System Level Test of Service-based Systems by Automated and Dynamic Load Partitioning and Distribution

Mirko Caspar, Matthias Vodel, Wolfram Hardt

Faculty of Computer Science, Chemnitz University of Technology, Germany
mica | vodel | hardt @cs.tu-chemnitz.de

Abstract: Load generation is a suitable concept for system level test of complex systems. We present a novel approach especially for service based systems. Test scenarios are defined as inputs and describe a quantification of the wanted load to a service. A test-automation component partitions and distributes these abstract load values to a set of clients. These clients execute service requests to generate the calculated load. Since the partition and distribution are done during the test runtime, it can handle a dynamic set of clients with fluctuating resources. The proposed framework will be able to test a wide range of systems. It is aimed to run system level tests under laboratory and field test conditions.

1 Introduction

The increasing complexity of systems causes a big challenge for all steps of the design process. Hence, not only the specification and implementation of a new system is difficult and complex but also the steps of testing. Many test paradigms aim not to use expert knowledge about the implementation of a module but to define specification related test cases. On the lower levels of the design process the specification of single modules can be very detailed. This allows the easy generation of suitable test cases. In contrast, the specification on the high level - especially on system level - is abstract and imprecise. The derivation of suitable test cases is difficult and needs experience. The execution of the test may be extensive.

It is some kind of best practice for laboratory system level tests to generate load for the system under test (SUT) by client devices. The test clients are part of dedicated test stations or allocated in a wide area and connected by a wired network. In most cases, test scripts for static or random scenarios are used. These scripts are not able to react dynamically to changes in the client infrastructure in a controllable or meaningful way.

Furthermore, field tests are used to check the SUT in a real world scenario. A restricted user group is allowed to use the system as it is intended to be used. The test cases are generated implicitly by the users, since they use the service of the SUT for daily work. The test engineers are completely dependent on the user behaviour and can hardly influence the generated test cases, respectively the load to the system.
In this paper we present a concept and a constitutive framework which supports the test engineer and solve the mentioned disadvantages of field / laboratory tests on system level. Our approach is optimised to test service providing systems, where the SUT offers any kind of well defined services for clients. The availability of the clients and their resources may change during runtime. For test support, we extended the well-known concept of load generation. Popular approaches base up on static scripts. In opposite, we aim to generate suitable test tasks for each test client automatically. The generation is done dynamically during the runtime and uses abstract test scenario functions as inputs. Another component maintains the set of clients and distributes the test tasks among them. The presented work is in state of proof of concept.

2 Classification and Related Work

A popular model for the description of the development and test process is the V-model [FG99]. It describes that development and testing are done within different levels of implementation/integration. The second important conclusion is that the specification of a test is done during the development in the according level. Hence, details about the implementation on lower levels cannot influence the test specification. On the other hand the executed test can only be used to check the system implementation of its own level.

Even the V-model has been proposed in the context of software engineering, the main ideas can be used for the realisation of all kinds of information systems. Our approach is placed on the level of system test and above. It is a framework to generate load on the SUT and to report about success or failures. The detailed analysis must be done by checking traces and performance keys of the SUT itself.

A lot of work has been done in the area of testing. We concentrate our inquest to the area of formal approaches for test case and load generation on system level.

IBM presents a linear programming test case generation for SoC [NSZ06]. The functional dependencies are modelled by a linear program to limit the set of test cases. Soft constraints are introduced to model random tests. The usage of system is restricted to generate binary vectors as test cases. The extension to complex service based systems is not useful.

Krishnamurthy et. al. present a system to test session based server architectures [KRM06]. They use scripts of user inputs to generate static load scenarios for the SUT. The load itself is generated locally on the server, so that no clients are used. The idea to use linear programming for the generation of ‘good’ test cases is interesting. All test cases are generated as preparation for the test. A dynamic adoption during test time is not provided.

ServMark is an approach related to ours. It is a framework for performance tests of grid systems and webservises. It is composed of DiPerf [DRR+04] and GrenchMark [IE06] and is set up on PC-based test clients that are connected by a wired network. Remote
procedure calls (RPC) are used by the test server to start tests on the clients. In contrast to our approach, the set of clients is fixed and cannot be changed during the test. A failure of a client causes the failure of the test. Additionally, ServMark is completely adapted to the test of grid systems.

