Towards Stochastic Performance Models for Web 2.0 Applications

Johannes Artner, Alexandra Mazak(✉), and Manuel Wimmer

Business Informatics Group (BIG), TU Wien, Vienna, Austria
{artner,mazak,wimmer}@big.tuwien.ac.at

Abstract. System performance is one of the most critical quality characteristics of Web applications which is typically expressed in response time, throughput, and utilization. These performance indicators, as well as the workload of a system, may be evaluated and analyzed by (*i*) model-based or (*ii*) measurement-based techniques. Given the complementary benefits offered by both techniques, it seems beneficial to combine them. For this purpose we introduce a combined performance engineering approach by presenting a concise way of describing user behavior by Markov models and derive from them workloads on resources. By means of an empirical user test, we evaluate the Markov assumption for a given Web 2.0 application which is an important prerequisite for our approach.

Keywords: Web application performance engineering · Markov models · Queueing theory

1 Introduction

Performance quality describes the degree to which a Web application meets its performance requirements. Typically, performance is expressed in *response time*, *throughput*, and *utilization* of the application [10]. In this paper, we concentrate on performance quality with respect to *response time*, since this is a central characteristic for users [10]. It is quantified by measuring the time between sending a request and receiving the response (e.g., response time of components/operations).

There are approaches for performance evaluation that are typically used "off-line" like the traditional performance analysis cycle [8]. In practice, performance problems mostly occur after the development of an application which means when the application is deployed and running. In addition to off-line techniques *performance management* combines measuring, analyzing, and improving measures as well as automates their interplay [11]. However, performance management is often done ad-hoc by trial & error, rather than based on engineering

A. Mazak and M. Wimmer are affiliated with the CD-Labor MINT at TU Wien funded by the Austrian Federal Ministry of Science, Research and Economy (BMWFW) and the National Foundation for Research, Technology and Development (CDG).

principles [6]. Performance indicators as well as the workload of a system may be evaluated and analyzed by using (predictive) model-based or (descriptive) measurement-based techniques [5]. Both approaches have different limitations that may be mitigated by each other, e.g., to integrate empirical measures in predictive models over time to re-estimate these models and to use predictive models to quickly analyse performance aspects already in the design phase instead of running costly and time-consuming tests on the implementation level.

The essential and most basic elements for describing the performance characteristics of Web applications are (*i*) resources that offer computing capabilities, (*ii*) the workload that describes how the resources are being used, and (*iii*) the workload intensity in terms of inter-arrival times. Based on these elements, we introduce an stochastic-based approach for evaluating the performance of Web Applications by systematically combining model-based and measurement-based performance techniques. Thereby, we measure, predict, and evaluate performance indicators by timely responses of components. We outline the approach by means of a real example for a typical Web 2.0 application. Furthermore, we present a new concept for describing user behavior by Markov models to derive workloads on resources.

The remainder of this paper is structured as follows. The next section presents an overview of relevant concepts underlying our approach. In Sect. 3, we introduce the main concepts of the approach. In Sect. 4, we present a Web 2.0 application designed for travellers and an empirical user test based on this application. In Sect. 5, we discuss related work, and finally, we conclude with an outlook on future work in Sect. 6.

2 Background

In this section, we briefly describe the theoretic background and main building blocks necessary for the context of this paper.

Performance Engineering. PE is an engineering discipline within systems engineering. It represents the whole collection of engineering activities and related analysis used throughout the development cycle to meet performance engineering requirements. In this context, it is necessary to distinguish between two types of models, *design models* and *performance models* [21]. Design models primarily focus on system topologies, workflows and interactions between entities in a prescriptive way. On the other hand, performance models focus on the evaluation of performance by using simulations, or other analytical means (i.e., queueing networks). As mentioned before there are two different approaches for performance evaluation, the *model-based* and the *measurement-based* ones. Typically, the model-based approach is carried out in an early project phase to predict the performance quality of the system, while the measurement-based approach is performed after the system is deployed [21].

Markov model. A Markov model is a stochastic model that fulfills the Markov property which implies that the future is independent of the past, given the present [7]. This means that all necessary information is encoded in the present state, and therefore, information about the past is not needed. This is why the Markov property is also known as "memoryless". A more formal definition of the Markov property is given here:

$$\Pr(X_{t_k}|X_{t_{k-1}}, X_{t_{k-2}}, ..., X_{t_1}) = \Pr(X_{t_k}|X_{t_{k-1}}) \tag{1}$$

where t_k denotes a set of times $t_k > t_{k-1} > ... > t_1$.

