Modelling Multi-tier Enterprise Applications Behaviour with Design of Experiments Technique

Tatiana Ustinova
Imperial College London
Exhibition road, South Kensington
London, UK, SW7 2AZ
tatiana.ustinova12@imperial.ac.uk

Pooyan Jamshidi
Imperial College London
Exhibition road, South Kensington
London, UK, SW7 2AZ
p.jamshidi@imperial.ac.uk

ABSTRACT
Queueing network models are commonly used for performance modelling. However, through application development stage analytical models might not be able to continuously reflect performance, for example due to performance bugs or minor changes in the application code that cannot be readily reflected in the queueing model. To cope with this problem, a measurement-based approach adopting Design of Experiments (DoE) technique is proposed. The applicability of the proposed method is demonstrated on a complex 3-tier e-commerce application that is difficult to model with queueing networks.

Categories and Subject Descriptors
C.2.4, C.4, D.2.8, D.4.8

Keywords
Multi-tier enterprise applications, design of experiments, two-level factorial designs, response surface models, linear regression, software performance testing

1. INTRODUCTION
DevOps is defined as a set of practices and principles bridging the gap between application development and operation stages [8]. One way to achieve this is continuous application performance modelling and prediction combined with automated feedback of the models to the developer and their update via continuous testing. A large body of work exists that employs Machine Learning algorithms [9] and tools, as well as linear regression, to obtain performance models based on measurements. In this paper we propose application performance modelling and prediction algorithm based on the Design of Experiments (DoE) technique.

DoE – widely used in engineering and industry for optimising processes – looks very promising for the use in DevOps, as it utilises measurements obtained at runtime to build performance models. These models can be fed to the application developer and updated in an automated way through continuous testing. However, its use is rather sparse in computer science, especially in the area of application performance modelling and prediction. This technique involves choosing a number of input parameters called 'factors', designing a set of experiments and then carrying them out on the system-under-study. The experiment results, called 'response variables', are then used to construct linear regression model representing a relationship between system output ('response variable') and inputs (factors). In this approach system under study is treated as a black box.

A number of studies exist that explore the capabilities of the DoE technique and DoE-based models in performance modelling, evaluation and prediction. Li et al. [4] presented a factor framework for performance evaluation of commercial Cloud services. This framework establishes factors that are currently used in the performance evaluation of clouds and can help facilitate designing new experiments for evaluating cloud services. However, this work does not provide any quantitative or qualitative assessment allowing to conclude which of these factors may be important for software performance testing.

Westerman et al. [10] apply statistical inference techniques to adaptively select experiments resulting in the optimal performance model. The approach automatically selects and conducts experiments based on the accuracy observed for the models inferred from the currently available data. The results demonstrate that this approach can automatically infer a prediction model with a mean relative error of 1.6% using only 18% of the measurement points in the configuration space. However, this work is focused only on the design of experiments and does not investigate predictive capabilities of the obtained model.

Molka and Casale [in revision] applied DoE techniques to generate response surfaces (non-linear models constructed using linear regression) that describe database performance as a function of workload and hardware parameters for in-memory databases. The response variables this study reported include response times, server utilisation, energy consumption and memory occupancy. They found out that the queueing network and response surface models yield mean prediction errors in the range 5%-22% with respect to response times and mean memory, but the accuracy for the latter deteriorates in response surfaces as the number of experiments are reduced, whereas model-based simulation is effective in all cases. This suggests that simulation can be more effective in performance prediction for in-memory database management. However, this queueing network model was tailored to describe in-memory database, which required significant effort and knowledge of the system under study.

The proposed method described in details in the following sections is based on the design of experiments technique, which is first used to establish the design space (screening procedure) – a set of factors and their low and upper bounds – that influence response variable(s). Then a linear regression is used to construct a model describing relationship between input parameters and performance metrics based on the experiment results obtained

QUDOS'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804374

during the screening. Afterwards, the model prediction accuracy is assessed. Additionally, the model prediction error is then compared to prediction made by the out-of-the-box Mean Value Analysis algorithm for queueing network models. To the best of our knowledge none of the work presented in this paper has been done before.

The rest of the paper is organised as follows. Section 2 presents the methodology of the proposed approach; Section 3 is dedicated to the case study – load simulation for the web-based e-commerce 3-tiered application; Section 4 provides analysis of the model prediction accuracy and discussion of the analysis results; Section 5 draws conclusions and gives suggestions for further work.

