12th International IEEE Enterprise Distributed Object Computing Conference

Modeling a Realistic Workload for Performance Testing

Christof Lutteroth, Gerald Weber
Department of Computer Science
The University of Auckland
38 Princes Street, Auckland 1020, New Zealand
Email: {lutteroth, gerald}@cs.auckland.ac.nz

***Abstract*—Load testing of web applications can be specified by simulating realistic user behavior with stochastic form-oriented analysis models. Stochastic models have advantages over load test models that simply play back recorded session data: they are easier to specify and achieve a higher coverage of the different operational paths. There are challenges when specifying load tests such as the generation of form parameters and the recognition of pages returned by the system. We propose how these challenges can be overcome by adding additional specifications to a form-oriented model. Furthermore, we discuss several workload models and explain why some commonly used workload models are in fact unrealistic and produce misleading results. The stochastic form-oriented load testing approach can be generalized to deal with other submit-response systems such as those consisting of web services.**

I. Introduction

Web applications are ubiquitous and need to deal with a large number of users. Due to their exposure to end users, especially customers, web applications have to be fast and reliable, as well as up-to-date. However, delays during the usage of the Internet are common and have been the focus of interest in different studies [4], [6]. The demands on a web site can change very rapidly due to different factors, such as visibility in search engines or on other web sites. Load testing is thus an important practice for making sure a web site meets those demands and for optimizing its different components [3]. The same holds true for web services in a service oriented architecture, which can also be subject to varying load over time. More and more, web service interfaces will be accessed as a direct consequence of user requests, and thus these web services will be often subject to similar usage patterns as web interfaces.

Raj Jain points out a number of common mistakes people do when performing load tests [15]. One of them is using an unrepresentative workload: "the workload used to compare two systems should be representative of the actual usage of the system in the field". In this paper, we address this issue by i) describing how to specify load test models using a stochastic approach and ii) analyzing the shortcomings of frequently (mis-)used workload parameters.

Our approach applies the methodology of form-oriented analysis [10], in which user interaction with a submit-response style system is modeled as a bipartite state transition diagram. The model used in form-oriented analysis is technology-independent and suitable for the description of user behavior. It describes what system output a user sees, and what he or she provides as input to the system. In order to simulate realistic users we have extended the model with stochastic functions that describe navigation, time delays and user input [7]. The level of detail of the stochastic model can be adjusted incrementally through refinement. Although we discuss load testing in the context of web applications, the same models can be used for other types of submit-response systems such as those based on SOA, e.g. for load testing of web services composition [1].

Section II gives an overview of the form-oriented analysis model, which serves as the basis for load testing specifications. Section III extends this model so that stochastic load test model can be formulated, explains the advantages of stochastic load testing and points out the importance of realism. Section IV discusses some of the challenges of load testing specification, and proposes solutions for specifying the generation of form parameters, recognition of pages, session drop out and server unavailability. Section V explains why some popular workload models are unsuitable and actually produce misleading load test results. Section VI discusses related work, and Section VII concludes the paper.

II. The Form-Oriented Model

Form-oriented analysis [10] is a methodology for the specification of ultra-thin client based systems. Form-oriented models describe a web application as a typed, bipartite state machine which consists of *pages*, *actions* and transitions between them. Pages can be understood as sets of *screens*, which are single instances of a particular page as they are seen by the user in the web browser. The screens of a page are conceptually similar, but their content may vary, e.g. in the different instances of the welcome page of a system, which may look different depending on the user. Each page contains an arbitrary number of *forms*, which in turn can have an arbitrary number of *fields*. The fields of forms usually allow users to enter information, and each form offers a way to submit the information that has been entered into its fields to the system. A submission invokes an action on the server side, which processes the submitted information and returns a new screen to the client in response. Hyperlinks are forms with no fields or only fields that are hidden to the user.

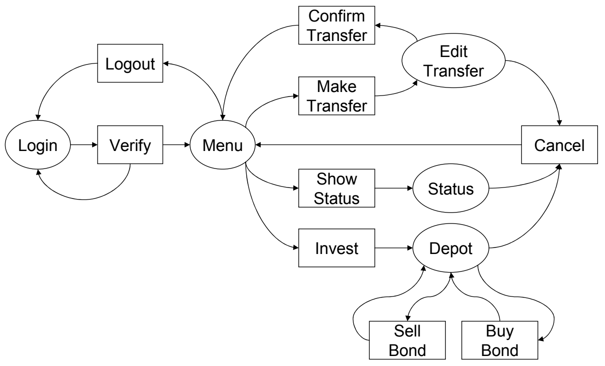

Fig. 1. Formchart of example home banking web application.

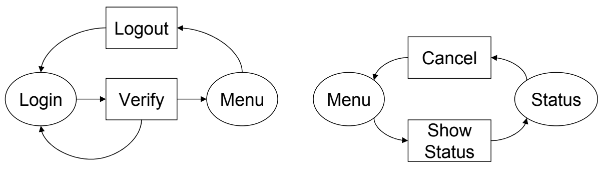

Fig. 2. Two features of the home banking application.

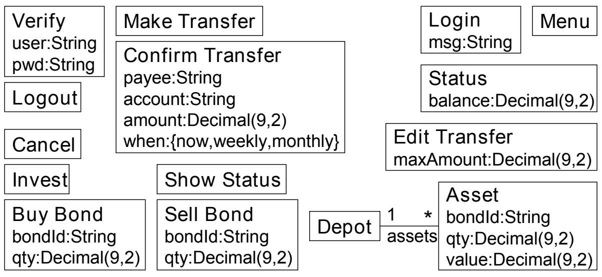

Fig. 3. Message model for the home banking application.

Form-oriented models can be visualized using *formcharts*. In a formchart the pages are represented as bubbles and the actions as boxes, while the transitions between them are represented as arrows, forming a directed graph. Formcharts are bipartite directed graphs, meaning that on each path pages and actions occur alternatingly. This partitioning of states and transitions creates a convenient distinction between system side and user side: the page-action transitions always express user behavior, while the action-page transitions always express system behavior.

