An Application of Stochastic Models To Monitoring of Dynamic Web Services

Marcelo De Barros, Manish Mittal
Bing Customer Experiences Engineering
Microsoft Corporation
Redmond, USA
marcelod@microsoft.com, manishm@microsoft.com

Abstract - Web search engines are very dynamic in nature; not only are the backend and data powering the site evolving, but the frontend is always adapting to different browsers, devices and form-factors, and experiments are often running in production. In fact, when it comes to User Experience (UX), it is likely that users are always falling into some live experiment in production: variation of colors, fonts, typography, different Java Scripts and so on. Issues (software bugs) can occur on the live site for very particular contexts, where a context is defined as a particular configuration of browser, market and experiment. As an example, a JavaScript error can occur on a certain page, for certain types of queries, against a certain market on a particular browser. The problem that we're trying to solve is to devise a probabilistic methodology to monitor and detect these particular software bugs in production environments by maximizing the chances of detecting the most relevant issues from the application users' standpoint. For this purpose, we at the Microsoft Bing Experiences Team developed a concept of synthetic exploratory monitoring that can focus on the important features on the sites and pages, and use invariants (conditions that should always hold true, or always hold false, for specific contexts), such as security-related invariants, to detect potential anomalies in the current context. We make use of stochastic models to ensure maximum relevant coverage of contexts and devices. We use the power of the Selenium testing framework to drive end-to-end automation on browsers and devices, the notion of exploratory tests, and a set of heuristics and invariants (text-based and image-based) that can auto-detect problems on the live site in very particular contexts. We compare and contrast two machine–learning models: Markov Chains and Time-Based Artificial Neural Networks (ANNs). We implemented the idea explained in this paper to monitor large-scale web sites such as Bing Search Engine where alerts are generated automatically whenever the anomaly conditions are detected. The solution is easily expandable to other sites. We envision, as future work, moving this technology to the cloud that would allow easy customization of all parameters (browsers used, definition of the finite-state machine, heuristics and invariants). This paper explains the fundamental principles to create a stochastic monitoring model and demonstrates how to apply the principles to large-scale web sites and services. We will utilize Bing Search Engine to illustrate the techniques explained here.

Keywords-software testing; large-scale services; quality of services; markov chains; artificial neural networks; selenium; testing in production; monitoring; stochastic models

I. SCALING SYSTEMS TO DEVICES, BROWSERS AND MARKETS

In today's world, whenever a new online system is launched, it is usually available across several devices (devices that display web contents), browsers and markets instantaneously and simultaneously. This poses a significant development challenge since:

a) Different browsers, devices and markets have specific requirements and resources that may differ from each other, and there is no enforced global standard across them.
b) Support for Cascading Style Sheet (CSS) and HTML5 compatibility and support vary significantly from browser to browser.
c) The form-factor for the different devices varies significantly. Because of smaller screens, code might need to be optimized to show the user different data or presentation of the information. Despite the development and adoption of responsive web design techniques [10], very often there is still a need for small code customizations. For instance, some devices are large enough to display data into two vertical panes (columns), where others require the use of a single pane.
d) Markets are also another important dimension given the differences in language grammars as well as geo-cultural differences. Large-scale systems such as Google, Bing and Facebook are always dealing with such challenges.

Many large-scale web sites are now making use of "flights" or "experiments". An experiment is a way to expose a percentage of the site's users to a different treatment of the site (which can be differences in the User Interface, middle-tier, backend or even data differences) in order to collect early feedback and then make an informed decision about the upcoming features for the system. For example, a search engine might want to expose 2% of its users to a SEarch Results Page (or SERP) that shows only eight "blue links" by default instead of ten blue links. The telemetry for that experiment is then collected and analyzed against the "control" (the ten blue links) and data analysts work on distilling the positive and negative aspects of the experiment, where positive aspects correspond to user metrics moving towards the expected direction (such as page load time being reduced, user abandonment reduced,

 ISBN: 978-1-61208-387-2

increased dwell time [16], amongst others) and negative correspond to the converse. Experiments can overlap with each other. At any point in time, there might be tens or even hundreds of experiments running in production environments [15].