2. METHODOLOGY

Design of Experiments (DoE) starts with determining the objectives of an experiment and selecting the factors for the study. The choice of the experimental design would influence the amount of runs required to obtain sufficient information about system under test [1]. For example, if software performance tester is interested only in identifying the parameters that significantly influence application's performance, then two-level factorial design would suffice. The objective in this case would be to find out parameters (factors) that cause significant change in the output by shifting from one (low) level to another (high). Additionally, because in order to investigate all possible combinations of levels, 2^k runs (where k = number of factors in the experimental design) would be needed, the so-called fractional factorial designs are often used, where only a part (fraction) of the 2^k (full factorial) design is used. These designs, however, should be treated with care, as they are constructed under a number of assumptions, which may not hold for the given system.

Two-level factorial designs are also widely used for construction of linear regression models, but their use implies that relationship between system inputs and output is linear. If there is a chance that this relationship is not linear, other designs, allowing to construct polynomial regression models (e.g. Response Surface Methodology), might be considered instead. Therefore, it can be summarised that well-chosen experimental design would involve minimum possible number of runs required to obtain necessary information about the system under test. Also on this step response variables should be agreed on.

Taking into consideration everything said above, the following actions are needed to implement the method:

a) Define response variables: those would be performance metrics (e.g. response time, CPU utilisation, throughput);

b) Create design space via screening for important factors and their interactions using two-level fractional factorial design: choose a number of factors that might influence performance metrics, set the low and high levels for them (the levels are chosen based on the experimenter's experience and knowledge of the system).

c) Validate results of the screening with full factorial design for the chosen subset of important factors (may or may not require additional runs) and allocation of variation [7]. Allocation of variation shows how much variation each of the factors causes in the response variable when changed from low to high level.

d) Construct linear regression model based on the experiment results from b) and c) (may or may not require additional runs).

3. CASE STUDY

3.1 Objective

The objective of this case study is to build the model allowing to describe and predict performance of the web-based 3-tier e-commerce application following the methodology presented in the Section 2. The outputs (response variables) considered are application response time and CPU utilisation.

3.2 Test Environment

3.2.1 Testbed Description

The testbed consists of workload generator syntactically generating requests to a backend web-based application. Experiments in this study were performed using model-driven workload generator called MDLoad [5]. MDload automatically generates requests to an application under test by simulating a set of users. Since the workload generator needs to create considerable number of virtual users, MDLoad was deployed on a Virtual Machine (VM) with 12 CPU and 3GB of memory. This VM is located on the private cloud at Imperial College London. The hosts of the private cloud are Intel Xeon with CPU E5-2450 2.10 GHz. The capacity of the VM machine was chosen based on the previous experience with MDload such a way that relatively small number of users would saturate the application, resulting in significant increase in application response time and CPU utilization, but not leading to the MDload outage. This decision allowed to reduce execution time needed for each experimental run while still obtaining sufficient samples to estimate mean values of performance metrics.

The software stack of workload generator comprises JAVA and shell scripts for submitting HTTP requests and controlling the behaviour of virtual users by creating session-based workload. The request composition of the sessions for the three MDload user classes adopted in this study is shown in Table 1:

Table 1: Request mix per session for 3 MDload user classes.

Request	Class I (light)	Class II (medium)	Class III* (heavy)
Home	+	+	+
Login	+	+	+
Login details	+	+	+
Main		+	+
Order History		+	+
QuickAddMain		+	+
CartAddAll		+	+
Checkout		+	+
CheckoutAddressNext		+	+
CheckoutPaymentNext		+	+
CheckoutShippingNext		+	+
Logout	+	+	+

*Class III has higher number of Checkout requests per session than Class II

3.2.2 Application Under Test

The application under test is Apache OFBiz [6] - an open source web-based e-commerce system. The OFBiz instance is

deployed on a VM with 1 CPU and 3GB of memory on the same private cloud at Imperial College London. Keeping both workload generator and backend application on the hosts in the same private cloud and connected through high-speed broadband network allows to remove 'noise' in the system response time (collected on the MDload side using tool's features) caused by network latencies. Therefore, measurements for the system response time can be considered response time on the application level.