In Fig. 1 we see the formchart of a simple home banking system, which will be the running example of this paper. The system starts showing page Login to a user, who can enter a username and a password. These data are submitted to action Verify, which checks if they are correct and either redirects the user back to the Login page or to the Menu page of the home banking system. Here the user can access the different functions, i.e. showing the account's status, making transfers, trading bonds, and logging out. Each of the functions may involve different subsystems and make use of different technical resources.

Formcharts can easily be decomposed into submodels, which we call features. This is illustrated in Fig. 2, which shows two features of the home banking application. The feature on the left side describes the login and logout procedure; the feature on the right side describes the status function of the application. As we can see, the features overlap, i.e. they both contain the Menu page, which means that we can combine the features in a meaningful way. Merging all features together results in the complete formchart of a system. Formchart diagrams are very flexible and it is, for example, possible to represent the same page or action several times, i.e. as several bubbles or boxes. This can be convenient for avoiding crossing transition arrows. It illustrates the set semantics of a formchart: elements can be represented several times with the formal form-oriented model remaining the same, and form-oriented models can be combined using simple set union.

In addition to the formchart artifacts, a form-oriented model also specifies message types for all the pages and actions. In the context of load testing, the message types for actions are useful because they specify the structure of the data that can be submitted through a form or link. This information helps us to generate form parameters during load testing. The message types for the pages specify the structure of the dynamic information represented on a page, and they are useful because we often need this information during load testing in order to choose appropriate form parameters.

Figure 3 shows the message model for the home banking example. The action types are on the left side of the figure, and the page types on the right side. Types of actions that are invoked by simple links such as Logout or Invest do not have any data fields. The actions that are invoked by forms such as Confirm Transfer have a field for each of the input fields in the form. We also allow ad-hoc types such as the "when" field of Confirm Transfer, which are very useful for representing single selections. If a page always looks the same, i.e. if there is no dynamic information content, then the corresponding message type does not contain any field, such as the type for page Menu. If a page contains a list of structured items, then additional types can be defined. In our example, the list of assets shown on the Depot page is modeled as an additional type Asset with a many-to-one association to type Depot.

A form-oriented model of a web application offers several benefits. It is suitable for testing as well as for the analysis and development of dynamic web applications. A typical shortcoming of many other models is that they do not capture fan-out of server actions, i.e. the ability of a server action to deliver many conceptually different client pages, which is covered by the form-oriented model. There are software development tools supporting the use of form-oriented models, including the stochastic models described in this paper, e.g. for forward engineering [11], reverse engineering [8], [9], and load testing [5], [7]. A form oriented model is also a good way to represent complex state-dependent web service interfaces. For the purpose of load testing, formcharts can be used as initial test cases for finding performance bottlenecks in web service orchestration scenarios.

III. Stochastic Formcharts

Formcharts specify web applications, which usually work in a strictly deterministic manner. In a load testing scenario, however, the system under test already exists, and the problem is to simulate the behavior of a large number of users. But just as a formchart is a specification of the web application, it is also a specification of possible user behavior; and while it is the web application that chooses in an action which page will come next and which data will be shown on the page, it is the user who chooses which of the available actions will be invoked afterwards and which data the action will get. In other words, when simulating users we have to model their navigational choices and the input they enter. Since we are aiming at real-time simulation, we also need to model the timing of user behavior. In the case of web applications, this can be reduced to a model of user response time or "think time", i.e. the time delay between reception of a screen and submission of a form. Since the fine-grained interaction involved in user input happens at the client side, transparent to the server, we do not model it.

We cannot predict user behavior as we can predict the behavior of a web application. Therefore, we use a stochastic model, which makes only assumptions about the probability of a particular user behavior and not about which behavior will actually occur. When estimating such probabilities, it can be important to take into account the session history of a user, which may influence the decision about the next step. For example, a user that has just logged into the system is unlikely to log out immediately afterwards, but much more likely to log out after he or she did other things. Consequently, we are dealing with conditional probabilities. The different parameters of stochastic formcharts were described in [7]. The following list gives a brief summary of them.

P_{form} is a discreet probability distribution for the forms that can be chosen on a page. Each page usually has a different P_{form}. In our stochastic load testing simulation this distribution is used to determine which choice a user makes on a page, and which action is invoked next.

p_{delay} is a probability density function for the think time of a user on a particular page. As a simple approximation, a normal distribution can be used, which expresses that extreme cases such as very long think times or very short think times are less frequent than think times that are average.

P_{input} is a probability distribution that is defined for each form in the system. It assigns a probability to each possible form parameter that a user can enter. Describing this distribution adequately is a challenge because the permitted values often depend on the application logic, and the application might not work properly if the values are nonsensical. One of the contributions of this paper is to shed light on a formal approach for specifying P_{input} as part of the form-oriented model.

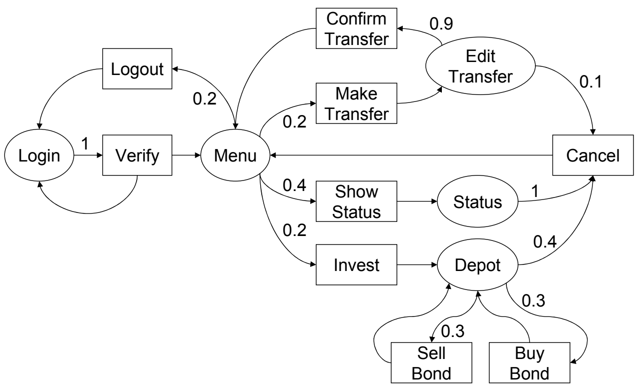

Fig. 4. Simple stochastic formchart for the home banking system.

During load testing, a random number generator is applied to generate artificial user behavior as specified in the above probability distributions. Such user behavior data can be represented efficiently with common random numbers [13]. However, as with deterministic load tests, replaying user behavior data may not always elicit the same server response. Even with the server state being the same, server actions may behave nondeterministically.

