A Tool for Online Experiment-Driven Adaptation

Ilias Gerostathopoulos
Faculty of Informatics
Technical University Munich
Munich, Germany
ilias.gerostathopoulos@tum.de

Ali Naci Uysal
Faculty of Informatics
Technical University Munich
Munich, Germany
ali.uysal@tum.de

Christian Prehofer
Faculty of Informatics
Technical University Munich
Munich, Germany
christian.prehofer@tum.de

Tomas Bures
Faculty of Mathematics and Physics
Charles Universtity in Prague
Prague, Czech Republic
bures@d3s.mff.cuni.cz

***Abstract*—In this paper, we present Online Experiment-Driven Adaptation (OEDA), a tool for performing end-to-end optimization of a target system abstracted as a black-box by combining statistical and optimization methods and providing statistical guarantees along the optimization process. We present the requirements and architecture of OEDA and describe its built-in optimization process that chains together factorial design, Bayesian optimization, and t-test. OEDA allows the user to create reusable abstractions of systems-to-be-optimized and specify, run and observe the execution of end-to-end experiments. For instance, we support data exchange with common tools like Kafka, MQTT and HTTP. We show the benefits of OEDA in a web server application example. OEDA can be a useful vehicle for research in the area of automated experimentation, an emerging challenge where systems are capable of performing experiments (akin to A/B testing) to themselves in order to self-optimize.**

Keywords—automated experimentation, data-driven runtime decision-making, statistical guarantees, tool

I. Introduction

Modern software-intensive systems become more dynamic, open-ended, user-facing and cyber-physical. It is thus difficult to fully model their internal behavior and their operational environments in advance in order to effectively adapt and optimize them at runtime. Consider the case of a ride-hailing service: the demand for rides shall match its supply despite unpredictable traffic situations. Such a service may be optimized for profit by offering each user an appropriate price for a ride. It may also be optimized for passengers' convenience by reducing their waiting time or their time-to-destination. The optimization process may, e.g., involve tweaking the parameters of the algorithm that matches drivers to passengers. How to choose the values that optimize the metric of interest taking into account that different values will be better in different situations (e.g. low demand, traffic congestion, holiday season)?

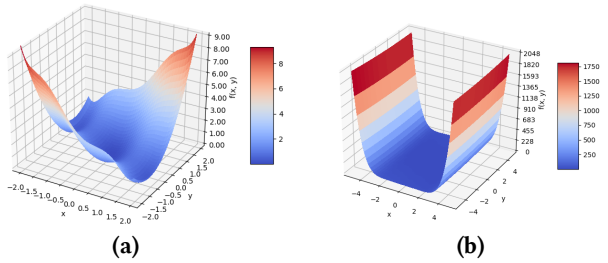
Instead of trying to anticipate all potential situations and model them a priori, an important approach in optimizing such a system is to deploy it and to experiment with it in production. Experimentation in production environments (also referred to as *online experimentation*) is exercised by many Internet companies such as Google [1], Microsoft [2], Facebook [3], LinkedIn [4] and Uber [5]. A popular method is A/B testing, where a version of a product or feature is compared against another version by measuring metrics related to reactions of the real users (click-through rate, number of transactions completed, etc.). While several sophisticated tools and techniques have been developed at these companies, they all share the following properties and limitations: (i) they focus on web-facing applications; (ii) the specification and fine-tuning of experiments require both expertise and manual effort; (iii) they focus on automating the execution of experiments, e.g. by enabling one to run multiple experiments in parallel [1] or automatically ramping-up experiments [6], but do not attempt to automate the execution of *a sequence of experiments*.

We believe that online experimentation can be taken to the next level in which systems (instead of humans) control and run experiments on themselves in order to self-optimize—*automated experimentation* [7], [8]. This calls for techniques and tools that overcome the above-mentioned limitations and, in particular, allow for chaining experiments and data-driven decisions together in an automated fashion.

