right) - f_i(t) \right| < \varepsilon_i$$

$$\varepsilon_i > 0$$

The main problem is the non-linear and discontinuous solution space. This results from multiplication by the discrete value g and possibly non-linear cost functions c. Hence, there is no common algorithm known to solve this specific optimisation problem.

B. Heuristic Algorithm

Some constraints of the DYNAMIC TESTPARTITIONING have been used to develop a heuristic algorithm with polynomial runtime. All loads $l_{s,i}$ are defined as a natural number so that the optimisation problem can be treated as a combinatorial optimisation problem. Additionally, the algorithm is executed on some kind of digital computer so that the time is discrete as well.

In a typical application, the algorithm needs to be executed every time when the *wanted load* or the *actual load* of some clients change. In this case it is only necessary to redistribute the relative load difference and not the absolute load. Accordingly, the absolute load $l_{s,i}$ of the previous statements is substituted by a combination of actual load (${}^t l'_{s,i}$) and offset load (${}^t l^+_{s,i}$). The offset load can also have negative values.

Based on these simplifications the Heuristic Algorithm 1 is proposed. Basic aim is to determine and map an initial common offset ($average[i]$) for each available client. This is realised in the function $InitialAllocation()$. The value depends on the needed $offset[]$, the available clients and the strategy. An obvious approach is to calculate the average of each service i: $offset[i]/\#clients$.

This initial mapping is optimised independently for each service in the function $OptimiseService()$. It is tested for each pair (i,j) of clients, if the transfer of the mapped $average$ from i to j leads to lower overall costs. This is tested for all possible values of j. The $average$ load of i is transferred to the client j that leads to the highest cost improvement. Everything is iterated until no improvement is achievable or a maximum number of iterations ($MAX\_ITERATIONS$) is reached.

[1]General Packet Radio Service - packet based data transmission in GSM networks

Algorithm 1 DYNAMIC TESTPARTITIONING

```
procedure TestPartitioning($offset[], recursionlevel$)
    $recursion \leftarrow false$
    for all services $i$ do
        $(correction[i], average[i]) \leftarrow$ InitialAllocation($offset[i]$)
        if $correction[i] \neq 0$ $AND$ $correction[i] < offset[i]$ then
            $recursion \leftarrow true$
        end if
    end for
    for all services $i$ do
        OptimiseService($average[i]$)
    end for
    if $recursion = true$ $AND$ $recursionlevel < MAX\_RECURSIONS$ then
        TestPartitioning($correction, (recursionlevel + 1)$)
    end if
    return
end procedure
```

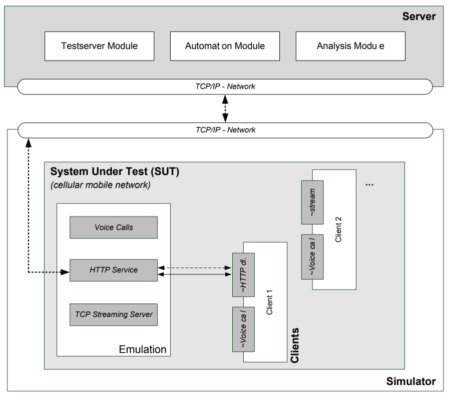


Fig. 1. Simulator Integration.

In some cases it is possible that not all load units can be mapped to a client. Typically this happens when the maximum load a of this client and service is reached. In this case, the difference is added to the value $correction[i]$. If this value is not equal to 0 for any service, all the algorithm is started again recursively with the array $correction[]$ as new $offset[]$. The maximum number of recursions is limited by $MAX\_RECURSIONS$.

$MAX\_ITERATIONS$ and $MAX\_RECURSIONS$ are constant values. Hence, the complexity of the algorithm is mainly influenced by the pairwise optimisation (the number of clients is m) in $OptimiseService()$ and the number of services n. On condition that the strategy for calculating $average[]$ has a complexity of $O(1)$ it holds that the overall heuristic algorithm has a complexity of $O(m^2 \cdot n)$.

IV. Architecture and Results

Based on the proposed algorithm, a testbench was developed allowing the automated testing of SUTs.

A. Testbench

The overall concept - especially the dynamic and heterogeneous client structure - leads to an testbench architecture that is separated in 4 different modules. The first one is the *test wrapper* running on each of the client devices. It works as agent between the central modules of the testbench and the service implementations of the according client. The wrapper needs to be implemented on every client. There needs to be different implementations in the case of heterogenous platforms.