3.3 Screening Procedure

There are a number of parameters (in DoE known as 'factors') that might influence application performance. These may be external inputs, such as, for example, number of users, user think time, or system parameters (e.g. hardware configuration on which application is deployed). Such parameters may be controllable (can be changed by the experimenter) and uncontrollable. For example, network delay, mentioned above, can be viewed as the noise factor, influencing response time as it is experienced by the user. An extensive taxonomy of factors is given in [4]. However, to explore all possible combinations of these factors would require 2^k experimental runs, as was mentioned in the Section 2. In the example from [4] that would be $2^{38}=274\text{x}10^9$ runs, which is, of course, infeasible. Therefore, not only fractional factorial design is needed, but also careful consideration for the choice of the candidate factors for the screening procedure, based on the experience of the experimenter.

In this study it was decided to start with a small set of well-known factors, such as number of users, user think time and workload mix. Additionally it was tested if the testbed set up described in 3.2.1 would allow to decrease execution time of the experimental run without causing deterioration of estimates. The low and high levels for the number of users were chosen based on the N*, where N*, following the definition from [3] is the point where application starts exhibiting saturation behaviour. Levels for other factors for the two-level design were chosen based on the authors' experience with application load testing and MDload. The summary of factors and their levels chosen for the screening procedure is given in the Table 2.

Table 2: Factors and their levels.

	Levels	
	Low (-1)	**High (1)**
Number of users*	3**	20
User think time, s	10	1
Execution time, min (steady state)	10	30
Workload mix (user class)	I	III

* N*=16 for user think time = 5 s.

** N_users = 3 instead of 1 is chosen to obtain more samples for averaging.

To investigate all possible combinations of these four factors would require $2^4=16$ runs, which is theoretically feasible, but with half of the runs requiring execution for 30 minutes it would take 5 h 20 min. Therefore it was decided to use fractional factorial design in line with the commonly-used procedure. It is important to note, though, that the price for the reduction in runs is so-called confounding of effects. This means that the effects (factors and their interactions) estimated based on the results of fractional factorial design are a combination of two or more effects. Hence it is important to choose fractional factorial design in such a way so that main effects are confounded with higher-order interactions. The higher-order interactions (interactions of N-1 factors in design for N factors) are generally considered negligible. The fractional factorial design of resolution IV (all main effects will be confounded with higher-order interactions, low order interactions will be confounded with each other) for the example data from Table 2 along with the confounding pattern is presented in Table 3:

Table 3: Fractional factorial design for four factors.

Exp. run	Number of users (A)	Think time (B), s	Execution time (C), min	User class (D)	Confounding pattern
1	3	10	10	I	I=I+ABCD
2	3	10	30	III	A = A + BCD
3	3	1	10	III	B = B + ACD
4	3	1	30	I	C = C + ABD
5	20	10	10	III	D = D + ABC
6	20	10	30	I	AB = AB + CD
7	20	1	10	I	AC = AC + BD
8	20	1	30	III	AD = AD + BC

As was mentioned above, higher-order interactions are considered negligible. Therefore, based on the results of the experimental runs it should be possible to make conclusion about significance of main effects (significance of interactions should be treated carefully as they are confounded with each other).

Response variables response time and CPU utilisation, obtained in the screening experiments can be analysed graphically (numerical analysis such as ANOVA or p-values is not recommended because of confounding). Example analysis for the response time is shown in the Figures 1 and 2.

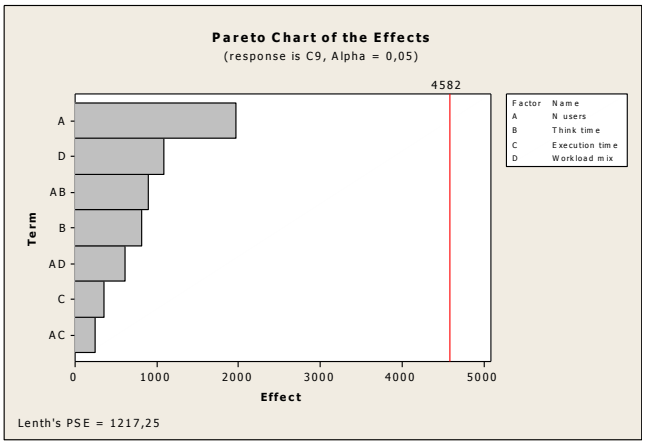

Figure 1. Ranking of effects.