The paper is organized as follows: in Section I, we describe the complexities involved with monitoring large-scale services. In Section II, we describe the current state of the art. In Sections III and IV, we introduce the ideas of Markov Chains and Selenium, respectively. In Sections V, we define the concept of exploratory runs. In Section VI, we define the concept of subscription-based validation methods. In Section VII, we devise the strategy for exploratory runs utilizing a stochastic model (such as Markov Chains), Selenium and the pre-defined concept of subscription-based validation methods to solve the monitoring problem. In Section VIII, we explore other stochastic models that can be used to solve the monitoring problem, such as Artificial Neural Networks. In Section IX, we provide a summary of the work as well as the direction for future research.

II. MONITORING COMPLEXITIES AND STATE OF THE ART

Since the code is somewhat customized to different user experiences (experiments, browsers, devices and markets), there is a possibility of encountering specific issues on any of these and worst, on combination of these dimensions: a specific problem may only happen on an experiment, on a given browser, on a given device and for a particular market. Some simple lower-bound calculations show the complexity and the scale of this problem. If we have around 30 experiments, 30 browsers, 30 devices and 200 markets, the number of possible combinations (assuming no overlaps on the experiments) becomes 30*30*30*200 = 5,400,000 different permutations. Even using well known monitoring techniques, such as Gomez [2] or Keynote [3], it becomes impossible to monitor all these variations. In reality, though, most of these contexts are either not significantly crucial to the business or are not valid at all (for example, most of the time experiments are limited to either a group of markets or a group of browsers), hence understanding the valid and important permutations can prune the combinatorial space considerably. Notice the usage of the terms “testing” and “monitoring” are interchangeable in this paper, both indicating the ability to proactively detect anomalies in real production environments.

The current state of the art for monitoring strategies consists basically of three approaches:

a) Synthetic Transactions [2]: market tools such as Keynote and Gomez give the capability of building custom, synthetic transactions to monitor particular features of web services and sites. However, synthetic transactions only target very limited set of features that need to be known a-priori which limits its effectviness when monitoring complex and dynamic systems. They are very inefective for highly dynamic services.

b) Performance Counters [14]*:* web services developers have the ability to implement performance counters on the server side which can give indications of potential system malfunctions. For example, a performance counter that tracks “75%tile server side latency” can be the initial lead to investigate real user issues with the service (in case of spikes or drops, for example). However, performance counters have the disadvantages that they usually fail to track client-only problems (such as javascript errors) and since they aggregate data across all the users, it only detects problems when a signficant number of users are affected by the problem – issues that affect only a very small percentage of users usually go undetected by performance counters.

c) Telemetry: telemetry consists of analysis of time series of logs from user activity as well as system probes in order to detect anomalies in production [12]. Although telemetry analysis has the capability of detecting widespread issues with one’s service, it is not a real-time monitoring system since the collection, aggregation and availability of the data are tasks that usually take signficant time to be performed, limiting its ability in detecting anomalies in a timely manner.

None of the current state of the art methoodologies hence is comprehensive enough to actually model the user’s behavior and detect in real-time relevant anomalies based on the true users’ patterns observed in production environments. The work described in this paper is an attempt to address this gap.

III. MARKOV CHAINS

We use Markov Chains [4] to model the behavior of the system, limiting the monitoring space to the most probable paths. A Markov Chain is a type of stochastic model based on the concept of Finite-State Machines [17]) that undergoes transitions from one state to another on a state space. It is a random process usually characterized as memory-less: the next state depends only on the current state and not on the sequence of events that preceded it. For search engines, the states in a Markov Chain are web site landing pages, such as: the search engine Home Page, the search engine Web Results Page, Videos Results Page, Images Results Page, Settings Page, and any the other page type included in the search engine substrate.

The actions that lead to a state transition are the different actions that can be performed by the end user, mainly Searches, Clicks, Tabs, Hovers, and so on. With enough anonymous log-based information about the different states and actions, one can build a comprehensive Markov Chain diagram modelling the proper behavior of the average user of the web system in questions. The assumption is that most web sites nowadays log information about their users’ iterations with the page (in an anonymized manner). The picture below (Figure 1) gives an example of a state transition, and the table below (Table I) gives an example of a simple Markov Chain. Notice that the key aspect here is that each action is associated with a certain probability (the "Probability Weightings" column in Table I), calculated based on the number (percentage) of users who triggered that respective action based on captured data. For example, the second row in Table I shows the state as being the "home page" and the probability weighting as being "20%". The semantics of such information is that when a user lands (or is) in the "Home Page" state, there is a probability of 20% that the user will perform the action of "typing a query" and hitting enter. The third row tells us that if the user is in the same state ("Home Page") there is also a probability of 15% that the user will perform the action of clicking on the "Top News" link. The table only shows a partial view of the probability weighting distribution.