All clients are connected to the *testserver module* by a communication system. The module maintains all information about the state, the availability, and the reachability of all clients. Furthermore, it manages the execution of all running test tasks on each client. All messages between the clients and the other parts of the testbench are routed by the testserver module.

The most important component is the *automation module*. Core is the dynamic testpartitioning algorithm presented in the previous section. It uses the test scenarios (wanted load) and information about running tests from the testserver module (actual load) and generates the load changes for each client. The additional testgeneration component creates according test task packets and transmits it to the testserver module. Optionally, it adds additional parameters that are necessary for the individual test.

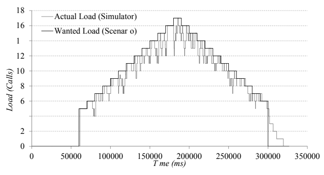
B. Simulation System and Results

The presented testbench was implemented to allow the evaluation of the concept and the system. The cellular mobile network have been selected as an exemplary SUT. It has a distinctive feature: the communication between the test wrappers and the testserver module is realised by the mobile data link (e.g. GPRS). As mentioned later, this link is one of the services under test. The cellular network, the mobile devices (clients), and the connections between it are realised in a simulator. An overview about the whole system is given in figure 1. The network simulator SimANet is used as platform [10]. Several extensions for the resource management had to be implemented before running the evaluation [11].

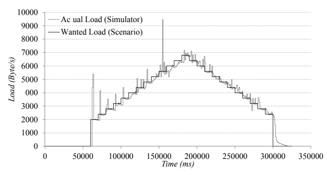
The cellular network related services ”voice call”, ”TCP-stream” and ”HTTP-download” are used for evaluation. All 3 services were evaluated independently. The scenario defines a wanted load between 0 and 17 load units. It is increased every 10 seconds. After the maximum is reached, it is decreased to 0 again. In the first and the last step the load is increased or decreased by 5 or 6 units. 20 clients are available during all the runtime. Each client can run only 1 voice call per time ($a_{s,1} = 1$). The maximum loads for the other services and each client are selected randomly for each client in predefined limits.

The results of the test for the voice call and the TCP-stream service are shown in figure 2. The actual load in 2(a) follows the wanted load besides some lower peaks. The peaks results from the delay between a finished and reported voice call of a client and the recalculation by the automation module. Additionally, there is a delay at the end of the test run. It results from the fact, that running voice calls cannot be stopped by the automation module.

Unlike voice calls, the TCP-stream service allows the setting of exact streaming rates of each client. As visualised in figure 2(b), the actual load corresponds better with the wanted load. The small differences are caused by inaccuracies of the streaming rates. The 2 main peaks are a result of a not clarified anomaly in the TCP-stack of the underlying operating system (MS Windows) hosting the simulation.

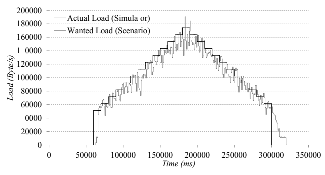


(a) Voice call service.

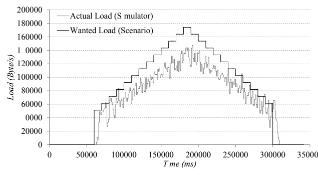


(b) TCP-streaming service.

Fig. 2. Comparison of wanted and actual load.



(a) Different file sizes > 100kByte.



(b) Different file sizes < 100kByte.

Fig. 3. Comparison of wanted and actual load of HTTP-download service.

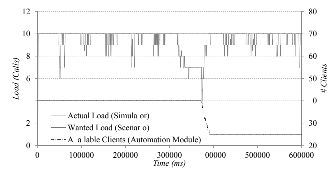
The analytical evaluation of the measured values shows that the ratio of actual load and wanted load is 0.913 for the voice call service and 1.01 for the streaming service. The average delay between changes of the wanted and the according adaptation of actual load is 0.8s respectively 2.1s.

The load generation of the HTTP-download service is depending on the file sizes. It is clearly observable that the differences of wanted and actual load are higher in the case of lower file sizes. The wanted load cannot even be reached for higher loads. The main reason is that the delays for starting new downloads are higher than the time to finish the download of small files. This kind of combination is a worse case for this kind of automated test concept and the presented architecture.