On the Figure 1 estimated effects are ranked by their magnitude. Red line represents Lenth's PSE – pseudo-standard error. All effects that cross this line are deemed significant. From the Figure 1 it is obvious that none of the factors are deemed significant, which is suspicious, because from the Figure 2 it is seen that at least number of users, think time and workload mix make an impact on the response time. To investigate this problem 8 more runs were conducted to create a full factorial design for 4 factors. The results (for the response time) of the full factorial design for 4 factors are shown on Figure 3.

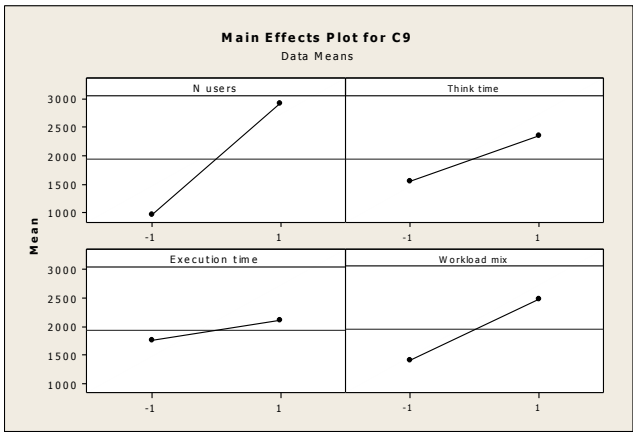

Figure 2. Main effects plot.

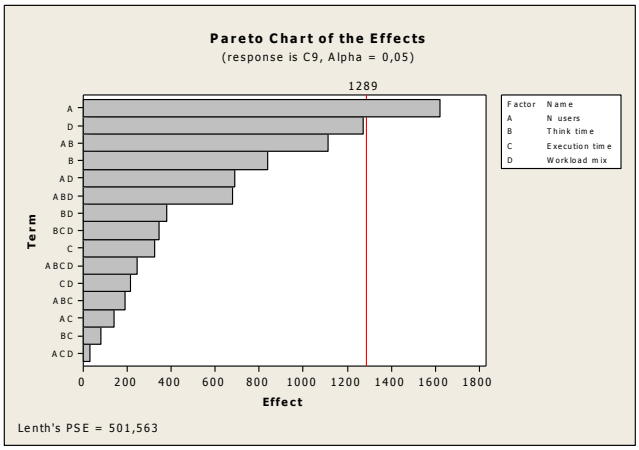

Figure 3. Ranking of effects.

It is clearly seen from the Figure 3 that number of users is significant, user class is close to significance, as well as the interaction between number of users and think time. Additionally, it can be seen that high-order interactions ABD and BCD (and even ABC) are not negligible as had been assumed. This resulted in the distortion of main effects and the value of Lenth's PSE, which is based on the effects' magnitudes. In the case of 4 factors, where at least two of them turned out to be significant (number of users and workload mix) as well as the two-way interaction for the third factor(think time), the combined influence ABD of these three factors turned out to be large. Such occurrence can be mitigated by screening for large number of factors, especially with deliberate addition of factors which should not be significant, because then there is a small chance that combined interaction of, e.g., 5 factors for 6-factor design would be present.

After screening test is conducted, and significant main effects are found, the full factorial design with replications should be conducted for this subset. If there are significant interactions (or close to significance), the factors that cause them also should be included into full factorial design, even if they themselves were not identified as significant. In the example think time (B) would be taken into the subset of significant factors, even though it is on itself wasn't flagged as significant, because the interaction AB (between number of users and think time) is very large. Execution time did not show any significant influence either on response time or CPU utilisation, therefore it was set at the low level (10 minutes). The full factorial design with 3 replications and response variables are presented in the Table 4. This design is needed to validate analysis conducted on the fractional factorial design stage and required 4 additional runs (2, 3, 5 and 8).

Table 4: Full factorial design for 3 factors.