Figure 4 shows a stochastic formchart for the home banking system. In this formchart, the client-server transitions are annotated with probabilities, expressing P_{form}. Such stochastic formcharts are similar to Markov chains, but there is a subtle and important difference: while a Markov chain creates a state machine with probabilities at every transition, a stochastic formchart is a bipartite state machine with probabilities only at the transitions going from page to action. Which transition will be chosen from an action to the next page is determined by the logic of the system. However, if we assign probabilities to these transitions the same way we do with page-action transitions, e.g. by measuring relative frequencies, estimation or simply using uniform probabilities, then we can analyze the model statically and make statistical estimates about the system under test similar to those described in [30] for Markov chains. The difference to Markov chains is that our model also captures behavior over time, i.e. delays. We call a formchart with probabilities on all transitions a *doubly* stochastic formchart.

The formchart in Fig. 4 can be used as the basis for a simple stochastic simulation. However, the model is relatively simplistic because it does not take into account what the user has already done in the system. For example, logging out is equally probable no matter if the user has actually used one of the system functions or not. This can be remedied with *history-sensitive formcharts*, which are based on the notion of decision trees (e.g. see [16]). Pages and actions are cloned if they are used differently in different historical contexts, i.e. with different preceding session histories.

The history-sensitive formchart in Fig. 5 shows the effect of session history on the probability of the Logout action. Instead of having only one Menu page, the formchart contains

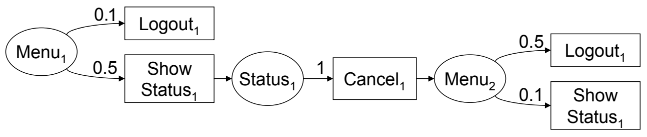

Fig. 5. History-sensitive feature of the home banking application.

two, each representing a different historical context: Menu$_1$, which represents the Menu page after a user has just logged in, and Menu$_2$, which represents the page after a user has chosen the Show Status action and gone back to the Menu. After having just logged in to the system, a user is very unlikely to immediately log out, therefore the probability that Logout is chosen is very low. However, after checking the status of an account, which is a very common function, a user is a lot more likely to log out. This case is represented in the transition from Menu$_2$ to Logout, which has a much higher probability. At the same time, if a user has just seen the status of an account, it is less likely that the user will choose the Show Status action again. More information about stochastic formcharts and how to use them for load testing can be found in [7].

A. *Advantages*

Stochastic load testing has several advantages over deterministic load testing, which consists mostly of replays of pre-recorded user sessions. First of all, it is a lot easier to specify one stochastic model instead of creating a multitude of pre-recorded user sessions. Even if we wanted to cover only the most important sequences of pages and actions, this would be a lot of work for a medium to large size real-world application. The number of possible sequences grows exponentially with the number of actions. With a stochastic model, the size of our specification is linear to the number of actions and pages: for each action and page we have to define parameters such as P_{input} and P_{form}. Stochastic load testing is a generalization of deterministic load testing: deterministic load testing means that transition probabilities are either 0 or 1.

Secondly, stochastic load testing can expose unexpected bottlenecks in a system because it naturally achieves a much higher coverage during testing. Due to the stochastic nature, every possible sequence of actions may occur, which may produce effects that remain hidden with a limited number of pre-recorded sessions. For example, some actions in a system may compete for the same resources, such as memory, processor time or database locks. The longer a stochastic load test is performed, the more likely it is that such dependencies in the system show themselves in a reduction of the measured system performance. After a stochastic load test, it is possible to analyze what actions actually caused a bottleneck by looking into the load testing logs. These logs can include an archive of the sent and received messages, as defined by the message model, which makes it easier to store and analyze this data in a typed, well-defined manner. Instead of having to come up with critical scenarios upfront, a stochastic load testing approach allows testers to find critical scenarios a-posteriori by analyzing collected data.

This is similar to the problem of developer-defined vs. automatically generated test cases for functional testing. To achieve an adequate test coverage, a very large number of test cases is necessary. It can save a lot of time to generate test cases automatically instead of defining them manually. However, in functional testing, automatic generation of useful test cases can be difficult due to the inherent complexity of program code. In the domain of web application load testing, the coverage does not so much relate to the program code but to the pages and actions, as expressed in a form-oriented model. It is much easier to automatically generate load test scenarios from such models, as they are a lot less complex.

B. *Realism*

It is important to note that the realism of load test models really matters. In the simulation of user behavior for the purpose of load testing it has been found to be crucial [3], [33]. "A load test is valid only if virtual users' behavior has characteristics similar to those of actual users" because "failure to mimic real user behavior can generate totally inconsistent results" [20]. Stochastic load test specifications are generally more realistic than specifications that consist of pre-recorded sessions because they cover a greater variety of the possible operational paths. However, when comparing different stochastic specifications, it becomes evident that a stochastic model alone is not a guarantee for realism. As discussed, stochastic models are easier to create than deterministic load models, but still, model parameters have to be chosen carefully.

For example, let us consider the simple model in Fig. 4 and the more sophisticated model in Fig. 5. In the simple model, the probability for Logout is generally quite high (0.2), whereas the probability for logging out in the more sophisticated model is only high after the user has used one of the system functions. Even if the probability for logging out from the Menu page is generally 0.2, it is important to recognize that users use the system first before logging out. Using the simple load model means that 20% of all users would log out immediately after logging in, putting hardly any stress on the system. As a consequence, the simple load test model would severely underestimate the effect of user sessions on system load, and produces misleading results.

IV. Specifying Load Testing Scenarios

The load testing process poses a number of challenges such as the generation of form parameters and the recognition of server responses by the load engine. In this section we describe how a form-oriented model can be annotated so that these issues can be specified formally. With the appropriate specifications, a form-oriented model can be used directly to generate load testing scenarios, without the need to develop load testing code manually.

A. *Parameter Specifications*

The task of the load engine is to simulate a user. It is relatively easy to automatically choose a form or link on a webpage, but it is not so easy to fill out a form with suitable parameters. Therefore, the form-oriented load testing model can be annotated with parameter specifications, which help the load testing engine in accomplishing this task. The point that makes parameter specifications formal and meaningful is that they refer directly to the message types of pages and actions as introduced in Section II. In the form-oriented terminology [10], parameter specifications are client output specifications. This methodology is flexible with respect to the chosen specification language, which can be for example the object constraint language (OCL) [24] of UML or another suitable specification language.