3 Structure of Test System

As mentioned in the introduction, our approach is designed for testing service providing systems (SPS). An SPS is a system with a set of resources which are encapsulated by well defined interfaces. Requests of service consuming clients to these interfaces are processed and responded by the according service implementation. For our approach it is necessary that the load of each service is quantifiable. This means that there is a meaningful value $l_i \in \mathbb{N}$ for the service $i$ that describes the load for this service.

An easy example for an SPS is a computer running some server services, like a HTTP server and a network file system. Both services provide a well defined interface - the protocol. Clients with a routed network connection to this server can use the services. The loads $l$ can be defined differently, depending on the situation. For example the load of the HTTP can be the number of open connections and the load of the NFS can be the current upload rates (to the clients).

Beside this simple example, more complex scenarios are possible. For instance, a whole cellular network can be seen as SPS. The entire infrastructure that is necessary to provide a mobile network is part of the SUT: antennas, hardware, software and the network infrastructure. The provided services may be voice calls, data calls and message services. The mobile phones represent the test clients.

This example illustrates the problems that have to be solved. Mobile phones are small devices with limited resources. They can lose the connection to the mobile network or run out of energy so that they cannot be used as test clients temporarily. In this section we will describe our approach for the test system and its components. It is illustrated in figure 1.

![Figure 1: Overview over the whole test system structure.](image)

The basic idea of our concept is to use the clients to generate a controllable number of requests to the SUT services. A central test server controls the clients and sends test tasks
to them. So the test of SPS can be influenced during runtime. In opposite to the approaches presented in section 2 we have to consider that the client architecture is not static. It may change during the run of the test. Furthermore, the available resources of the client devices may differ. Not every client is able to handle a special test task at any time. To use the clients remotely, they need to run a control unit which can receive test tasks and can start requests to the services of the SUT.

A test may consist of hundreds of test clients. A manual control of this set is not possible for a human test engineer. In consequence it is essential to provide the engineer a more abstract view and definition of the test. Hence, we define a test for a single service as a function of quantified load \( l \) depending on time. This describes how many requests for a service have to be executed by the clients at a defined time.

Obviously, this value of load has to be partitioned and distributed to test tasks for a set of test clients. The availability of each client and its available resources must be considered for the calculation. Hence, a simple and popular script based approach is not suitable anymore. An algorithm has to be implemented which is able to calculate the distribution based on the available client resources. Details about the model and the algorithm will be given in the next section.

Usually, some kind of communication network is necessary for the clients to communicate with the SUT. This can be used for the communication of the test server with the clients too. In special cases, the communication channel can also be part of the SUT. An example is the cellular network infrastructure that was mentioned above. The data links of this network can be used for communication between clients and test server even if they are a part of the services that has to be tested. This scenario leads to some special requirements that are not mentioned in detail here.

Since our concept aims for the load generation for black box systems, we do not have any information about the success or the performance of tests from the SUT itself. We introduce state and result messages which are sent from the clients to the test server. The test server manages the received messages and calculates the states of the test tasks and the test client. Additionally, all results are stored in a database too to allow an analysis of the test.

Furthermore, state changes of test tasks are reported to the test-automation system as well as information about the availability of the clients. This information is needed for the calculation of the next test tasks and the distribution to the clients.

A test-analysis component evaluates the results of the tests and calculates overall results that can be visualised for the test engineer.
4 Model and Algorithm

The test-automation component calculates the test tasks that are mapped to available clients. It is a generic part of the test system which is not adopted for the test of a special SUT. Hence, it needs two inputs as description of a concrete test: a model of the SUT and the conditions of the test itself.