Exp. run	N_users	Think time, s	User class (D)	Execution time, min
1	3	10	I	10
2	3	10	III	10
3	3	1	I	10
4	3	1	III	10
5	20	10	I	10
6	20	10	III	10
7	20	1	I	10
8	20	1	III	10

The analysis of results confirmed that all three factors, as most of their low-order interactions were significant. Additional analysis was conducted to estimate the allocation of variation: how much variation each of the factors causes in the response variable when changed from low to high level [7]. Variation of responses (in %) due to factors and their interactions is shown in Table 5:

Table 5: Variation of responses (in %) due to factors and their interactions.

Effect	Response time	CPU utilisation
N_users	26.03	54.27
Think time	4.53	42.99
User class	36.25	1.14
N_users:Think time	19.13	0.59
N users:User class	6.63	$7.886*10^{-6}$
Think time:User class	$1.5x10^{-8}$	$1.8917*10^{-4}$
N_users:Think time: User class	5.42	0.91
Error	2.01	$7.6946*10^{-4}$

Error term in the Table 5 contains both random error and influence of any factors that were not considered when constructing screening design. As this error term is very small for both response variables, it is safe to assume that all major sources of variation were identified.

3.4 Constructing the Model

As both response time and CPU utilization exhibit non-linear behaviour, Response Surface (RS) design, namely central-composite Box-Wilson design, was chosen. This design contains full factorial design for 3 factors and centre points, therefore can be used to construct both linear, quadratic and polynomial models. Additionally, the 'faced' configuration of the design was implemented. This configuration does not use points outside of the design space. The prediction capabilities of the model constructed based on this design can be worse than of a combination using the points outside the design space, but in our case this combination is impossible to implement (we can't go beyond user classes I and III). This design requires 24 runs in total (centre points are run 10 times to allow for a more uniform estimate of the prediction variance over the design space). The design is shown in the Table 6 (shaded area shows full factorial design):

Table 6: Box-Wilson central composite 'faced' design.

N	1	2	3	4	5	6	7	8	9	10	11	12
X_1	-1	-1	-1	-1	1	1	1	1	-1	1	0	0
X_2	-1	-1	1	1	-1	-1	1	1	0	0	-1	1
X_3	-1	1	-1	1	-1	1	-1	1	0	0	0	0
N	13	14	15	16	17	18	19	20	21	22	23	24
X_1	0	0	0	0	0	0	0	0	0	0	0	0
X_2	0	0	0	0	0	0	0	0	0	0	0	0
X_3	-1	1	0	0	0	0	0	0	0	0	0	0

As was mentioned above, chosen RS design allows to construct various types of regression models. We want to investigate how they fare in prediction. Summary of the constructed regression models is given in the Table 7:

Table 7: Regression models constructed from the experiment results and used in subsequent analysis.

Name	Description	Formula
Linear	Model contains an intercept and linear terms for each factor	$y=I+a_1x_1+a_2x_2+ +a_3x_3$
Interactions	Model contains an intercept, linear terms, and all products of pairs of distinct factors	$y=I+a_1x_1+a_2x_2+ +a_3x_3+a_4x_1{:}x_2+ +a_5x_1{:}x_3+a_6x_2{:}x_3$
Pure Quadratic	Model contains an intercept, linear terms, and squared terms	$y=I+a_1x_1+a_2x_2+ +a_3x_3+a_4x_1^2+ +a_5x_2^2+a_6x_3^2$
Quadratic	Model contains an intercept, linear terms, interactions, and squared terms	$y=I+a_1x_1+a_2x_2+ +a_3x_3+a_4x_1{:}x_2+ +a_5x_1{:}x_3+a_6x_2{:}x_3+ +a_7x_1^2+a_8x_2^2+a_9x_3^2$
Full Polynomial	Model is a polynomial with all terms up to degree 3 in the first factor, degree 3 in the second factor, and degree 3 in the third factor*	$y=I+a_1x_1+a_2x_2+ +a_3x_3+a_4x_1{:}x_2+ +a_5x_1{:}x_3+a_6x_2{:}x_3+ +a_7x_1{:}x_2{:}x_3+a_8x_1^2++a_9x_2^2+a_{10}x_3^2+ +a_{11}x_1^2{:}x_2+ +a_{12}x_1{:}x_2^2+ +a_{13}x_1^2{:}x_3$

*x^3 terms are zero, the third level was chosen to include 3-way interaction between number of users, think time and user class into the model.

Prediction curves R=*f*(N users) and U_cpu=*f*(N users) were constructed for each model type for every combination of user class and user think time. As an example, the curves for 'full polynomial' model type and user class III are shown in the Figures 4 and 5.