Many theories are built upon this simplified Markov model of first order. However, for modeling real-world systems it is important to check the Markov assumption, whether the Markov property holds or not. The simplest form of a Markov model is a *Markov chain*. Another special form of a Markov model is a *Hidden Markov Model (HMM)*, where states cannot be directly observed, and therefore, they are "hidden".

Queueing theory. The queueing theory provides a formalism to describe systems in which "waiting" plays a key role. Waiting lines occur whenever the demand exceeds the service availability. The main goals of queueing theory are prediction as well as proposing design improvements of systems. There are two basic elements in a queueing system: (*i*) a number of (limited) resources that are capable of executing tasks, called *servers*, and (*ii*) *customers* that request tasks handled by servers. In terms of software systems, a server might be, e.g., an image server or a CPU. Examples for customers in the context of Web applications are users browsing a website.

3 Stochastic Performance Models for Web Applications

In this section, we present our approach based on two metamodels to systematically combine model-based with measurement-based techniques. This hybrid approach bases on the assumption that the software development process is done in an agile manner. In this context "agile" means that parts of the software system can be continuously delivered to a test- or production environment in which the running software can be profiled. This means that as much information as possible is gathered from continuously observing parts of the running system. Based on these observations a mix of predictive models and measured data can be derived for computing performance indicators.

CETO (Components Emission and Timely Observations). CETO represents the central data format. It is enriched over time with static information like the functionality and topology of the system as well as predictive and measured data of user behavior. After every development iteration the system is deployed and profiled. In particular, CETO tracks the observations of use case and operation calls and the duration time of operations. There are existing models such as the *Core Scenario Model* [14], or the *Performance Model Interchange Format* [19] that also provide capabilities to model static and predictive data, but no measurement data during development.

MUPOM (Markov Usage Process and Operation Measurements). MUPOM is designed with the goal to model stochastic processes of user behavior. We use this model to describe user behavior by applying the Markov formalism. A single state in this model represents a single use case. The user behavior combined with the workload intensity represents the workload of the system. This workload is described by arrival rates of resources, which we compute by using *queueing networks (QN)*. The arrival rate of resources is determined by external and internal arrivals. There are three available distributions to model these arrivals: Poisson, Exponential, and Normal distribution.

Transfer CETO to MUPOM. In order to combine the CETO and MUPOM metamodels, we apply *model-to-model transformations* as introduced in Berardinelli et al. [2]. We apply these transformations to transform: (*i*) the use cases and operation topology to CETO, (*ii*) CETO to MUPOM, and (*iii*) MUPOM to QN. This means that all necessary information is kept in a CETO model and solved by transforming it to a MUPOM model. To utilize these transformations the following ten steps are needed: (*i*) to (optional) classify user types, (*ii*) to (optional) unveil the underlying Markov chain, (*iii*) to canonicalize operation durations, (*iv*) to calculate probability distributions for think times and operation durations, (*v*) to construct the transformation matrix P_{ij} of the Markov model, (*vi*) to check whether the Markov property holds, (*vii*) to calculate the steady state distribution, (*viii*) to (optional) derive the causal orderings and parallelisms of operations, (*viv*) to check whether the system satisfies the product form assumption of QN, and finally, (*vv*) to construct a performance model.

Transferring Markov Models to QN. Resource utilization in software systems is mainly a result of user behavior. The MUPOM model, besides the Markov model itself, also contains the solution for a steady state if it exists. A Markov chain has a limiting distribution if it is irreducible and all its states are positive recurrent [17]. If such a limiting distribution exists, it can be referred to as *steady state* that describes the long-run behavior of a system independent of the starting state. For calculating the steady state in our approach we use approximations.

In order to transfer a Markov models to QN, we have to achieve the characteristics of an *ergodic* system. Such a system requires the following assumptions that must hold: (*i*) *irreducibility*, if it is possible to reach each state from any other state; (*ii*) *aperiodicity*, if the system state is not systematically connected to time; and (*iii*) *recurrence*, if all states are recurrent. The reasons therefore are defined in *Little's law*, which is a very prominent operational law applied to queueing theory. It states that the long-term average number of customers in a stable system L is equal to the long-term average effective arrival rate λ multiplied by the average time a customer spends in the system W; or expressed formally: $L = \lambda W$.

Given these findings, we are able to translate MUPOM models to QN. In a first step, we calculate the average number of users in a system by applying Little's law:

$E(N) = \lambda * E(T)$. In a second step, we calculate the average number of users in each state by the approximated steady state distribution (π): $N_i = \pi_i * E(N)$.