The definition of the test is given by 2 sets. First, a set of functions \( \{ f_1(t), \ldots, f_n(t) \} \) where each one describes the wanted load \( l \) in time for a service. The function \( f \) for the service \( i \) is called test-scenario and defined by \( f_i : T \rightarrow N \). Secondly, the test engineer has to define which client is able use which service at what point of time. It is defined as a set of functions \( \{ g_1(s, t), \ldots, g_n(s, t) \} \). The function \( g \) for the service \( i \) is defined as \( g_i : S \times T \rightarrow \{0, 1\} \), where \( S \) is the set of known clients. Obviously, the value 0 for a client \( i \) means that it shall not take part on the test. This set of functions can be defined manually by the test engineer or automatically generated, e.g. based on a random distribution.

The SUT itself cannot be described directly since it is a black box test and information about the SUT are rare. The only relevant information are the available services \( 1, \ldots, n \) that have to be tested. It is important for the test system to describe how the clients can use these services. Mainly, this is done by the implementation on the clients. It realises the necessary technologies and protocols to use the services.

The test-automation has to calculate the quantification of service load that has to be started by each client. Therefore, we have to model the resources of a client and optionally a simulated user behaviour. This can be easily described by a set of 2-dimensional cost functions \( \{ c_{s,1}(l_1, t), \ldots, c_{s,n}(l_n, t) \} \) for each client \( s \). Each function is depending on time and the load quantification \( l_i \) (introduced in section 3). They have to be defined for each service and each client: \( c_{s,i} : N \times T \rightarrow \mathbb{R} \), where \( T \) is the time. Hence, \(|S| \cdot n \) cost functions have to be defined. They can also be equal for classes of client devices.

Furthermore, it is necessary to define the maximum load that can be generated by a client for a service. A vector \( \{ a_{s,1}, \ldots, a_{s,n} \} \) defines the absolute values of client \( s \) for all services \( 1, \ldots, n \).

With the definition of this model and the test functions \( f \) and \( g \) the test-automation is able to generate test tasks for each available client. In detail, we are looking for a vector \( \{ l_1, i, \ldots, l_n, i \} \) of natural numbers, that defines the generated amount of load for each client \( 1, \ldots, n \) and the service \( i \).

Necessarily, the runtime of the used algorithm will cause a discrete time system of the whole test process. For this reason the calculated load vectors must be considered as time dependent. To express this, the elements of the mentioned solution vector \( l \) will be tagged with the time index: \( \{ t_{l_1, i}, \ldots, t_{l_n, i} \} \). Since clients can get temporarily unavailable during the test (lose resources) this has to be modelled too. In addition to the input vector \( g \) the \(|S| \cdot n \) matrix \( G' = (g'_{s,i}) \) is used to provide this information to the algorithm. The value
The content of the matrix is maintained by status messages from the test-infrastructure.

The costs of the solution shall be optimised concerning the given cost functions \( c_{s,i} \). The following equations and inequalities describe the common optimisation problem that has to be solved.

\[
(t' l_{s,i}) \in N : \sum_{s \in S} \sum_{i=0}^{n} (c_{s,i} ((t l_{s,i} + t' l'_{s,i}), t) \cdot g'_{s,i} \cdot g(t(s,t))) = \min! \quad (1)
\]

\[
\forall s \in S, i \in [1, n] : t l_{s,i} + t' l'_{s,i} \leq a_{s,i} \quad (2)
\]

\[
\forall i \in [1, n] : \left| \sum_{s \in S} ((t l_{s,i} + t' l'_{s,i}) \cdot g'_{s,i} \cdot g(t(s,t))) - f(t) \right| \leq \varepsilon_i \quad (3)
\]

Opposite to the mentioned absolute load value \( t l_{s,i} \) for each client we are now looking for a relative value \( t' l'_{s,i} \). It describes the alteration of client load for the service \( i \). This separation is necessary due to possible technical restrictions of the clients. For example, the alteration may be limited to adding more requests since the implementation does not allow cancelling requests.

The equations (2) and (3) describe the constraints that have to be satisfied by a correct solution of the equation system. On the one hand it has to be checked that the calculated value is not greater than the maximal allowed load value for this client. On the other hand the summation of all calculated values of a service \( i \) has to be equal to the wanted service load \( f_i(t) \) (considering an uncertainty of \( \varepsilon \)).

We have chosen 3 different kinds of algorithms with different grades of optimisation accuracy for further analysis.