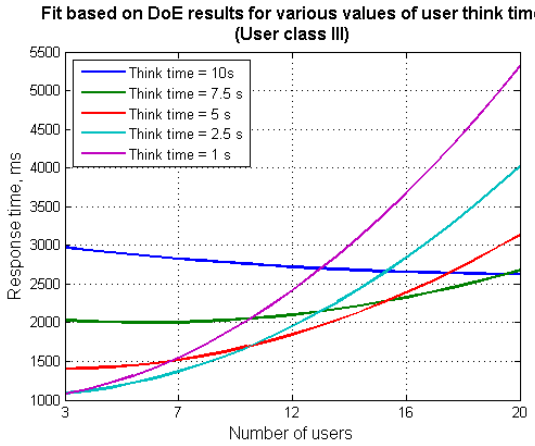

Figure 4. Prediction for the response time.

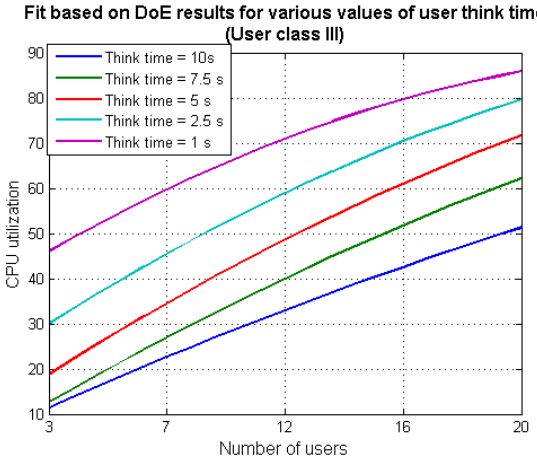

Figure 5. Prediction for CPU utilization.

4. ANALYSIS AND DISCUSSION

4.1 Collect Independent Observations.

In order to assess the model prediction capabilities, a series of experiments with parameter values from the design space was run. One experiment point was run per each prediction, i.e. pair {User think time, user class}, for N users = 16. The points collected for the model verification are given in the Table 8:

Table 8: Independent observations.

Think time, s	User class I		User class II		User class III	
	RT, s	Ucpu, %	RT, s	Ucpu, %	RT, s	Ucpu, %
10	0.79	31.8	2.72	33.1	2.7	39.8
7.5	0.83	41.0	3.14	39.9	2.02	37.0
5	0.88	50.7	2.72	53.5	2.00	67.0
2.5	1.29	76.1	2.03	77.8	2.83	64.8
1	1.43	80.1	2.91	80.99	3.71	92.0

4.2 Prediction Accuracy.

Prediction error for each {User think time, user class} pair is defined as a relative standard error

$$\delta = \left|\frac{(Y-\bar{Y})}{Y}\right| * 100\%$$

where Y is an observation and $\bar{Y}$ is predicted value. Accuracy of prediction for the entire model is estimated as a standard deviation of the sum of squares of prediction errors

$$\sigma = \sqrt{\frac{\sum \delta^2}{N-P}}$$

where N = 15 (3 user classes, 5 think time values (1, 2.5, 5, 7.5 and 10 s)) and P = 4 (intercept and 3 independent variables).

Additionally, these observations were compared to the prediction based on the out-of-the-box Mean Value Analysis (MVA) algorithm for queueing network models, implemented in the Java Modelling Tool [2].

Total prediction error and estimation bias (average of all differences between observed and predicted values, not their absolute values) for response time and CPU utilisation for each

model type, full factorial design (FF) for 3 factors and MVA prediction are summarised in the Table 9:

Table 9: Total prediction error and bias for various model types.

		Total prediction error σ, %		Bias, %	
		RT	CPU	RT	CPU
Response Surface models	Linear	6.51	4.3	-3.62	-0.75
	Interactions	6.32	4.09	-2.6	-0.65
	Pure quadratic	5.11	4.93	-2.02	-0.79
	Quadratic	5.42	4.09	-1.0	-0.69
	Full polynomial	5.12	4.06	-1.97	-0.96
FF		6.896	3.987	-4.96	-0.32
MVA		40.0	11.4	-234.6	7.29

From the Table 9 it may be seen that prediction error σ for the response time is a bit higher in the case of linear models ('linear', 'interactions' and full factorial design), which is to be expected since relationship between number of users and response time is not linear. As for the CPU utilization, all DoE models showed error 4-5%. This may be explained by the fact that within most of the design space CPU utilization increases linearly with increase in the number of users. However, prediction by MVA algorithm produced the error of 40% for the response time.