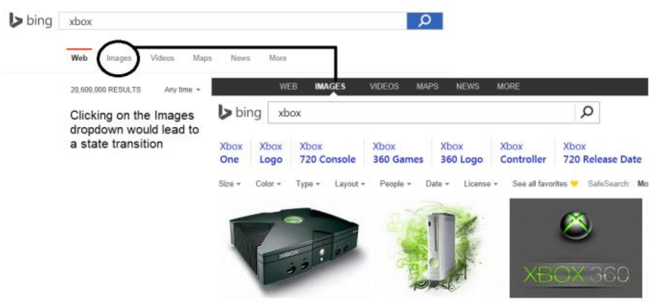

Figure 1. Example of state transitions

TABLE I. EXAMPLE OF MARKOV CHAINS STATES AND WEIGHTED TRANSITIONS

Initial State	Final State	Action	Probability Weightings
Home Page	Search Results Page	Typed Query	20%
Home Page	Search Results Page	Clicked on "Top News"	15%
Home Page	Home Page	Refresh the Page	5%
Search Results Page	Search Results Page	Typed Query	60%
Search Results Page	Non-Search Page	Clicked on Ads	15%
Search Results Page	Non-Search Page	Clicked on Algo Result	33%
Images Results Page	Videos Results Page	Click on the Videos link	10%
Images Results Page	Images Results Page	Click on Related Images	25%
Images Results Page	Images Results Page	Click on Related Entities	7%
Images Results Page	Images Results Page	Refresh the Page	13%

The granularity of the states and the actions is something that varies depending on the applications. In the example above, the Typed Query action could certainly be further refined by specifying the category/class of query being typed, such as "Local Query", "Adult Query" or "Electronics". Likewise the "clicked on" event can be grouped into categories (such as "clicked on Algo Results") or further refined down to the domain of the link being clicked (such as "clicked on an amazon.com link"). The important aspect is to create the chain in such a way that it truly encompasses the users' behavior but keeping it concise enough to prune the overall search space. For our project, we also added some random aspects to our testing in order to provide extra coverage. For example, when the action is "Send a new query" we take a random query from a pool of pre-defined queries, usually a combination of head and tail queries (See "Web Search Queries" [9]).

Some states are outside the scope of the pages being tested. For example, if the scope being tested is all the pages under the bing.com domain, any site outside that domain would be considered an out-of-scope state. It is important to model the chain in such a way that once out of the scope, actions will lead to in-scope states (such as clicking the back button, or navigating back to the initial state).

With the Markov Chain created, the monitoring approach can be tweaked to randomly follow the paths and probabilities specified by the chain. Notice that the approach will necessarily focus on the most probable paths (assuming a random distribution), which is the desired approach.

In addition to using a Markov Chain for transitions, another important aspect that needs to be taken into consideration is the overall distribution of browsers, devices, markets and flights (experiments).

There are two different approaches to integrating Markov Chains for these additional dimensions into the monitoring system:

1) Create the Markov Chain to take into account browsers, devices, markets and flights (experiments). In such cases, there can be multiple Markov Chains for each dimension, or combination of dimensions, or one Chain where states and transitions take into account these dimensions; or, alternatively
2) Create the Markov Chain without the particular data about browsers, devices, markets and flights, and use an orthogonal table with the distribution of the population across these dimensions, and randomly switch to a certain dimension as you navigate the chain.

The approach we have taken is the second one. The Markov Chain is created with the overall usage pattern across all the users in the system. At the same time we get the distribution of users across all browsers, devices, markets and experiments. In the following hypothetical example (Table II), we see several user context distributions across browsers, devices, markets and experiments. We then combine these two sources of data (the Markov Chain and the User Context Distributions) in order to come up with the proper stochastic model for the exploratory tests. Section VII explains the details of how these two data sources come together. Section VIII explains how the User Context

 ISBN: 978-1-61208-387-2

Distribution can be used as input into an Artificial Neural Network.