The first algorithmic approach is a linear program. Since we do not restrict the functions \( c_{s,i} \), we cannot guarantee a convex solution space for the equation (1). A new target function has to be deviated to get a linear program. Hereby, the costs can only be integrated as a linear combination with the values \( t' l' \), as given now with equations (4) and (5).

\[
(t' l'_{s,i}) \in N : \sum_{s \in S} \sum_{i=0}^{n} (d_{s,i} \cdot t' l'_{s,i}) = \min! \quad (4)
\]

\[
d_{s,i} = c_{s,i} (t l_{s,i}) \cdot g'_{s,i} \cdot g(t(s,t)) \quad (5)
\]
Since the co-domain of \( t \) is discrete, the given problem is an integer linear program (ILP) which is known as NP-hard. To solve the problem in polynomial time, we decided to use LP-relaxation [HO02] and to cast the results to integer values.

The ILP model has the serious restriction, that the costs can only be modelled by a linear factor whereas the daily testing practice may generate non linear and even discontinuous functions. We decided to develop a heuristic algorithm that allows to consider any kind of cost functions and to calculate satisfying results in polynomial time. We are using the fact that the whole system is necessarily time discrete. Changes in the input values of the equations are derived from status messages of the clients or from changes in the function value \( f_i \). Furthermore we introduce the restriction, that every cost function \( c_{s,i} \) for a service \( i \) does not depend on the load value of any other service \( j \). Under these conditions, load changes can be handled for a single service.

Since the described test automation problem is similar to partitioning problems we map it to a hierarchical-clustering algorithm [GVN+94]. After a load change \( \Delta l_i \) has occurred, it is separated in pieces of equal size and mapped to every available node. A closeness-matrix is calculated where the value \( a_{i,j} \) expresses the difference of costs when the new piece of load is transferred from client \( i \) to \( j \). The operation that leads to maximum saving is executed and the closeness-matrix is updated. This iterates until no more optimisation can be reached. The complexity of this algorithm is \( O(n^2) \) which results mainly from the generation of the matrix.

As a third type of algorithm we are going to use a random distribution. It does not consider the cost minimisation of equation (1). Independently, the constraints of equations (2) and (3) must be fulfilled. This approach will be used to analyse the test quality of randomised partitioning and distribution of test scenarios.

5 Discussion

In this section we want to expose the advantages and prospects as well as some limitations of our approach. We will present our simulation concept to proof and optimise the system.

5.1 Concept

The most important advance of our approach is the ability to handle a dynamic client structure and heterogeneous resources of these clients. It is realised by the dynamic recalculation of the necessary load values which can take account on the availability of clients or resources. Furthermore, this changing availability can also be simulated in laboratory tests by the accordant calculation of \( g \) (see section 4). If all secondary conditions, like user behaviour or resource availability, are deterministic, the test can be reproduced.

On the other hand there are some limitations due to the proposed model, the algorithms
and technological conditions. The most distinguished limitation is the time discretisation that is caused by the runtimes of algorithms and messages. This leads to time periods on each client, which are not controllable in detail. Hence, differences between the wanted load (given by the test scenarios \( f \)) and the time actual real load may occur.

Technological restriction on the clients may lead to a second problem of accuracy. We know services, where load can only be generated indirectly. E.g. network traffic may only be generated by downloading files. Hence, the real load value \( l \) that is generated by a client is a function of time and depending on the available resources of the service itself, the network and the client resources. If this value cannot be measured and reported from the client to the test system, the value has to be estimated.

5.2 Implementation and Simulation

Our system is able to manage several hundreds of test clients. It is difficult to get access to so many 'real' clients. So we decided to use a special adoption of a simulation environment to check our concept, evaluate parameters and show the performance.

The basis is SimANet \([VSC^+08]\), a modular and extensible simulation environment. It is used to simulate the behaviour of huge wireless networks with different wireless communication standards \([VCH10]\). SimANet has been optimised to run on parallel systems to increase the simulation performance. Movement models are available to change the positions of network nodes.

We implemented the mentioned cellular network example. The clients and the mobile network will be simulated in SimANet. The automation- and infrastructure-system are implemented as a server that is communicating with the simulation. Each simulated client runs a program to start service requests and to communicate with the test server by virtual sockets of the simulated network.