In the form-oriented load testing model, parameter specifications are given in a common mathematical, functional notation for case distinctions of the form:

$$\begin{array}{rcl} variable & = & condition_1 \rightarrow value_1, \\ & & condition_2 \rightarrow value_2,\ \ldots, \\ & & defaultValue \end{array}$$

We generalize this by allowing values between 0 and 1 for the condition specifying nondeterministic choice. The given condition specifies the probability of the condition being true, i.e. the probability of the associated value being chosen as the value for the variable.

Let us consider some examples. The first challenge for the generation of suitable form parameters in our system is the Verify action. Most users will enter a well-defined, correct username/password pair, so purely random generation of parameters would be completely unrealistic here. However, sometimes users do enter wrong credentials, so our load engine should do the same. This behavior can be specified in the following manner:

$$(user/pwd) = 0.9 \rightarrow \text{rnd}(Credentials),\ (void/void)$$

In other words, with a probability of 0.9 the username/password pair is taken randomly from a list $Credentials$ of valid credentials, and with the remaining probability of 0.1 invalid credentials are used.

The following parameter specification can be used for action Confirm Transfer:

$$\begin{array}{rcl} payee & = & 0.95 \rightarrow \text{rnd}(Payees),\ void \\ account & = & 0.95 \rightarrow \text{rnd}(Accounts),\ void \\ amount & = & 0.95 \rightarrow \text{rnd}(0, maxAmount), \\ & & maxAmount + 1 \\ when & = & 0.5 \rightarrow now,\ 0.25 \rightarrow weekly,\ monthly \end{array}$$

With a 0.95 probability the $payee$ is randomly chosen from a list $Payees$ of valid payees, otherwise an invalid value is produced. The same happens for the $account$. With a 0.95 probability, the $amount$ is chosen randomly between 0 and $maxAmount$, otherwise an invalid value greater than $maxAmount$ is produced. Note how the value $maxAmount$, which is defined on the page from which Confirm Transfer is invoked, can be used in the specification. $maxAmount$ is formally well-defined in the type of page Edit Transfer, and can be recovered by the load testing engine with simple screen scraping techniques, e.g. by using path expressions and regular expressions [22]. It is common that values represented on a page are important for the generation of suitable form parameters, hence the page types, combined with simple screen scraping, are very useful.

As a last example, we want to look at a parameter specification for action Sell Bond:

$$\begin{array}{rcl} a & = & \text{rnd}(assets) \\ bondId & = & a.bondId \\ qty & = & 0.95 \rightarrow \text{rnd}(0, a.qty),\ a.qty + 1 \end{array}$$

In this example, a is an auxiliary variable. It holds a randomly chosen asset, with $asset$ referring to the assets that are associated with page Depot from which action Sell Bond is invoked. $bondId$ is always the chosen asset's $bondId$. The quantity to sell is either a valid quantity, i.e. chosen randomly between 0 and the available quantity $a.qty$, or an invalid quantity greater than $a.qty$.

Parameter specifications can be used to infer probabilities for action-page transitions. In a stochastic formchart, only page-action transitions are annotated with probabilities. But a parameter specification usually describes choices that deterministically lead to a certain action-page transition. In the user credentials example above, correct credentials should always lead to successful login, and incorrect credentials should always lead to an error. Hence the probabilities of the parameter specification are probabilities for the action-page transitions chosen by the targeted server action, and we can use these probabilities as annotations in the formchart. Note that it is not always possible to infer action-page transition probabilities like this because the server behavior may not be easily predictable. In general, server behavior cannot be determined by the load testing engine; it is part of the observation, not part of the controlled parameters. An example illustrating this is an auction site where the virtual users influence each other through competing bids.

B. Page Recognition Specifications

Sometimes the load engine has to recognize which page was actually returned after invoking an action. In the home banking example, this is the case with action Verify: if the username/password combination was correct, then the Menu page is returned, otherwise we go back to page Login. Whenever we have several outgoing transitions at an action, we need to specify how the load engine can recognize which transition was chosen. This is done with page recognition specifications. Note that these specifications are not used to evaluate whether an action gave a "correct" response, as in functional testing. They are used to make sure that a load engine stays in sync with the load test model, i.e. that it can recognize the current state.

Page recognition specifications are predicates that are either true or false, expressing if a particular page was recognized. For example, the recognition specification for page Login

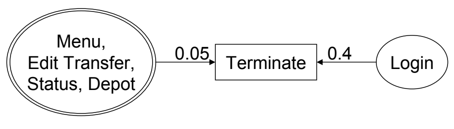

Fig. 6. A feature modeling session drop out.

could be:

$$title \text{ contains } "Login"$$

Similar to screen scraping, page recognition specifications operate on the document object model (DOM) [31] of a page. Consequently, they use path expressions and regular expressions for matching the values of a page against the values that are expected.

C. Session Drop Out and Server Unavailability

There are two types of possible events that are not considered in the simple stochastic model of Fig. 4: session drop out and server unavailability. Session drop out means that the user simply aborts the session, e.g. by browsing to a page outside the web application or closing the web browser. To model session drop out, which is a common occurrence in real user sessions, we need to reduce the probabilities in Fig. 4 slightly to accommodate additional transitions from every page to an artificial action Terminate, which symbolizes the end of a session. Because the arrows between all the pages and Terminate would render a formchart unreadable, we model session drop out as a feature, as introduced in Section II. As a simple approximation, we could assume that the session drop out probability is the same for all pages except the Login page (0.05). For the Login page the dropout probability is a lot higher (0.4). This is because most users use the Logout action before terminating a session, which leads to the Login page. Hence, sessions are usually terminated from that page.

Figure 6 shows the session drop out feature, which uses the state set notation, i.e. a double bubble with a list of page names. The double bubble with its transition to action Terminate signifies that all the pages listed have such a transition. It is much easier to read than drawing a separate transition for each of the listed pages.