Such an automated approach to online experimentation and data-driven decision-making comes with a number of challenges. A central challenge is how to automatically generate new experiments based on the already performed ones and, in particular, how do this in a way that is both safe and efficient. Consider, e.g., automatically chaining A/B experiments: the process needs to understand the guarantees provided by the statistical method (e.g. t-test) used in evaluating each A/B experiment and guide the selection of a promising next experiment by inspecting, e.g. the effect sizes in the different comparisons. Such decision-making needs to rely on solid statistical grounds. Failing to do so may lead to wrong or irrelevant decisions that can slow down the optimization process or even jeopardize the system under experimentation.

As a first step towards supporting automated experimentation and data-driven decision-making, we have developed a prototype tool called Online Experiment-Driven Adaptation (OEDA). OEDA provides an end-to-end solution for optimizing a target system via a sequence of experiments, while making explicit the statistical guarantees involved in data-driven runtime decision-making. In the current stage, OEDA supports the research idea (originally reported in [9]) of chaining together three different methods used for generating and evaluating experiments: factorial design, Bayesian optimization, and t-test. Following the experimentation-as-a-service (EaaS) model [10], OEDA can be easily integrated with target systems via different input and output channels (Kafka or MQTT topics, HTTP end-points). In this paper, we detail on the requirements, design and implementation, and possible extensions of OEDA.

The tool is available for download as open-source software at https://github.com/alinaciuysal/OEDA, where also a Getting Started guide and an explanation video are available.



(a) (b)

Figure 1: Plots of Three-Hump Camel Function with x and y ranging in (a) [-2,2] (b) [-5,5].

II. Running Example

To illustrate the use of OEDA, we will use a simple HTTP-based server application as a running example throughout the paper. The application is computing and returning a bivariate function *f(x,y)* every time it receives a GET request. Via POST requests, the values of parameters *x* and *y* can be set. The optimization goal in this case is to minimize the unknown but externally observable function *f*—which corresponds to a single-objective, two-variate mathematical optimization problem.

In particular, we have used as *f* the Three-Hump Camel Function (THCF), an existing test function for evaluating optimization algorithms[1]:

$$f(x,y) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2$$

As can also be seen in its plot (Figure 1), in the [-5,5] range for *x* and *y,* THCF has its minimum (of value 0) at x=y=0.

In order to simulate a noisy system, we have introduced random noise in the calculation of THCF by multiplying the result of the actual function with a random number from [0,1) before returning it.

It is important to note that we use this running example for illustration purposes; for evaluating OEDA, we have used it in optimizing a complex self-adaptation exemplar featuring a crowd-sourced traffic navigation system—CrowdNav [11]—with seven numeric tunable parameters[2] [9].

III. OEDA Optimization Process

The online optimization process built in into OEDA so far can be used to optimize any system which can be configured at runtime by setting its tunable (input) parameters. A valuation of all parameters is called a *configuration*. At its current version, the optimization process works with numeric parameters that can take values within a range.

As depicted in Figure 2, three different methods are used in the process. First, a factorial design is performed; its results are analyzed using factorial Analysis of Variance (ANOVA) [12]. The goal here is to profile the target system by applying only pre-selected values of its input parameters and determine which of its parameters or combination of parameters have a statistically significant effect on the output of the system.