The Markov chains in the MUPOM model are described in discrete time. The time step is in steps of seconds. Since, we know how many users are in each stage on average, the transition rates between each state can be simply calculated by multiplying the average number of users with the corresponding transition probability: $\lambda_j = p_{ij} * N_i$. In a final step, we calculate the total number of new enterings on average to a state per second and sum up all of these incoming transitions: $\Lambda_J = \sum_{j=1}^{k} \lambda_j$. The final outcome is the sum of all arrival rates to a certain state. By the inverse, we get the inter arrival rate, which forms together with the service rate, the basis for analyzing queuing networks for the purpose of performance evaluation.

4 Evaluating the Markov Assumption for an Example Application

In order to combine the model-based with the measurement-based approaches of performance engineering, we defined a transformation chain based on ten steps (cf. Sect. 3). Step (*vi*) of this chain checks whether the Markov property holds when transferring CETO to MUPOM. Due to page limitations, we primarily focus on this step in our case study. The evaluation of the validity of the Markov assumption is an essential pre-condition that must hold before we can check whether the Web application satisfies the product form assumption of QN to finally construct a stochastic performance model. For more insights, we refer the interested reader to our project website[1] where all artifacts of the case study, including the Travelistr software, are available.

Research question. *Is it sufficient to describe the transition probability between pages/features of a Web 2.0 application with Markov models and does the transition probability fulfill the Markov assumption?* Thereby, we assume that Web 2.0 applications are ergodic system. And if not, they can be transferred to one.

Case Study Design. As an essential part of our case study, we evaluate the Markov assumption for an example Web 2.0 application. For this purpose we use the approximative approach introduced in Li et al. [13] as a basis. In particular, we compare transition probabilities of a Markov model of first order to transition probabilities of a Markov model of second order. If the probabilities of order one and two are not diverging more than a certain threshold, the Markov assumption is fulfilled.

Selected Web 2.0 Application. *Travelistr* is a Web 2.0 application designed for travellers. It enables people to share pictures of their journey with other travellers that are nearby. Pictures can be enriched with information, e.g., of the geographic location of the photograph. Interested users get three pictures and

[1] http://www.johannes-artner.at/#ASPE.

are able to like one of them. Travelistr is a JavaEE application based on Spring Web MVC. For database access, Hibernate is used together with the `c3p0` connection pool. The frontend is created with static HTML pages and dynamic JSP pages. `Log4j` is used as logging framework and `OperationsAndTraceMonitor` instructions are added at the language level. Travelistr is hosted at an Apache Tomcat server. The data is stored in a `PostgreSQL` database and the images are stored using the external vendor `Cloudinary`.

Tool Setting for Data Collection. Based on the CETO and MUPOM metamodels, we implement two Java-based tools to track and analyze observations, the *OperationsAndTraceMonitor* tool and the *UserTrace2Markov* tool. The observational data is gathered by using the *UserTrace* component of the *OperationsAndTraceMonitor* tool. It writes these observations to a CSV-file in a thread-safe manner in order to provide an appropriate format to use it as input to other tools. The *UserTrace2Markov* tool is a semi-automatic tool to transform user traces, tracked with the *OperationsAndTraceMonitor*, from the CSV-format to a Markov model of first and second order.

Results. We represent the results of applying our approach for the given set up. The user test took place for one week and in total 32 users participated. The participants of our evaluation study were students from our institute. They uploaded 146 pictures, and 839 *Likes* were received. 4520 transitions between states were observed within 173 distinct user interactions. Two consecutive transactions from a user are considered to be in the same user interaction if their timely difference is equal or less than 120 s. In its first version, the action of the *UserTrace2Markov* tool can only be of type *GET*. The results of the analysis are written to a txt-file (i.e., path to the output file). The tool calculates transition rates between states for a first and second ordered Markov model. Furthermore, think times in every state are calculated and described by mean values on basis of a standard normal distribution. Additionally, the total aggregated time spent in each state by every user is summed up. This is useful for analyzing the steady state assumption. The figure showing the results of the empirical user test is available for downloading under http://www.sysml4industry.org/?download=811.

To distinguish users, a randomly generated ID was assigned to each session. A user session was never invalidated during the whole test. So, if a user hasn't logged out on purpose, the user could continue where she left previously without logging in again. Table 1 shows the results of this approximative evaluation of the Markov assumption. A sample trace of every state and transition logged during the user test is available at http://www.sysml4industry.org/?download=822.