Server unavailability covers the case that the web server responds with a “Service temporarily overloaded” code due to high load. It can be modeled with transitions from every action to a new page Unavailable. Similar to probabilities on action-page transitions, this is only relevant for static, model-based analysis, not for actual load testing. Analogously to session drop out, it is convenient to model server unavailability as a separate feature using the state set notation.

V. Workload Models

A workload for a given web application under test is completely described by a stochastic formchart and one of several possible usage intensity parameters; both elements together give a workload model. A usage intensity parameter can be a function over time, modeling a changing workload. For the introduction and comparison of the different parameters here we focus on scenarios where the parameter is constant over time after an initial warm up.

The combination of a system under test, a given constant usage intensity and a given stochastic formchart is *measurable* if the stochastic formchart has a defined *average number of requests* ($AVGR$) before the session terminates and an *average session duration* ($AVGD$). This excludes formcharts that do not terminate. It also excludes systems that age, i.e. systems where the same operation performs slower over time due to some internal state change. In many cases it is possible to statically check whether a stochastic formchart has such a finite $AVGR$; then we call it a finite user session.

In the load test, the stochastic formchart is executed several times in parallel. For one point in time, every running stochastic formchart is a *virtual user*. The *client request rate* (CRR) is the request rate of the individual virtual user averaged over its lifetime. Hence we have:

$$CRR = AVGR/AVGD$$

or

$$AVGR = AVGD \cdot CRR$$

AVGD is determined by the server response time and the user think time after receiving the response. The think time of every request should be greater than 0. If this think time is kept constant, then CRR decreases as soon as the server response degrades. For load testing tools, a load-independent CRR is often recommended, but this requires in general a non-trivial implementation [3].

A. Usage Intensity Parameters

There are two traditional usage intensity parameters: one is the *number of virtual users* (VU), i.e. the number of user processes active at a point in time; the other one is the *request rate* (RR), i.e. the number of requests generated per time unit. All client requests – in our terminology form submissions – are counted. Based on our workload models we can define a different usage intensity parameter, the *starting user session rate* (SUR). This is the average number of finite user sessions that is started per time unit. It is related to the arrival pattern in queuing theory [12]. All these three usage intensity parameters can be used to describe a constant load. In fact, for all three parameters the following holds on measurable systems under test: if the usage intensity parameter is constant over time, then the load is constant over time. A constant workload model can hence be described as a pair $(stochasticformchart, paramname = value)$ where the second element is a key-value pair naming a usage intensity parameter as a key and a dimensioned number as a parameter value.

Operationally, in a load test framework, the parameters are applied differently. If VU is the usage intensity parameter, a constant load is achieved by generating a certain number of virtual users, and then ceasing to generate new users. We can also control the load with SUR. A constant load is achieved by continuously generating new user sessions with a constant

SUR. After an initial start-up time in the order of $AVGD$ we have constant load. VU is in this case an observable parameter that is affected by SUR and other parameters.

B. Model-Realistic Workload Description

The two elements of the workload model, i.e. stochastic formchart and usage intensity parameter, should model truly different aspects of the model as a separation of concerns. Otherwise, the model will be badly maintainable as we will see in the following. We say a workload description is *model-realistic* if the following holds for a change in the stochastic formchart: if any value in the stochastic formchart is changed, for example if a think time is shortened to model an improved page readability, or if the number of requests per user session is changed because the user navigation is improved, then this change in the user model should have the same effect on the load test as it would have on the real system. We restrict our attention here to these two types of changes in the user model. We can show that the two major conventional usage intensity parameters, namely VU and RR, do not give model-realistic workload models, but our newly defined parameter SUR does. In order to obtain this result, we have to clarify the relationships between the different parameters defined so far.

C. Workload Laws

We capture the mutual relation between the usage intensity parameters in a set of laws. As said earlier, these laws are derived for the case of a constant workload. These laws describe the case of a negligible server response time, meaning that the server response time is much smaller than the user think time. They create the foundation on which the behavior for non-negligible server response time can be discussed.

First, we derive a law concerning the relationship of RR, VU and CRR. We want to show:

$$RR = VU \cdot CRR$$

For that we assume a constant VU. In Fig. 7, executions of the stochastic formchart – we call them sessions – are indicated by gray bars with teeth for the individual requests. A constant VU means that a new session starts immediately whenever another session has stopped, giving rise to VU horizontal lines of sessions in the diagram. The average request rate per unit of time for the whole experiment is now obviously VU times the request rate of each horizontal line in the diagram. The request rate in each such line is CRR by definition, hence proving the equation above.

We now consider a workload model with usage intensity parameter SUR. After an initial start-up we have:

$$RR = SUR \cdot AVGR$$

This can be seen by assuming a fixed SUR. Let case a be the case that there is only one request per session, $AVGR_a = 1$, then we have as many requests as we have starting user sessions. Hence we have $RR_a = SUR = SUR \cdot AVGR_a$ in this case. If in case b the session model is changed to any other

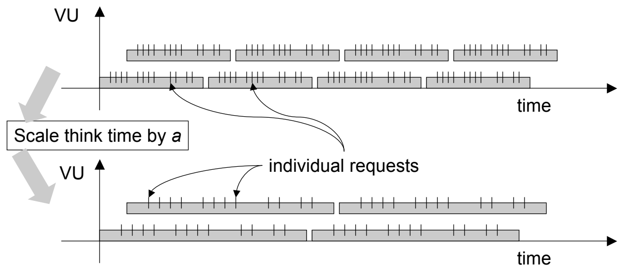

Fig. 7. The Request Rate (RR) for a fixed number of virtual users (VU) changes with the think time. As one sees, the request density is lower in the lower picture.

user model so that each session issues on average x request $AVGR_b = xAVGR_a$, then ceteris paribus the request rate changes by a factor of x as well $RR_b = xRR_a$ so we have:

$$\begin{aligned} RR_b &= xRR_a = xSUR \cdot AVGR_a \\ &= SUR \cdot xAVGR_a = SUR \cdot AVGR_b \end{aligned}$$

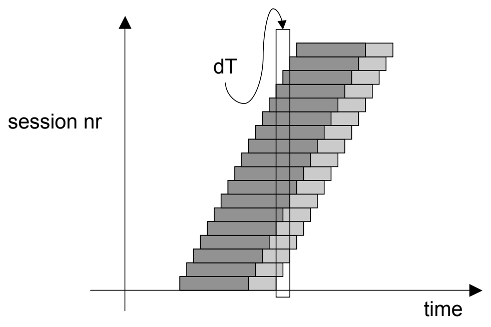

Fig. 8. RR for fixed SUR and varying $AVGR$.