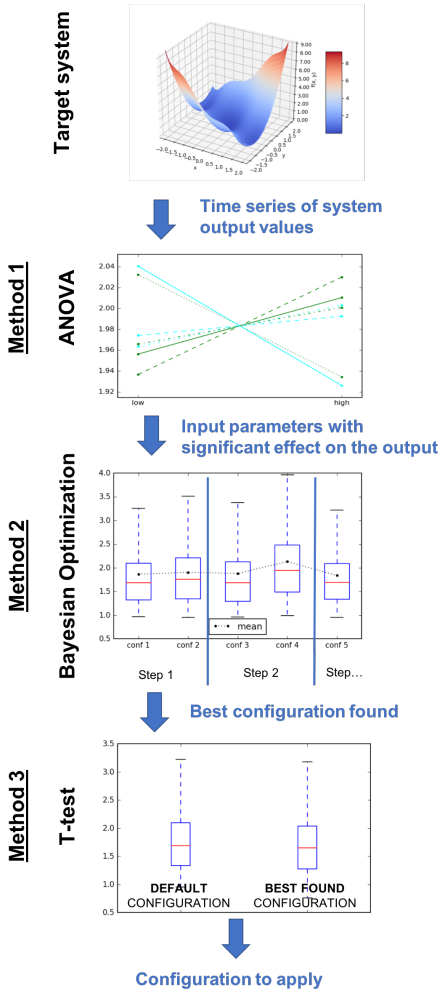


Figure 2: OEDA optimization process.

Second, Bayesian optimization (BO) [13] is performed considering the output of ANOVA. In particular, for each parameter or combination of parameters outputted by ANOVA, one run of BO is performed, with the goal of optimizing the particular parameter or combination of parameters. To deal with noise, the integrated BO uses Gaussian process as regressor model. Finally, the third method used is t-test. T-test in OEDA answers the question of whether the best configuration found by the BO is statistically significantly better than the default configuration. If this holds true, then the optimization process ends by suggesting to apply the best-found configuration to the system, otherwise the suggestion is to keep the default. For details on the above process and the individual methods, we refer the interested reader to [9].

IV. OEDA Architecture

The main functional requirements of OEDA were (i) to allow the user to specify a target system, (ii) to allow the user to specify an OEDA experiment by customizing the optimization process outlined in Section 3, (iii) to allow the user to see the

[1] https://en.wikipedia.org/wiki/Test_functions_for_optimization

[2] https://github.com/alinaciuysal/OEDA

results of both a running and a completed OEDA experiment. Usability and extensibility were the main non-functional goals.

OEDA is developed as a client-server application with a web-based graphical user interface. In particular, as depicted in Figure 3, the Client (written in Angular 4) serves both for user input and for presenting the results of experiments (including plotting data where necessary). The Server (written in Python using Flask[3]) orchestrates the experiment execution via its Workflow Executor. This is responsible for communicating to the Target System via the Change Provider, which issues changes to the Target System for applying configurations, and the Data Provider, which listens to values of the Target System output(s). The Data and Change Providers are commonly referred to as Middleware in Figure 3. Workflow Executor is also responsible for consulting the Optimization component for the next configuration to run. Optimization is responsible for running the Bayesian optimization with Gaussian Processes (BOGP) algorithm. We have so far integrated a Python (skopt[4]) and an R implementation [14] of BOGP, which offer different sets of acquisition functions (an important parameter of BOGP). Finally, all the data and the results of the analysis are saved in the Database—we are using Elasticsearch.

V. Specifying a Target System

To perform an experiment, the user needs to first specify a target system. A target system in OEDA is always viewed as a black box; its specification consists of identifying its inputs and its output, and the way to connect to the target system.

Concretely, to specify a target system in OEDA, the user needs to specify (via the web-interfaces of the OEDA tool):

- a single Change Provider;
- one or more Data Providers;
- the settling time in terms of data measurements that are to be ignored after applying a configuration in order to let the system stabilize;
- the name and scale (nominal, ordinal, metric, or ratio) of an Output, along with its optimization direction (minimize or maximize);
- the name, scale, range, and default value of one or more Inputs.

A Data Provider or Change Provider can be of one of the following three types: Kafka [15], MQTT [16] or HTTP. Kafka is a distributed messaging system, where producers publish to topics (grouping messages of a particular type) and consumers subscribe to topics and pull messages from Kafka brokers. MQTT is another publish-subscribe system which allows for publishers and subscribers to communicate asynchronously via topics. Finally, an HTTP Data provider is essentially the specification of a GET request used to pull data from a web server, whereas an HTTP Change Provider is a POST request used to send a configuration to a web server.