To realise this, some enhancements of SimANet were necessary. This contains the implementation of an access point mode to simulate the behaviour of a cellular network. Furthermore, a capability to allow clients the allocation of all necessary services or resources is implemented now. Hence, we have the possibility, to maintain and check the current load to each service easily.

The implementation of the test-infrastructure component is finished. It is responsible to maintain a list of available clients, send test tasks to the clients and receive result messages from them. To test this implementation, we simulated up to 100 clients that has to be controlled and maintained by this infrastructure component. Just by the possibility to simulate so many parallel clients, we were able to find some critical errors in our test-infrastructure component. Most of them were related to parallel access to data structures. Due to this success we are confident that the simulation approach is feasible.
The test tasks are generated by static scripts since the proposed algorithms are in an evaluation process. The LP model is implemented as a Matlab program whereas the clustering heuristic is a dedicated Java-program. First runs of the heuristic algorithm show a good runtime but weaknesses in the optimisation results.

5.3 Timing

As explained above, the timing behaviour of all the test process is important for the quality of the load generation process. For this reason it is one of the most important properties that have to be analysed and optimised by the implemented simulation. The significant delays are shown in figure 2 (A).

![Timing diagram](image)

Based on this timing model, two important values can be derived. The cycle time \( t_{cycle} \) is responsible for the update speed of calculation and hence for the accuracy of the load generation. The \( t_{latency} \) is the delay that occurs after the calculation has been started until the clients execute the new calculated service requests. The cycle and latency of the system can be derived as follows:

\[
\begin{align*}
  t_{cycle} &= \max(\lambda_1, \ldots, \lambda_6, \eta_1, \ldots \eta_4) \\
  t_{latency} &= \sum_{i=1}^{4} \lambda_i + \sum_{i=1}^{2} \eta_i
\end{align*}
\]

Obviously, \( \lambda_1 \) is the runtime for the test-automation algorithm. Most likely, it will be the maximum value of all delays so that it is the basis for the interval time \( t_{cycle} \) of our time discrete system. It must be the aim to keep it as small as possible.

The other delays down to \( \lambda_4 \) are not eminent for the performance of the system. They are part of a kind of pipeline that has to be passed by each generated test tasks. The
summation of these delays is the time that passes until the generated test task is started as service request. Parallel to the transmission of a test tasks, status and report messages are sent to the automation system where they have to be evaluated. The delays are mainly depending on the communication and server system and not part of a critical path.

Figure 2 (B) illustrates the influences of $t_{cycle}$ and $t_{latency}$ to the generated load. The cycle leads to periods of time, where the load is constant and cannot be adopted to the wanted load, given by $f_i$. The latency may lead to a difference between the wanted and the generated load at a point of time.

6 Conclusion

We presented an approach for the automated and dynamic load generation to support the system level test of service based systems. The primary objective deals to control a set of test clients from a central test server. This test server generates test tasks for all the clients based up on test scenarios. A test-infrastructure component transmits the tasks to the target client and manages status and result messages from the clients. The clients generate quantified amounts of service requests to the SUT following the instructions of the test tasks. In opposite to existing approaches, we are able to manage a heterogeneous and dynamic set of clients with fluctuating resources. Central component of the system is the test-automation. It has to partition and allocate the test scenarios to concrete test tasks.

Basic parts of the framework, like the test-infrastructure component, are finished. The work for the test-automation framework has been started whereas the LP model and the heuristic algorithm are already implemented. It is part of continuative work to improve the quality of the heuristic algorithm to find better solutions for the test-partitioning problem. With the framework and the presented simulation platform we are going to compare the performance of the LP, the heuristic and a random based algorithm.

When the system is stable with the given algorithms, we are going to analyse some prediction models to improve the mentioned inaccuracies caused by the latencies and the cycle.

Beside the area of testing, we noticed that there are similar algorithmic problems in the area of parallel computing. Load balancing aims to map tasks to the processors during runtime [SKH95] [MCS+09]. The result should be that no processor is idle when there are available tasks. Further research work will analyse if parts of our work can be used for this problem too.

References