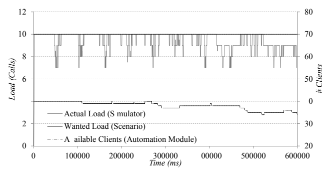
The analytical evaluation shows a relation of actual and wanted load for the small files of 0.8 and for the big files of 0.95. The delays for one step of load increasing are 5.1s and 2.2s. In the first case it is higher since more clients have to be ordered for creating new loads.

One main requirement for this work is the capability to handle clients dynamically. The previous results base up on a fixed number of clients. In contrast, results for the adaptation to a changing number of clients can be found in figure 4. The initial number of clients is increased to 40. In the first case, 15 randomly selected clients are removed from the network after 300s of test runtime. The according behaviour can be seen in the figure 4(a). The actual load decreases due to the missing clients. After around 90s the testserver module and the automation module react and generate new load by the existing clients. The reason for the relatively long delay is the missing report of the deactivated clients. Immediately after the communication between clients and testserver module is timed out (light, dotted curve), the automation module adapts to the changes and generates new load by available clients.

The second measurement introduces a movement simulation for the nodes. During the runtime the nodes might leave the simulated mobile network so that the communication with the testserver module is not possible anymore. In this case, the dynamics are much smoother. Accordingly, the differences between wanted and actual load are not as high as in the first case. The behaviour is similar to the previous case.



(a) Reference.



(b) Random.

Fig. 4. Dynamic adaptation to changing client availability.

V. Conclusion and Outlook

A solution for automated testing of service providing systems was introduced. It supports the usage of dynamic and heterogenous sets of clients. A test is executed by defining abstract loads that have to be generated by the clients on the services of the SUT. A testbench architecture was presented. It allows the maintaining of the client sets and the automatic partitioning and mapping of service loads to the clients. An adapted heuristic algorithm was introduced. It allows the solution of the DYNAMIC TESTPARTITIONING - an optimisation problem - in polynomial runtime. The testbench and the algorithm were evaluated using a simulation platform. It was shown, that it is possible to match the wanted loads with overall differences of 5 to 10% (worst case: 20%) and delays of maximal several seconds. The algorithm is able to handle several hundreds of clients. Presently, there are still problems about the delays for detecting missing clients. TCP timeouts are currently used. This concepts is communication platform dependent and results in too long delays.

References

[1] M. Caspar, M. Vodel, and W. Hardt, "System level test of service-based systems by automated and dynamic load partitioning and distribution," in *Proceedings of the 10th International Conference on Innovative Internet Community Systems (I2CS2010)*, ACM Press. Bangkok, Thailand: ACM Press, June 2010.

[2] C. Rankin, "The software testing automation framework," *IBM Syst. J.*, vol. 41, no. 1, pp. 126–139, Jan. 2002.

[3] R. P. Singh, "Managing software testing automation framework with rational quality manager," IBM, Tech. Rep., August 2012.

[4] S. Dustdar and S. Haslinger, "Testing of service-oriented architectures - a practical approach," in *Net.ObjectDays*, 2004, pp. 97–109.

[5] D. Krishnamurthy, J. A. Rolia, and S. Majumdar, "A synthetic workload generation technique for stress testing session-based systems," *IEEE Trans. Softw. Eng.*, vol. 32, pp. 868–882, November 2006.

[6] S. Gaisbauer, J. Kirschnick, N. Edwards, and J. Rolia, "Vats: Virtualized-aware automated test service," in *QEST*, 2008, pp. 93–102.

[7] J. Zhang, "A mobile agent-based tool supporting web services testing," *Wirel. Pers. Commun.*, vol. 56, no. 1, pp. 147–172, Jan. 2011.

[8] M. I. Andreica, N. Tapus, C. Dumitrescu, A. Iosup, D. Epema, I. Raicu, I. Foster, and M. Ripeanu, "Towards ServMark, an Architecture for Testing Grid Services," *Technical University of Delft - Technical Report*, Jul. 2006.

[9] C. Dumitrescu, I. Raicu, M. Ripeanu, and I. Foster, "Diperf: An automated distributed performance testing framework," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, ser. GRID '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 289–296.

[10] M. Vodel, M. Sauppe, M. Caspar, and W. Hardt, "The simanet framework," in *Proceedings of the 1st International Conference on M4D: Mobile Communication Technology For Development*. Karlstad, Schweden: Karlstad University, December 2008, pp. 88–97.

[11] M. Caspar, *Lastgetriebene Validierung Dienstbereitstellender Systeme*, ser. Eingebette, Selbstorganisierende Systeme, W. Hardt, Ed. Universitätverlag Chemnitz, 2013, vol. 11.