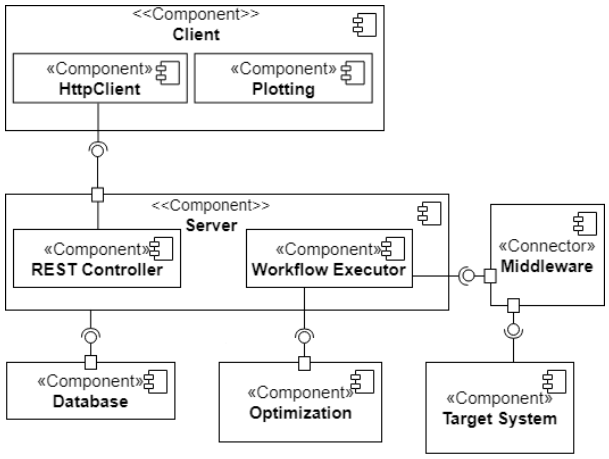


Figure 3: OEDA Architecture.

In terms of tool workflow, it is important to note that once a target system specification is specified (with the help of the dedicated page) and saved, it can be seen in the list of available target systems and used to perform different—but non-overlapping—experiments (Section 6).

A. *Illustration on the Running Example*

The target system of our running example is specified in the following way.

- Change Provider is of type HTTP, with URL pointing to the port the web server listens to (e.g. localhost:3003).
- Data Provider is of type HTTP, with URL similar to the Change Provider.
- Settling time is set to 0, since there is no hysteresis in the system (configuration changes have immediate effect).
- Output is a variable called result of scale metric with the optimization direction of minimize.
- Inputs are two variables: (i) a variable named x of scale metric and range [-5,5] and (ii) a variable named y of scale metric and range [-5,5]; default values for these inputs were x=-4 and y=-5 (arbitrarily chosen).

VI. Performing End-to-end Optimization

End-to-end optimization is performed by specifying and running an OEDA experiment. The specification of an experiment starts by selecting a target system and can be broken down into the following four steps.

Step 0: Specify the summary statistic for the output values.

The optimization process needs to know how to compare system configurations based on the effect they have on the output of the system. If multiple output values are collected for each configuration, which is a common case in real-life systems