Since b is the general case, we have proven the equation. This argument is visualized in Fig. 8. The horizontal bars are sessions, and we assume that they have a constant CRR. Hence the longer sessions have a proportionally higher $AVGR$. The law can also be illustrated by assuming a fixed $AVGR$, $AVGD$, CRR, and varying SUR. This is shown in Fig. 9. The starting user rate is the slope, i.e. the first derivative of the most recent session number n. Fixed CRR implies $RR \sim VU$. In the case of a larger SUR, i.e. a steeper slope, each small time interval dT intersects more sessions and therefore contains more requests. Hence RR is proportional to SUR.

Now we see that globally scaling all think times ceteris paribus does not change RR, since the duration of the single user session is irrelevant after start-up. The equation contains $AVGR$ and not $AVGD$. This behavior of the load test is exactly the behavior of the real system; for the real system, the same equation holds. We now consider the second condition of realism. If we change the user model by, say, halving $AVGR$, and if we assume for simplicity that all requests create the same load, then in both, the load test and the real system, the

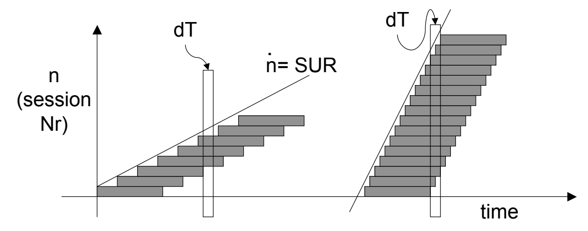

Fig. 9. RR for fixed $AVGR$, $AVGD$ and varying SUR.

load will be halved. According to our definition, this indicates that workload models with usage intensity parameter SUR are model-realistic. In fact, workload models with usage intensity parameter SUR have other advantages. RR is not sensitive to server load, even if the think time of the session clients would be sensitive to the server load. Even if the actual user agents are programmed in a way that they have $AVGD$ that are dependent on the server response, if SUR is kept constant, then the load is constant. This is because VU is changed appropriately if we scale the think times. Taking all three previous laws together we have:

$$VU = SUR \cdot AVGD$$

We sum up the laws in the following table. This set of laws is redundant, in that exactly one of the equations can be deleted, since every equation can be derived from the other three. It is deliberately presented in redundant form because for that same symmetry reason it would be to some extent arbitrary to choose one the laws and delete it.

traditional parameters	$RR = VU \cdot CRR$
session metrics	$AVGR = AVGD \cdot CRR$
request rate	$RR = SUR \cdot AVGR$
virtual users	$VU = SUR \cdot AVGD$

We can now discuss whether usage intensity parameters RR and VU are model realistic. First, we assume constant VU. Figure 7 shows what happens after globally scaling all think times in the stochastic formchart by factor a: this changes CRR by factor a, and also RR. The scaling of think times hence changes the load on the system under test if VU is constant. On the real system however the load is not expected to change in a typical scenario where the think time changes, for example if a think time is shortened by improved page readability. If all users simply take less time to think, but still do the same number of requests, RR does not change! Instead, on the real system VU would change because of the law $VU = SUR \cdot AVGD$ demonstrated in Fig. 8. Hence realism is violated for a workload model with usage intensity parameter VU. This means that VU is not model-realistic.

We now discuss whether a workload model with usage intensity parameter RR is model-realistic. We note that for such models the request rate is trivially kept constant if think times are scaled. We now consider the second condition of realism: if we change the user model by reducing $AVGR$ and we assume for simplicity that all requests create the same load, then the load during the load test remains constant if we have fixed RR. But on the real system, the load would decrease. Hence workload models with usage intensity parameter RR are not model-realistic.

D. Example of Non-Model-Realistic Load Test Results

We discuss now an example showing that an unrealistic workload model, if applied naively, can create misleading load test results. We take as example an enrollment system for university students. Using the system is mandatory for students. We compare the behavior of the different load test approaches during maintenance of the application. We assume that a workload model with usage intensity parameter RR is used, that the system performance is sufficient, and that every student performs 10 requests during enrollment with only the last one creating a heavy load on the system. Now imagine the user interface was improved, and only two requests per user are necessary: the first one being lightweight, and the last one causing the same heavy load as before. If we naively change only the user model in the workload model with RR, and not the usage intensity parameter RR itself, then the load on the system under test would increase roughly by factor 5 and could bring down the system. In the real application the system load would not increase, but rather decrease. The problem remains if the workload model is set up with usage intensity parameter VU. In contrast, if the workload model is set up with the realistic usage intensity parameter SUR, the system load correctly decreases in the load test, hence resembling the behavior of the running application.

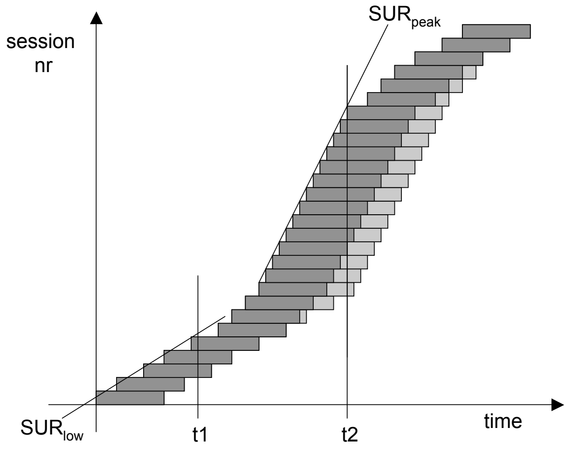

Fig. 10. RR for fixed $AVGR$ and varying SUR.