TABLE II. EXAMPLE OF USER CONTEXT DISTRIBUTIONS

Browser	**Percentage of users**
Internet Explorer 7	6%
Internet Explorer 8	8%
Internet Explorer 11	15%
Firefox	9%
Others	*62%*

Device	**Percentage of users**
Windows Phone	34%
iPhone	17%
Kindle Fire	17%
Android	9%
Others	*23%*

Market	**Percentage of users**
United States	52%
China	17%
Brazil	4.5%
Canada	7%
Others	*19.5%*

Experiment	**Percentage of users**
Experiment #1: light-blue background color	2%
Experiment #2: larger font size for titles	3%
Experiment #3: larger images	20%
Experiment #4: new relevance ranker	1%

A potential limitation of the Markov Chains is the fact that transitions from one state to the other do not depend on the path taken to get to the current state. This might be seem as a limitation of the model if there is a need to build more complex, "state-full" scenarios. That can easily be overcome by developing more detailed states inside the Markov Chain (adding complexity to it). For example, if there is a need to model a scenario where users come from page A through page B, we can build a state named "AB" that reflects that path.

IV. SELENIUM

Selenium [6] is a portable software testing framework for web applications that provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including Java, C#, Groovy, Perl, PHP, Python and Ruby. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and Macintosh platforms. The way we use Selenium for exploratory tests and monitoring is through Selenium WebDrivers. Selenium WebDriver accepts commands and sends them to a browser. This is implemented through a browser-specific browser driver, which sends commands to a browser, and retrieves results. Most browser drivers actually launch and access a browser application (such as Firefox or Internet Explorer). Selenium WebDriver does not need a special server to execute tests. Instead, the WebDriver directly starts a browser instance and controls it. There is an ongoing effort by the inventors of Selenium to make it an internet standard.

Selenium provides an easy interface to interact with the browser, and the same test scripts can be used against many supported browsers. The ability to perform clicks, hovers, navigation manipulation, simulate different keyboard commands to the browser, scroll, change the browser settings and even detect and manipulate pop-up windows make it ideal for web automation.

In order to provide extra reliability, one can make use of a Selenium Grid. Selenium Grid is a server that allows tests to use web browser instances running on remote machines. With Selenium Grid, one server acts as the hub. Tests contact the hub to obtain access to browser instances. The hub has a list of servers that provides access to browser instances (WebDriver nodes), and lets tests use these instances. Selenium Grid allows running tests in parallel on multiple machines, and to manage different browser versions and browser configurations centrally (instead of in each individual test).

V. EXPLORATORY RUNS

The term Exploratory Runs here is loosely used to define the process of semi-randomly exploring different parts of a system while performing different verifications and validations that are pertinent to the current part of the system in question. The semi-random nature is accomplished via two methods: walking the generated Markov Chain, and modifying the context based on the users' distribution of markets, browsers, devices and experiments. The process usually starts at the initial page of the system, such as the user's home page. At that point a frequency-weighted random set of actions gets triggered based on the weight (probability) of the actions in the Markov Chain. It continues from that point on following the same approach indefinitely or until a certain time amount elapses. The transition of the states is implemented via commands in Selenium. Figure 2 below illustrates a simple Markov Chain being walked probabilistically:

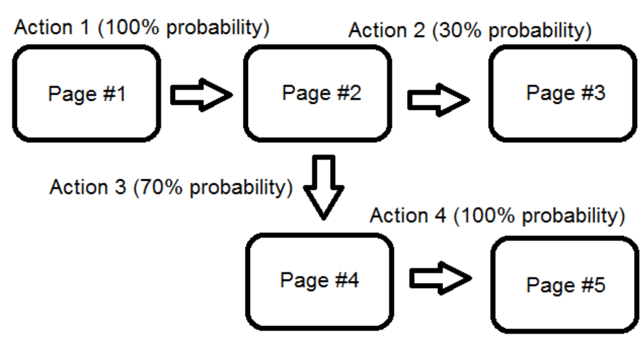

Figure 2. Schema depicting a simple Markov Chain

Orthogonally to the walk of the Markov Chain, we make use of the contexts distribution in the following manner:

a) Markov Chain traversal keeps happening randomly for a period of time (say N minutes)
b) After that period of time elapses, a change of context happens based on the distribution table

We use N = 30 minutes, which is based on our observations with real Bing user data, 30 minutes is the average time for a user web session. After 30 minutes, the contexts in which the tool is running may change: browser, device, market or experiment. The change is random but weighted based on the distribution tables. We utilize a number of Selenium Grids, one for each type of Internet Explorer (IE) browsers (from version 7 to the latest version), and all the grids also contain other browsers, such as Chrome and Firefox. Markets also change based on a set of pre-defined markets (around 200 in our case). The device is simulated on the desktop browsers by manipulating the user-agent. This simulation isn't ideal as some issues only appear or repro on the actual devices, but it is a good stopgap solution to catch some types of issues (like features being under/over triggered). We also force the exploratory run to fall into one (or combination of) experiments by using test hooks (in our case query-string parameters that are only enabled/visible inside the Microsoft corporate network). The automation keeps running indefinitely as a monitoring mechanism against production.