[3] http://flask.pocoo.org/

[4] https://scikit-optimize.github.io/notebooks/bayesian-optimization.html

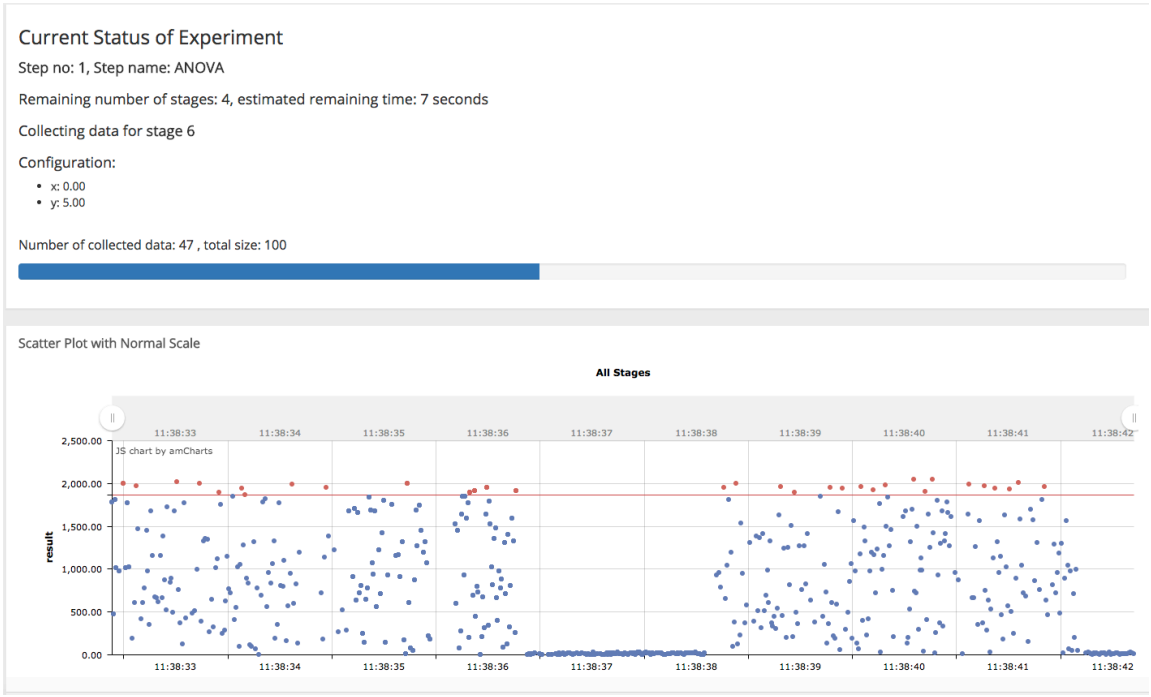


Figure 4: Observing status and incoming data in an OEDA experiment.

that are noisy, we need to specify how to summarize values for the comparison. The chosen summary statistic is in particular used to calculate the target system's response to a configuration applied by the Bayesian optimizer in step 2. Available summary statistics include average, median, 95th percentile, and max.

Step 1: Customize factorial design and factorial ANOVA.

At this step, the user needs to specify the actual factorial design to perform by selecting which input parameters to include (out of the ones retrieved from the specification of the target system) and which values to consider for each input parameter. This selection will also determine the factorial design to be performed. In addition, the user needs to specify:

P1.1 The number of samples to collect for each configuration;

P1.2 The significance level (alpha) for determining significant interactions in ANOVA (default value is 0.05);

P1.3 The max number of significant interactions that should be considered in the next step (Bayesian optimization).

Step 2: Customize Bayesian optimization runs.

At this step, the user needs to select which Bayesian optimizer implementation to use (Python or R) along with the acquisition function to use (option for expert users only, we provide meaningful defaults). The user also needs to specify:

P2.1 The number of samples to collect for each configuration tried by the Bayesian optimizer (which might be e.g. larger than the corresponding value in factorial design);

P2.2 The number of iterations (budget) of the Bayesian optimization which will be equally split into the different runs of the optimizer.

Step 3: Customize T-test.

At the final step, the user specifies:

P2.1 The number of samples to collect for the two configurations to be run;

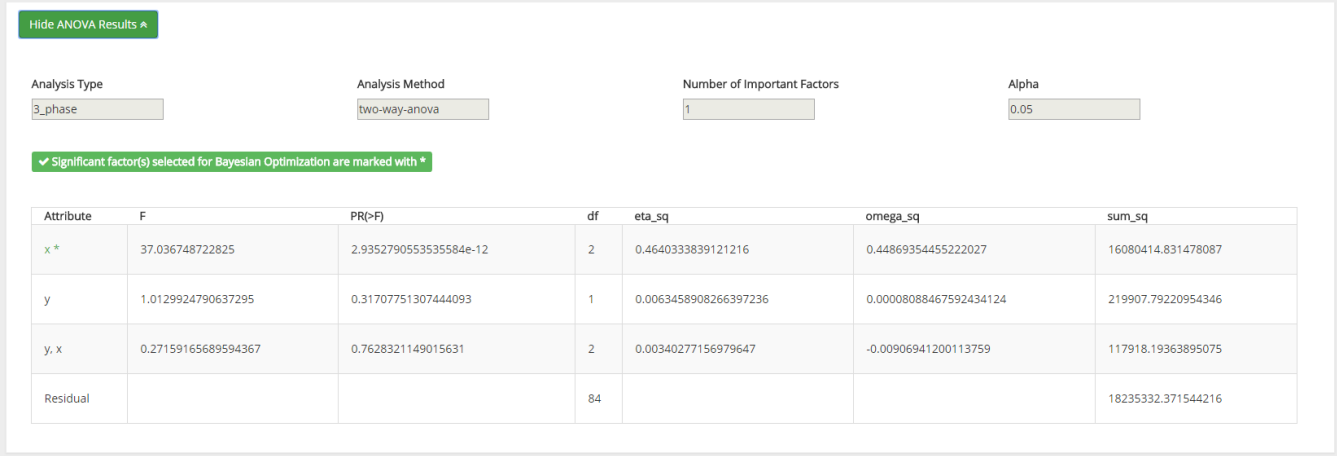
P2.2 The significance level (default is 0.05) for determining statistically significant differences between the datasets corresponding to the two configurations of the t-test;