We said that we assume negligible server response times. The question arises whether non-negligible server response times, as they arise during heavy load, make this argument stronger or weaker. Figure 10 shows a scenario with non-constant load. The light gray extensions of the sessions around $t2$ indicate session delays by non-negligible server response times. They are an observed variable. The intersections of the vertical lines at $t1$ and $t2$ with the sessions indicate the values of VU at these times. VU is modified by the observed session delay – in simple terms this is visible because $t2$ cuts the light gray area of sessions. It is therefore a bad control variable. SUR is the slope of the starting edge of the user sessions. SUR is purely a controlled variable. If the longer session duration would not only be caused by longer server response time but also by more requests per session, perhaps due to more error cases in each session, then the usage intensity parameter RR would be similarly affected as VU. However, the details of this argument are beyond the scope of this discussion.

As we see, workload models with usage intensity parameters RR or VU can deliver spurious results if the parameters are not changed with every change to the user model. These changes can theoretically be done, but they would require exceptional insight into the effects of the above equations or lucky intuition to the same effect. In contrast, the correct load test behavior comes for free in the workload model with the realistic usage intensity parameter SUR.

VI. Related Work

One distinct difference between the form-oriented load testing model and other load testing approaches is their purpose. Other load testing approaches are concerned with load testing only, but the form-oriented model is in fact an analysis model that is used for the development of enterprise applications. The same form-oriented model can be used as a basis for system specification, forward and reverse engineering, and load testing. This means that changes of the system specification result directly in changes of load testing specifications, i.e. the correlation between the two makes maintenance of the load test models a lot easier.

In many cases, load-testing is still done by hand-written scripts that describe the user model as a subprogram [26], [28]. For each virtual user, the subprogram is called, possibly with a set of parameters that describe certain aspects of the virtual user's behavior. Often the users are also modeled by a multimodel, i.e. a model consisting of several submodels for different categories of users, which defines a subprogram for each user category. Data is either taken from a set of predefined values or generated randomly. With regard to input data this approach has a certain degree of randomization, however, user behavior itself mainly remains a matter of repetition. This approach does not support the degree of abstraction and formal specification of the form-oriented model. It is purely script driven and suffers, like any hand-written program, from being prone to programming errors. The load engine itself has to be developed and brought to a mature state, which is usually a very time consuming task.

Products for industry-strength load testing [23] such as Mercury Interactive's LoadRunner [21] take a similar script-driven approach. LoadRunner offers a visual editor for end-user scripts. No conventional programming is needed, and the scripts describe the load tests in a more domain-specific manner. End-user scripts are run on a load engine that takes care of load balancing and monitoring. Most current load testing tools operate in a similar manner. A detailed discussion of bottleneck problems is given in [3], where the authors also present a nontrivial implementation for load test clients. More information about load test practice can be found in [14], [15], [20], [27].

There already exist model-based approaches for testing of web applications, e.g. in [2], [19], [29], but they usually focus on the generation of test cases for functionality testing. For example, finite-state models with constraint annotations for form parameters have been used for generating test cases [2]. It has also been shown that functional test cases can be efficiently generated based on real user session data [18].

Different studies have shown that stochastic models, in particular Markov chains, provide benefits for functional testing [17], [29], [30]. They can be used for the automatic generation of large randomized test suites with a high coverage of operational paths. For example, in [25], [29] analysis models are used for regression testing in web site evolution scenarios. The model for user navigation is a stochastic one similar to Markov chains, but all user input data has to be given in advance for the system to work. While this may be appropriate for regression testing, it is not flexible enough for performing load tests. A Markov chain model like that in [29], [30] can only be used for a system where identical inputs cause identical state transitions, which is not the case in most web applications that rely on session data or a modifiable database. Consider, for example, an online ticket reservation system: after a specific place has been booked, it is not available any more; thus, repeating the same inputs will cause different results.

The motivation of using a statistical model based on data about real user behavior for realistic load testing of web sites was already anticipated in [17], but their model fails to distinguish the user behavior, which can only be adequately modeled as a stochastic process, from the system behavior, which is given by the implementation. Transforming a state model of a web site directly into a Markov chain is not sufficient and does not account for the system's behavior, which is usually not stochastic. The form-oriented model offers page recognition specifications to deal with this problem. In [32] it was shown that creation of a simple stochastic user model with real user data represents a valid approach for load testing. However, most approaches offer no model for specifying user behavior over time, and it is usually neglected that form choice probabilities may change during a session.

From the perspective of queuing theory [12], a web application can be regarded as a queuing system, with requests being queued and processed by sending back responses. The workload models discussed in Section V influence the arrival patterns in the system. While workload models such as starting user session rate are deterministic, arrivals at the server actions are stochastic due to random variables such as the user think time. In order to model a web application as a queuing system, a service pattern for the web application would have to be specified. This would be very hard, as the service time probabilities of the server actions may influence each other due to contention for resources, e.g. lock contention in a database. The service pattern of an action, in turn, influences the arrival patterns at subsequent actions. Load testing allows us to explore the performance of a web application empirically, rather than doing so analytically using a complex queuing theoretical approach. The stochastic formchart model is not a queuing system at all because it models the actions of a single user, and thus does not involve any queues. If we were to model web application as a queuing system, the stochastic formchart model and the workload model would be among the factors influencing the interarrival-time distributions of the server actions.

VII. Conclusion

We discussed the form-oriented stochastic load testing approach, and showed that with appropriate annotations all the information necessary for realistic load testing can be specified in the context of a form-oriented analysis model:

- Stochastic user navigation, with the possibility to make stochastic behavior dependent on the session history
- Stochastic form parameter generation and recognition of returned pages
- Appropriate workload parameters

Stochastic load testing has advantages over load testing with pre-recorded session data. It is a lot less work to create load tests with a high coverage of the operational paths of the application, and helps to reveal unanticipated bottlenecks. Some popular workload models such as number of virtual users are not suitable for load testing, and our work shows that the user session starting rate is a more realistic workload parameter. This paper focused on submit-response style systems, with web applications as the best known members of this class. The advantage of our methodology is that it is a technology-independent model that can be applied to submit-response style systems built on web services as well. The basic stochastic model and the findings on usage intensity parameters are applicable to the whole range of service oriented systems.