In order to investigate this phenomenon, we looked into independent observations and predicted values obtained from both DoE models and MVA algorithm. From the Table 8 and Figure 4 it may be seen that for the response time both observed and predicted response times do not follow classical trend of monotonous increase with decrease in user think time [3]. The comparison between independent observations, DoE RS 'full polynomial' model and MVA algorithm predictions, along with prediction errors (on the example for the user class III) are presented in the Table 10. Comparison of results in Table 10 revealed that both independent observations and values, predicted by RS model, follow the same trend. It indicates that there is some persistent (i.e. constantly present) behaviour, which RS model, having no knowledge of the system under test, however, is able to capture based only on the application inputs and outputs. MVA algorithm also captures this trend, however, it drastically overestimates response time values.

Table 10: Trend for response time (s) in predicted values and observations.

	User think time, s				
	10	7.5	5	2.5	1
Observed	2.7	2.02	1.9	2.8	3.7
RS model	2.66	2.33	2.39	2.84	3.67
Error,%	**1.1**	**-15.3**	**-25.8**	**-1.4**	**0.8**
MVA	33	24.9	25.5	43.8	58.4
Error, %	**-1122**	**-1124**	**-1216**	**-1464**	**-1478**

All RS models demonstrate negative bias, which means that overall prediction tends to overestimate both response time and CPU utilisation, except MVA algorithm, which underestimates CPU utilisation.

5. CONCLUSIONS AND FUTURE WORK

This study highlighted the importance of software performance modelling and prediction, identified existing gap in the knowledge and proposed a new performance modelling approach, based on the Design of Experiments technique.

The results demonstrate that proposed method produces good prediction of an application performance while treating it as a black box, even in the presence of an anomalous behaviour. Additionally, it showed much better prediction capabilities compared to the out-of-the box Queuing Network model. This allows to suggest that proposed method may be used for the performance modelling and prediction on the application development stage, where models based on measurements may be a better alternative as they provide a good trade-off between efforts required for model specification and accuracy of estimation and prediction.

Considering the study outcomes, some of the directions for further work may be investigation of the approach predictive capabilities in multiclass models and as a tool for anomaly detection.

6. ACKNOWLEDGMENTS

This work was supported by the funding from the European Union's Horizon 2020 research and innovation programme [grant agreement No. 644869]. Authors also would like to thank Dr. G. Casale and Dr. J.F. Perez-Bernal from Imperial College London for their support and invaluable comments.

7. REFERENCES

[1] NIST/SEMATECH e-Handbook of Statistical Methods http://www.itl.nist.gov/div898/handbook/

[2] Java Modelling Tools. http://jmt.sourceforge.net/

[3] Lazowska E et. al. 'Quantitative system performance'. Available online from: http://homes.cs.washington.edu/~lazowska/qsp/

[4] Z. Li, L. O'Brien, H. Zhang, and R. Cai. A factor framework for experimental design for performance evaluation of commercial cloud services. In Cloud Computing Technology and Science, 2012 IEEE 4th International Conference on, pages 169, 176.

[5] MDload load generation simulator. https://github.com/imperial-modaclouds?query=modaclouds-mdload

[6] OFBiz web-based 3 tier e-commerce application. http://ofbiz.apache.org/

[7] J. Rai. Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling. Wiley Computer Publishing, John Wiley & Sons, Inc. ISBN: 0471503363 Pub Date: 05/01/91

[8] Software Engineering Institute - Blog https://blog.sei.cmu.edu/post.cfm/continuous-integration-in-devops

[9] Spinner S., Casale G., Zhu X., and Kounev S. LibReDE: A Library for Resource Demand Estimation (Demonstration Paper). In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*, Dublin, Ireland, March 22-26, 2014. ACM. March 2014

[10] D. Westermann, R. Krebs, and J. Happe. Efficient experiment selection in automated software performance evaluations. In Computer Performance Engineering, pages 325-339. Springer, 2011.