VI. Subscription-Based Validation Modules

It is common to see the schema of a validation module (or test case) as a self-contained unit that performs all the steps necessary to set up the proper pre-validation before the validation takes place, followed by the validation itself, culminating with the post-validation (or teardown). Schematically we have:

```
SampleTestCase()
{
  Pre-ValidationSetup();
  Validation();
  Post-ValidationSetup(); //Teardown
}
```

There are many advantages of such scheme: simplicity, standard pattern, readability, reproducibility, determinism, to name a few. However, such a model does not fit well into the exploratory runs mentioned previously. Instead, what we want is a subscription-based model where the test case subscribes to the current state (or action) if the current state (or action) meets certain criteria pertinent to that test. Schematically, subscription-based test cases have the following format:

```
SubscriptionBasedSampleTestCase()
{
  If(IsRelevantState(this.CurrentState))
    Validation();
}
```

In essence, we are proposing a separation of the validation method from the configuration. The test becomes opportunistic rather than deterministic: if we reach a situation during the traversal of the Markov Chain where the test is applicable, then it runs; otherwise it ignores the current state.

An example of a subscription-based test case would be the following: suppose that we want to write a test case to validate behaviors for a certain segment of queries called navigational queries, which are queries that seek a single website or web page of a single entity. A query such as "sales force" is a navigational query. There are several types of validation that can be performed for navigational queries. As per the example in Figure 3, when searching on "sales force", we can base validation on:

a) Correctness of the algorithmic first result returned
b) Proper attribution for "Official Site"
c) Proper number, format, truncation for deep-links
d) Proper placement and usage of inner-search boxes

The picture below (Figure 3) depicts the items that can be subjected to validation:

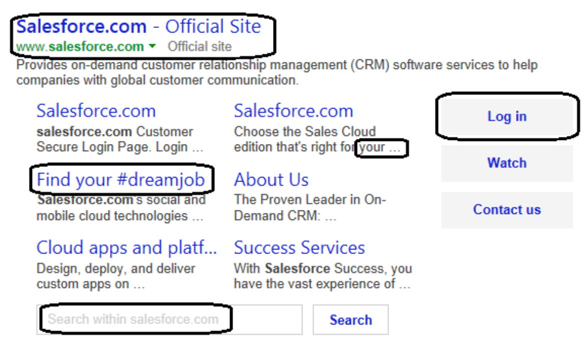

Figure 3. Validation aspects for web-search deep-links

There are two types of tests that can be used in this model:

1) *Custom Tests* are specific for only certain states (or actions). For instance, the deep-links validation shown above is an example of custom test since it only applies to pages originated from navigational queries
2) *Invariant Tests* verify general invariants that should always be true (or always be false) no matter what state we are

Invariant Tests are very powerful since they apply to all states (or actions). It is important and recommended that the product being tested be properly instrumented with test hooks in order to enable invariant conditions that can then be tested through invariant tests. An example of an invariant test would be a test that looks for java script errors. No state (or action) should lead to a Java script error on the page. We instrumented some of the Bing pages so that whenever inside the Microsoft corporate network and when a certain query string parameter is passed in the URL, any Java script error is caught via a global try/catch and written into a hidden HTML div tag [5]. With such instrumentation implemented, the invariant test for java script errors becomes trivial – basically checking for the presence of the java script error div tag. Other types of invariant tests are:

a) Links: no links should lead to 404 pages
b) Server Error: no state/action should lead to server errors
c) Security: no state/action should expose any security flaw (such as cross-site scripting [13])
d) Overlapping: no state/action should contain overlapped elements

Security invariants for instance are implemented by scanning the page and attempting to exploit potential vulnerabilities. An example would be cross-site scripting [13]: all the links and JavaScripts on the page are exercised with custom parameters handcrafted in order to exploit cross-site scripting vulnerabilities. Since Selenium allows the test to actually open and run the browser, if a cross-site scripting vulnerability is found the monitoring validation can then detect it based on the handcrafted parameter passed to the link or JavaScript.