P2.3 The minimum effect size (default is 0.7), measured by the Cohen's d coefficient, of the measured difference.

Upon starting an experiment, the user can observe its status via guiding messages and a scatter plot (Figure 4), along with a histogram and a Q-Q plot—all being updated in real time.

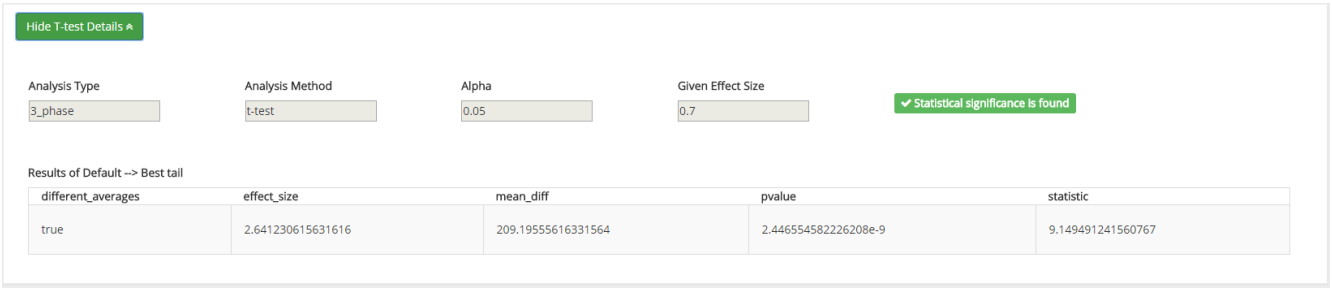
A. Illustration on the Running Example

We describe here an example of a concrete optimization process applied on the running example and report on the obtained results. First, the user starts the creation of a new experiment by selecting the "Black-box function" target system on the web-interface. For Step 0, the user selects average as the summary statistic of the result output. For Step 1, the user selects both x and y to be included in the factorial design—each parameter takes three values: -5, 0, 5. The user sets the number of samples to 100, the significance level to 0.05 and the max number of significant interactions to 1. For Step 2, the user picks the Python implementation with the default acquisition function and sets the number of samples to 200 and the number of



Attribute	F	PR(>F)	df	eta_sq	omega_sq	sum_sq
x *	37.036748722825	2.9352790553535584e-12	2	0.4640333839121216	0.44869354455222027	16080414.831478087
y	1.0129924790637295	0.31707751307444093	1	0.0063458908266397236	0.00008088467592434124	219907.79220954346
y, x	0.27159165689594367	0.7628321149015631	2	0.00340277156979647	-0.00906941200113759	117918.19363895075
Residual			84			18235332.371544216

Figure 5: Presentation of ANOVA results (step 1) in OEDA.



different_averages	effect_size	mean_diff	pvalue	statistic
true	2.641230615631616	209.19555616331564	2.446554582226208e-9	9.149491241560767

Figure 6: Presentation of t-test results (step 3) in OEDA.

iterations to 10. Finally, for step 3, the user specifies 500 as the number of samples for the t-test and sets the significance level to 0.05 and the minimum effect size to 0.7. The user starts the experiment.