Interpretation of Results. We conclude that Markov models of order one are good approximations for describing the user behavior of typical Web 2.0 applications. The Markov assumption is an important prerequisite and holds for the Travelistr-App. Thus, it can be assumed that it also holds for other Web 2.0 applications.

Table 1. Results of comparing the first-order with second-order Markov model solutions.

State	Avg. information loss from 2nd to 1st order model	Observed transitions (n)
Nearby	4.36%	1067
Dashboard	11.45%	506
Profile	6.72%	193
Publish	2.87%	190
Published	1.77%	153
Start	7.58%	128
Login	3.14%	106
Register	12.48%	35

Threads to Validity. The empirical user test for evaluating the Markov assumption was successful for the given example. However, only 32 users took part in the test and not every state/transition was visited as often as desirable. We are aware that this sample is not large enough, so that any bias may change the conclusion. To reduce the risk of evaluating insufficient data, only states and their respective transitions with more than 20 observed transitions should be taken into consideration for evaluating the Markov assumption. Furthermore, there is a risk that the users did not use Travelistr as they would, if Travelistr would be an application they would use in their daily life. An additional risk is the time period of the conducted user test. It can be assumed that the behavior of users over a longer period accordingly will change, e.g., due to a better understanding of the system. The fractions of some states (e.g., *Register*) is therefore assumed to be different in the long run. And last but not least the results may differ if a Web application with more functionality had been considered.

5 Related Work

There are measurement methods that need performance probes of different levels (e.g., resource level, application level), or request-based techniques to estimate specific parameters of performance models. Kraft et al. [12] present a high-level request-based measurement technique to estimate service resource consumptions based on two approaches, a linear regression method and a maximum likelihood function. Their approach has several advantages compared to lower-level approaches, e.g., request measurements are easy to obtain. Barham et al. [1] present a tool chain, named “Magpie”, which extracts the workload of a software system based on hardware, middleware, or application level traces. The resulting traces are correlated to requests. The computing demand for the requests is calculated and dependencies are ordered based on causality. The results of this procedure are used as basis for computing performance models. *Kieker* [16] is a framework to continuously monitor and analyze a system's runtime behavior. It has two main components, *Kieker.Tpmon* and *Kieker.Tpan*. Kieker.Tpmon monitors and logs data, while Kieker.Tpan is an analysis component which logs the runtime behavior to reverse engineer it (e.g., by sequence diagrams, class diagrams).

Markov models and QN are two formalisms widely used for performance engineering of systems as, e.g., introduced in [3,6,7]. Also, Hidden Markov models (HMM) have been used in many different fields, predominantly in the field of speech recognition [15]. To use HMMs for performance evaluation of software systems is a relatively new application field [18]. Hoorn et al. [20] introduce approaches based on statistical modeling for generating probabilistic and intensity-varying workloads for Web-based software systems. They present three models: (*i*) an *application model* to specify possible sequences of usage by a hierarchical finite state machine, (*ii*) a *user behaviour mix-model* to define which user behaviors are used for computing a workload, and (*iii*) a *workload intensity model* to specify the varying numbers of users over time. Based on these models, the authors extend JMeter to a probabilistic-based workload generator. Jespersen et al. [9] evaluate the Markov assumption for mining Web usage. The authors examine the quality of rules derived from two example websites by using the *Hypertext Probabilistic Grammar (HPG)* model introduced in [4], which relies on the Markov assumption and is mostly used for website analysis. The authors define *similarity* and *accuracy* for quality evaluation. On the one hand, similarity compares the amount of rules that are derived by using HPG and that are equal to the true usage patterns. On the other hand, accuracy compares the derived probabilities of the rules with the true usage patterns. As a result, the authors suggest that Markov-based approaches are better suited for tasks which require less accuracy. In addition, Li et al. [13] present a simple approach for evaluating the Markov property of Markov chains to model user behavior. In their approximative approach, they compare transition probabilities only considering the current state to transition probabilities considering the current and one previous state. They demonstrate their approach for Web usage profiling.

6 Conclusion and Future Work

Using Markov models to express user behavior is not new. However, to express the usage profile by Markov models in a model-driven way by model-to-model transformations is to the best of our knowledge a novel idea. We systematically combine the model-based and the measurement-based approach in an agile manner. We use Markov models to define the user behavior of a system in a predictive way. This implies that the underlying stochastic processes of user behavior fulfill the Markov assumption. Therefore, we conducted an empirical user study to show that this assumption holds for our implemented Web 2.0 application.