Selenium also provides a capability of taking the screenshot of the current page. This allows the engineers to implement image-based test methods, some of which can be custom methods (such as the rendering and placement of some objects on the page specific to certain contexts) or invariants (such as the space between blocks on the page). Also, it is important to notice that some of the methods only apply to certain contexts (browsers, devices, markets or experiments). In such cases, the test needs to verify that the current context is relevant for the test in question to be executed.

VII. METHODOLOGY

Combining all the approaches described in this paper, we come up with the following methodology for synthetic exploratory testing or monitoring of large-scale web systems:

1) Mine the logs to create the user's profile Markov Chain. A user profile reprsents the states, actions and states transitions based upon mining of the logs

2) Retrieve the percentage distribution of different contexts (browsers, devices, markets and experiments)

3) Create custom and invariant tests that adhere to the subscription-based model

4) Stochastically run through the Markov Chain using Selenium or Selenium Grid. When testing search engines a key aspect is the generation of relevant queries to be used. It can be a combination of top queries based on frequency as well as segment-specific queries (such as queries that trigger local results or movie results)

5) Sporadically (time-based) switch contexts based on the distribution from #2

6) At each state (and action), apply the subscription-based tests from the library (#3). Alert in case of failures.

We differentiate monitoring from testing in terms of running the tests post-production and pre-production, respectively. The approach can be used for either one. However, we prefer to have deterministic tests as a pre-production mechanism, leaving the non-deterministic ones (such as the stochastic ones based on Markov Chains) as a monitoring mechanism (post-production). Also, the different tests have different priorities, so not all the tests will lead to an escalation (usually the invariant ones are deemed higher priority than the custom ones).

As the approach above executes, over time the critical monitoring paths will certainly be covered. Given that the approach follows a weighted-probability model, the critical paths will be covered more often than the non-critical ones. That is desirable since in today's fast-pace development environment of large-scale web systems, only the critical problems (the ones affecting the vast majority of users) get real attention; others are treated as low priority. The stochastic model is an elegant way to ensure highly-probable coverage of critical scenarios, and yet also cover some low-key scenarios.

Below are two examples of invariant failures when the model was applied to Bing.com. We used a set of 5 high-end servers executing around 1,000,000 state transitions per day, and running over 100 validation methods (of which 15% were invariant ones). The first example (Figure 4) is an invariant that looks for HTTP 500 server errors, in this case, generated by a combination of experiment and different interactions with the site:

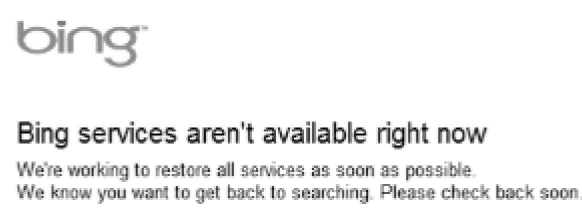

Figure 4. Issue discovered through an invariant test method

The second one is a low-priority invariant test based on image processing. In this case, the area to the right of the end of the search box should always contain background color only. But in the case of the German market, whenever search filters are present due to the long words in German, the placement of the filters (bottom left inside the top right rectangle) are going beyond the limits of the search box, breaking the pre-specified requirement (the requirements consist of User Interface principles and rules determined by designers that the code should always adhere to. In this particular case the specification clearly calls out that only background color can show up at the right side of the search box. Such a rule is violated in the case of German strings given that strings in the German language are usually longer than the ones in English). Figure 5 shows an example of such an issue:

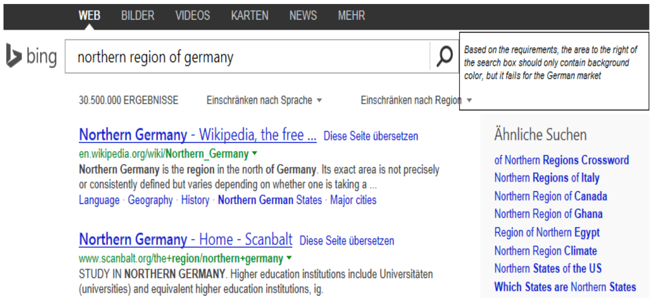

Figure 5. Example of an image-based error related to markets

Notice that the use of Markov Chains and context distributions allows the monitoring system to be highly adaptive: as the user patterns and context distributions change over time, the system will adapt itself based on the new data. The other important aspect is that the validation and monitoring mechanisms can certainly be extended to more than functional use, such as covering security concerns. At each step during the traversal of the Chain, we can also plug-in penetration tests which would be characterized as invariant methods.