Upon completion of the experiment, the user observes the tables and plots where the following results are reported:

- ANOVA identified parameter x as having a significant effect on the result of the function (Figure 5);
- the Bayesian optimizer performed 10 iterations selecting different values for x; the best value found was -0.68 (which is close to the optimal value of 0);
- the t-test identified a significant difference between the best configuration outputted by the optimizer (x=-0.68, y=-5) and the default configuration (x=-4, y=-5) with a p-value close to 0 and effect size ~2.6, which was larger than the 0.7 threshold (Figure 6).

VII. Discussion

In this section, we compare OEDA to related work and detail on the applicability and the possible extensions to the tool.

A. Comparison to Related Tools and Techniques

OEDA follows the idea of externalizing optimization and experimentation as a service that is managed separately from the system to be optimized (target system) [10]. The same idea is behind Vizier, Google's internal service for black-box optimization [17], and Optimizely.com which focuses on web optimization via A/B testing tools [18]. Contrary to them, though, OEDA offers the possibility to specify and run a chain of experiments (which in OEDA is called an *experiment* with several *steps*).

A work that comes close to OEDA is the PlanOut language proposed by Facebook [3], [19]. PlanOut allows for specifying experiments ranging from simple A/B tests to factorial designs or even designs with conditional logic (fractional factorial). Although PlanOut is valuable in separating the experimental design from the application code, it neither allows for specifying which analyses methods to use (e.g. t-test, ANOVA) nor allows for chaining experiments together by interpreting their results as OEDA does.

B. Applicability and Lessons Learned

OEDA can be currently used to optimize any target system that can be abstracted as a black box with multiple inputs of metric scale and a single, metric-scaled output. To plug a system to OEDA, the system needs to provide either HTTP endpoints or Kafka or MQTT topics which can be used for applying a configuration to the system and listening to its output.

When using OEDA to minimize car trip durations in CrowdNav, a traffic navigation system abstracted as a black box with seven input parameters (controlling the car navigation), we were able to detect the parameters with significant effect on the output (via ANOVA), restrict the optimization space and reduce the number of required Bayesian optimization steps [9].

C. Extensions to OEDA

One of the next milestones to reach is to allow for performing multi-criteria optimization within OEDA. While this extension may be straightforward in the case of Bayesian optimization, where the summary statistic can be set as a linear combination of two or more output values, it is not yet clear how the ANOVA and the t-test steps can be extended for the multi-criteria case.

Another extension is to allow for specifying a threshold on the permissible cost of performing an experiment or an experiment step, along with dedicated support for observing a *cost metric* of a target system. In the case of optimizing a ride-hailing service (the example from the Section I), e.g. the optimization metric may be the time-to-destination while the cost metric can be the number of received complaints. This will allow for automatically stopping the evaluation of configurations that incur high experimentation cost [20].

We would like to provide different experimentation chains in OEDA. For instance, we could have a separate chain for optimizing discrete parameters, in contrast to the current chain that can be used only with continuous parameters. Even better, we envision to allow the users of OEDA to specify their own experimentation steps and connect them together to create new experimentation chains. This would allow, e.g. for using a different method than ANOVA for profiling a target system (step 1 in the built-in chain).

Finally, to support automated experimentation we would like to have OEDA running side-by-side a target system and autonomously start different experiments whenever e.g. the target system resides in a situation it has no optimal configuration for. OEDA would then be used more as a tool for observing and understanding the data-driven decisions taken via continuous, automated experimentation rather than for manually triggering specific experiments to be run on an external system.

VIII. Conclusion

In this paper, we presented OEDA, a tool for end-to-end optimization of systems abstracted as black boxes that combines different statistical and optimization methods and provides statistical guarantees. We have used our tool to optimize the running example, but also a larger, more complex example of a traffic navigation service [9]. OEDA is part of our ongoing research in supporting automated experimentation, whereby systems (instead of humans) control and run experiments on themselves in order to self-optimize. Automated experimentation will need to combine the execution of experiments and the analysis of their results via rigorous statistical methods in an automated process with minimal human input and supervision—it thus presents a number of important and interesting challenges for future research.