We outlined how to retrieve QN from Markov models based on user behavior. For this, necessary conditions have to hold. In addition to the Markov assumption, this is the ergodicity of the system. For Travelistr, both conditions hold.

Other software system types may be of the same nature. In this context, it is required to understand the characteristics of software systems that are ergodic. Furthermore, it is of interest to extrapolate observed user interactions of a limited set of features to a user behavior model of an entire system, and how the measurements can be validated. The practical benefit would be that usage patterns can be quantified already during development. Besides enhancing the accuracy of operation durations, also the user behavior model may be adapted and enriched with findings from measurements. A possible solution might be to mock the behavior of missing features.

References

1. Barham, P., Donnelly, A., Isaacs, R., Mortier, R.: Using Magpie for request extraction and workload modelling. In: OSDI (2004)
2. Berardinelli, L., Maetzler, E., Mayerhofer, T., Wimmer, M.: Integrating performance modeling in industrial automation through AutomationML and PMIF. In: INDIN (2016)
3. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. Wiley, Hoboken (2006)
4. Borges, J., Levene, M.: Data mining of user navigation patterns. In: Masand, B., Spiliopoulou, M. (eds.) WebKDD 1999. LNCS, vol. 1836, pp. 92–112. Springer, Heidelberg (2000). doi:10.1007/3-540-44934-5_6
5. Franks, G., Al-Omari, T., Woodside, M., Das, O., Derisavi, S.: Enhanced modeling and solution of layered queueing networks. IEEE TSE **35**, 148–161 (2009)
6. Harchol-Balter, M.: Performance Modeling and Design of Computer Systems: Queueing Theory in Action, 1st edn. Cambridge University Press, New York (2013)
7. Hevizi, G., Biczó, M., Poczos, B., Szabo, Z., Takics, B., Lorincz, A.: Hidden Markov model finds behavioral patterns of users working with a headmouse driven writing tool. In: IJCNN (2004)
8. Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley, New York (1991)
9. Jespersen, S., Pedersen, T.B., Thorhauge, J.: Evaluating the markov assumption for web usage mining. In: WIDM (2003)
10. Kappel, G., Pröll, B., Reich, S., Retschitzegger, W.: Web Engineering. Wiley, New York (2006)
11. Kotsis, G., Pinzger, M.: AWPS – an architecture for pro-active web performance management. In: Hummel, K.A., Hlavacs, H., Gansterer, W. (eds.) PERFORM 2010. LNCS, vol. 6821, pp. 215–226. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25575-5_18
12. Kraft, S., Pacheco-Sanchez, S., Casale, G., Dawson, S.: Estimating service resource consumption from response time measurements. In: VALUETOOLS (2009)
13. Li, Z., Tian, J.: Testing the suitability of Markov chains as Web usage models. In: COMPSAC (2003)
14. Petriu, D.B., Woodside, M.: An intermediate metamodel with scenarios and resources for generating performance models from uml designs. SoSyM **6**(2), 163–184 (2007)

15. Rabiner, L.: A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE **77**(2), 257–286 (1989)
16. Rohr, M., Van Hoorn, A., Matevska, J., Sommer, N., Stoever, L., Giesecke, S., Hasselbring, W.: Kieker: continuous monitoring and on demand visualization of Java software behavior. In: IASTED-SE (2008)
17. Serfozo, R.: Basics of Applied Stochastic Processes. Springer, Heidelberg (2009)
18. Souza e Silva, E., Leão, R.M.M., Muntz, R.R.: Performance evaluation with hidden Markov models. In: Hummel, K.A., Hlavacs, H., Gansterer, W. (eds.) PERFORM 2010. LNCS, vol. 6821, pp. 112–128. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25575-5_10
19. Smith, C.U., Lladó, C.M., Puigjaner, R.: Performance Model Interchange Format (PMIF 2): a comprehensive approach to queueing network model interoperability. Perform. Eval. **67**(7), 548–568 (2010)
20. Hoorn, A., Rohr, M., Hasselbring, W.: Generating probabilistic and intensity-varying workload for web-based software systems. In: Kounev, S., Gorton, I., Sachs, K. (eds.) SIPEW 2008. LNCS, vol. 5119, pp. 124–143. Springer, Heidelberg (2008). doi:10.1007/978-3-540-69814-2_9
21. Woodside, M., Petriu, D.C., Merseguer, J., Petriu, D.B., Alhaj, M.: Transformation challenges: from software models to performance models. SoSyM **13**(4), 1529–1552 (2014)