VIII. TIME-BASED NEURAL NETWORKS

One of the limitations of Markov Chains is the fact that there is a need to introduce into the chains specific weighted-random events in order to account for the different contexts [11]. An alternative to overcome such limitation is to use a prediction model to, given a particular state and time for a user, predict the next state that the user is going to be based on the training data. The model that we selected was Artificial Neural Networks (ANNs [7]). The idea is to use features related to the current and previous states (pages), current and previous actions, current context, and generate the next most probable state, action and contexts. However, since the execution will have a temporal factor, there was an attempt to introduce a time-based feature into the ANN: the information about the user is segmented on a per-time unit, in our case every second. Such approach is similar to a Time Delay Neural Network (TDNN [8]). Mathematically speaking, the function F that the ANN will implement would then be:

F(States, Actions, Context, Time) = {State', Action', Context')

Here, we use the previous three states that the user has been previously (three previous pages visited), the corresponding three previous actions, the current context (which is a tuple consisting of browser, device, market and experiment) and the current time unit of the day in seconds (from 0 to 86,399). Figure 6 below depicts the ANN used.

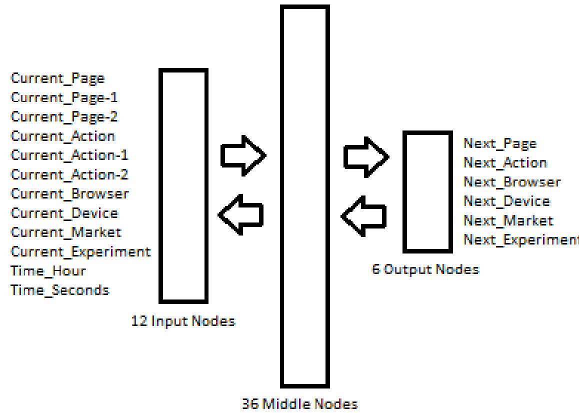

Figure 6. Time-based ANN

We decided to try the feed the network with the three previous pages and actions in order to have more accurate prediction. Trying with further states and actions did not improve its precision numbers (around 65% precision, see Table III below). We obtained the maximum precision numbers with the schema aforementioned: 12 input nodes where we split the time feature into two features: hour of the day (0-23) and seconds of the hour (0-59), 36 nodes in the middle (hidden) layer and 6 output nodes consisting of the state, actions, browser, device, market and experiment that the user should be in. During the execution, the ANN is fed with the current information about the user and the new information is given, changing the current user. Notice that the benefit of this new model is that it can be expanded to use other features too, and we do not need to rely on a weighted-random parallel mechanism to take into account the user's context. Also, given that the execution of the action usually takes over a second, it is very likely that the next input to the ANN will contain a different time parameter hence likely leading to a different output (the concern was that during the execution mode, the input would be consistently the same, hence producing the same output. It was not the case since the execution of each step took longer than 1 second). The learning model utilized was the error back-propagation [9]. We utilized a data set consisting of 1.5 million impressions in a 24h timeframe, proportionally sampled and distributed over the 24h window. 80% of this data was used as the training set whereas the remaining 20% was used as a test set.

With the Time-Based ANN fully trained, we swapped the model in the execution engine (Section VIII.4) with the Time-Based ANN. everything else in the methodology remained the same as described in Section VIII. Table III below depicts some comparative data between the Markov Chain Model and the Time-Based ANN model:

 ISBN: 978-1-61208-387-2

TABLE III. COMPARISON BETWEEN MARKOV CHAINS AND ANN FOR MONITORING

	Markov Chains	*Time-based ANN*
Training Time	~10min	~60min
Execution Time	400ms	30ms
Precision	N/A	65%
Min-Time-To-Failure (MTTF)	30min	108min