Acknowledgements

This work has been partly funded by the Bayerisches Staatsministerium für Wirtschaft und Medien, Energie und Technologie as part of the TUM Living Lab Connected Mobility Project and partly sponsored by the German Ministry of Education and Research (BMBF) under grant no 01Is16043A. The work has been partially supported by project no. LTE117003 (ESTABLISH) from the INTER-EUREKA LTE117 programme by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] D. Tang, A. Agarwal, D. O'Brien, and M. Meyer, "Overlapping experiment infrastructure: More, better, faster experimentation," in *Proc. of SigKDD 2010*, ACM, 2010, pp. 17–26.

[2] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The evolution of continuous experimentation in software product development: from data to a data-driven organization at scale," in *Proc. of ICSE 2017*, IEEE, 2017, pp. 770–780.

[3] E. Bakshy, D. Eckles, and M. S. Bernstein, "Designing and Deploying Online Field Experiments," in *Proceedings of the 23rd International Conference on World Wide Web*, 2014, pp. 283–292.

[4] Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin, "From Infrastructure to Culture: A/B Testing Challenges in Large Scale Social Networks," in *Proc. of KDD'15*, 2015, pp. 2227–2236.

[5] "Uber Experimentation Platform," 18-Jun-2018. [Online]. Available: https://eng.uber.com/tag/experimentation/.

[6] Y. Xu, W. Duan, and S. Huang, "SQR: Balancing Speed, Quality and Risk in Online Experiments," p. 9.

[7] D. I. Mattos, J. Bosch, and H. H. Olsson, "More for Less: Automated Experimentation in Software-Intensive Systems," in *Product-Focused Software Process Improvement*, Springer, Cham, 2017, pp. 146–161.

[8] D. I. Mattos, J. Bosch, and H. H. Olsson, "Your System Gets Better Every Day You Use It: Towards Automated Continuous Experimentation," in *SEAA 2017*, IEEE, 2017, pp. 256–265.

[9] I. Gerostathopoulos, C. Prehofer, and T. Bures, "Adapting a System with Noisy Outputs with Statistical Guarantees," in *Proc. of SEAMS 2018, to appear.*, 2018.

[10] D. I. Mattos, J. Bosch, and H. H. Olsson, "Challenges and Strategies for Undertaking Continuous Experimentation to Embedded Systems: Industry and Research Perspectives," in *Agile Processes in Software Engineering and Extreme Programming*, 2018, pp. 277–292.

[11] S. Schmid, I. Gerostathopoulos, C. Prehofer, and T. Bures, "Self-Adaptation Based on Big Data Analytics: A Model Problem and Tool," in *Proc. of SEAMS 2017*, IEEE, 2017, pp. 102–108.

[12] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 4th ed. Chapman & Hall/CRC, 2007.

[13] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.

[14] B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang, "mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions," *arXiv preprint arXiv:1703.03373*, 2017.

[15] J. Kreps, N. Narkhede, J. Rao, and others, "Kafka: A distributed messaging system for log processing," in *Proc. NetDB'11*, 2011, pp. 1–7.

[16] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi, "Internet of Things: Survey and open issues of MQTT protocol," in *2017 International Conference on Engineering MIS (ICEMIS)*, 2017, pp. 1–6.

[17] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google Vizier: A Service for Black-Box Optimization," 2017, pp. 1487–1495.

[18] Optimizely, "Optimizely: The World's Leading Experimentation Platform," 03-Jun-2018. [Online]. Available: https://www.optimizely.com/.

[19] Facebook, "PlanOut: A framework for online field experiments," 03-Jul-2018. [Online]. Available: https://facebook.github.io/planout/.

[20] I. Gerostathopoulos, C. Prehofer, L. Bulej, T. Bures, V. Horky, and P. Tuma, "Cost-Aware Stage-Based Experimentation: Challenges and Emerging Results," in *Proc. of ICSA 2018, to appear.*