References

[1] Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[2] A.A. Andrews, J. Offutt, and R.T. Alexander. Testing web applications by modeling with FSMs. *Software and Systems Modeling*, 4(3):326–345, 2005.

[3] Gaurav Banga and Peter Druschel. Measuring the Capacity of a Web Server under Realistic Loads. *World Wide Web*, 2(1-2):69–83, 1999.

[4] Paul Barford and Mark Crovella. Measuring Web Performance in the Wide Area. *SIGMETRICS Perform. Eval. Rev.*, 27(2):37–48, 1.99.

[5] Y. Cai, J. Grundy, and J. Hosking. Synthesizing client load models for performance engineering via web crawling. In *ASE 2007 – Proceedings of the 22th International Conference on Automated Software Engineering*, pages 353–362, 2007.

[6] Kevin Curran and Connor Duffy. Understanding and Reducing Web Delays. *Int. J. Netw. Manag.*, 15(2):89–102, 2005.

[7] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, and G. Weber. Realistic load testing of web applications. In *CSMR'06: Proceedings of the 10th European Conference on Software Maintenance and Reengineering*. IEEE Press, 2006.

[8] Dirk Draheim, Elfriede Fehr, and Gerald Weber. JSPick - a server pages design recovery. In *7th European Conference on Software Maintenance and Reengineering*, LNCS. IEEE Press, March 2003.

[9] Dirk Draheim, Christof Lutteroth, and Gerald Weber. A Source Code Independent Reverse Engineering Tool for Dynamic Web Sites. In *9th European Conference on Software Maintenance and Reengineering*. IEEE Press, 2005.

[10] Dirk Draheim and Gerald Weber. *Form-Oriented Analysis - A New Methodology to Model Form-Based Applications*. Springer, October 2004.

[11] Dirk Draheim and Gerald Weber. Specification and Generation of Model 2 Web Interfaces. In *APCHI 2004 - 6th Asia-Pacific Conference on Computer-Human Interaction*, LNCS 3101. Springer, June 2004.

[12] D. Gross and C.M. Harris. *Fundamentals of queueing theory*. Wiley, 1998.

[13] R.G. Heikes, D.C. Montgomery, and R.L. Rardin. Using common random numbers in simulation experiments–an approach to statistical analysis. *SIMULATION*, 27(3):81, 1976.

[14] Arun K. Iyengar, Mark S. Squillante, and Li Zhang. Analysis and Characterization of Large-scale Web Server Access Patterns and Performance. *World Wide Web*, 2(1-2):85–100, 1999.

[15] Raj Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[16] K. Jajuga, Andrzej Sokoowski, and Hans Hermann Bock. *Classification, Clustering and Data Analysis*. Springer, August 2002.

[17] Chaitanya Kallepalli and Jeff Tian. Measuring and Modeling Usage and Reliability for Statistical Web Testing. *IEEE Trans. Softw. Eng.*, 27(11):1023–1036, 2001.

[18] Srikanth Karre. Leveraging user-session data to support web application testing. *IEEE Trans. Softw. Eng.*, 31(3):187–202, 2005. Member-Sebastian Elbaum and Member-Gregg Rothermel and Member-Marc Fisher II.

[19] David Chenho Kung, Chien-Hung Liu, and Pei Hsia. An Object-Oriented Web Test Model for Testing Web Applications. In *COMPSAC '00: 24th International Computer Software and Applications Conference*, pages 537–542, Washington, DC, USA, 2000. IEEE Computer Society.

[20] Daniel A. Menascé. Load Testing of Web Sites. *IEEE Internet Computing*, 6(4):70–74, July 2002.

[21] Mercury Interactive Corporation. Load Testing to Predict Web Performance. Technical Report WP-1079-0604, Mercury Interactive Corporation, 2004.

[22] J. Myllymaki. Effective Web data extraction with standard XML technologies. *Computer Networks*, 39(5):635–644, 2002.

[23] Newport Group Inc. Annual Load Test Market Summary and Analysis, 2001.

[24] Object Management Group. OCL 2.0 Specification, June 2005.

[25] Filippo Ricca and Paolo Tonella. Analysis and Testing of Web Applications. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 25–34, Washington, DC, USA, 2001. IEEE Computer Society.

[26] Andreas Rudolf and Raniner Pirker. E-Business Testing: User Perceptions and Performance Issues. In *Proceedings of the First Asia-Pacific Conference on Quality Software*. IEEE Press, 2000.

[27] Jonathan C. C. Shaw, Colin G. Baisden, and Will M. Pryke. Performance Testing – A Case Study of a Combined Web/Telephony System. *BT Technology Journal*, 20(3):76–86, 2002.

[28] B.M. Subraya and S.V. Subrahmanya. Object Driven Performance Testing of Web Applications. In *Proceedings of the First Asia-Pacific Conference on Quality Software*. IEEE Press, 2000.

[29] Paolo Tonella and Filippo Ricca. Statistical Testing of Web Applications. *Software Maintenance and Evolution*, 16(1-2):103–127, April 2004.

[30] James A. Whittaker and Michael G. Thomason. A Markov Chain Model for Statistical Software Testing. *IEEE Trans. Softw. Eng.*, 20(10):812–824, 1994.

[31] World Wide Web Consortium. Document object model (DOM) level 1 specification, 1998.

[32] Lei Xu and Baowen Xu. Applying Users' Actions Obtaining Methods into Web Performance Testing. *Journal of Software (in Chinese)*, (14):115–120, 2003.

[33] Lei Xu, Baowen Xu, and Jixiang Jiang. Testing Web Applications Focusing on their Specialties. *SIGSOFT Softw. Eng. Notes*, 30(1):10, 2005.