As one can see, due to the nature of the error back-propagation algorithm (with the high number of interactions for convergence), the time for the function to converge takes approximately six times longer compared to the training of the Markov Chain (which consists primarily of creating the weighted transitions). On the other hand, once the ANN is properly trained, its execution is significantly faster than the Markov Chain (likely attributed to the heavy weighted-random computations on the Markov Chains). The precision achieved for the ANN was not very high, around 65% for the test set, likely due to the fact that the time-based concept does not give a very predictable aspect to the back-propagation function despite its convergence. The Min-Time-To-Failure (MTTF) is characterized as the minimum time during the monitoring aspect to find the first monitoring failure or potential failure. In this aspect the Markov Chain converges faster than the Time-Based ANN. We believe this fact is related to the low precision for the Time-Based ANN. Our conclusion is that the Time-Based ANN gives a more elastic and expandable model where more features can be added in order to improve its precision; however the Markov Chain still gives the best outlook in terms of speed of training as well as better modeling the user's behavior. The Markov Chains are also significantly easier to implement compared to ANNs. Future work will be focused on augmentation of the ANN in order to improve its precision.

IX. CONCLUSION AND FUTURE WORK

Monitoring large-scale dynamic web sites across multiple browsers, devices, markets and experiments is a very complex task. In this paper, we have proposed a way to model the users' behavior via two stochastic methods: Markov Chains and Time-based Artificial Neural Networks. We compared these two methods in terms of their complexities, precisions and overall fitness for the problem of monitoring large-scale services. We combined Markov Chains and Selenium to recreate the same conditions experienced by real users in production. In addition, the validation approach is also changed from self-contained validation methods to a subscription-based model where the validation method subscribes to only the applicable states. Finally, validations can be invariant ones (applicable to all states) or custom ones (applicable to specific states). Future work will be focused on the time-based artificial neural network in order to achieve higher precision and better suitability for the problem of monitoring of web services. We presented an instance of the solution to monitor web search engines, but the same approach can be used to monitor other types of dynamic web services.

REFERENCES

[1] M. De Barros and C. Alex "Agile quality-centric development process of large-scale web systems", Swiss Testing Day 2014, March. 2014

[2] Gomez Network [Online]. Available from https://www.gomeznetworks.com/?g=1 2014.12.29

[3] Keynote [Online]. Available from http://www.keynote.com/ 2014.12.29

[4] M. De Barros, J. Shiau, C. Shang, K. Gidewall, H. Shi, and J. Forsmann, "Web Services Wind Tunnel: On Performance Testing Large-Scale Stateful Web Services", 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007

[5] The HTML <div> tag [Online]. Available from http://www.w3schools.com/tags/tag_div.asp 2015.01.12

[6] Selenium HQ Browser Automation [Online]. Available from http://docs.seleniumhq.org/ 2014.12.29

[7] Neural Networks [Online]. Available from http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html 2015.01.12

[8] Neural Network, Component of Measuring Set for Error Reduction [Online]. Available from http://www.measurement.sk/2004/S1/Vojtko.pdf 2015.01.12

[9] Error Backpropagation [Online]. Available from http://www.willamette.edu/~gorr/classes/cs449/backprop.html 2015.01.12

[10] E. Marcotte, "Responsive Web design", A List Apart, May. 2005

[11] M. De Barros, "Automated Synthetic Exploratory Monitoring of Dynamic Web Sites Using Selenium", PNSQC 2014, October. 2014

[12] R. Ramakrishnan, "Big Data @ Microsoft" [Online]. Avaiable from http://research.microsoft.com/en-us/events/fs2013/raghu-ramakrishnan_bigdataplatforms.pdf 2015.01.15

[13] S. Cook, "A Web Developer's Guide to Cross-Site Scripting" [Online]. Available from https://www.grc.com/sn/files/A_Web_Developers_Guide_to_Cross_Site_Scripting.pdf 2014.12.29

[14] Web Service Counters for the WWW Service [Online]. Available from https://technet.microsoft.com/en-us/library/cc786217(v=WS.10).aspx 2015.01.16

[15] User Experience at Google "Focus on the user and all else will follow", CHI 2008 Proceedings, April. 2008

[16] C. Liu, R. W. White, and S. Dumais, "Understanding Web Browsing Behaviors through Weibull Analysis of Dwell Time

[17] M. Arbib, A. Theories of Abstract Automata (1st ed.). Englewood Cliffs, N.J.: Prentice-Hall, Inc. ISBN 0-13-913368-2, 1969