1

Ilias Gerostathopoulos, Technical University Munich
Alexander auf der Straße, Universität Duisburg-Essen

Online Experiment-Driven Learning and Adaptation

This chapter presents an approach for the online optimization of collaborative embedded systems (CES) and collaborative system groups (CSG). Such systems need to adapt and optimize their behavior at runtime to increase their utilities and respond to runtime situations. We propose to model such systems as black boxes of their essential input parameters and outputs, and efficiently search in the space of input parameters for values that optimize (maximize or minimize) the system's outputs. Our optimization approach consists of three phases and combines online (Bayesian) optimization with statistical guarantees stemming from the use of statistical methods such as factorial ANOVA, binomial testing, and t-tests in different phases. We have applied our approach in a smart cars testbed with the goal of optimizing the routing of cars via tuning the configuration of their parametric router at runtime.

1.1 Introduction

The behavior of CES and CSG is difficult to completely model a priori

Collaborative embedded systems (CES) and collaborative system groups (CSG) are often large systems with complex behavior. Complexity stems mainly from the interaction of the different components or sub-systems (consider e.g. the case of several robots collaborating in pushing a door open or passing through a narrow passage). As a result, the behavior of CES is difficult to completely model a priori. At the same time, CES need to be continuously adapted and optimized to new runtime contexts (e.g. in the collaborating robots' example, consider the case of an extra obstacle that makes the door harder to open).

Approach for online learning and adaptation of CES and CSG abstracted as black-box models

In this chapter, we present an approach for online learning and adaptation that can be applied in CES and CSG (but also other systems) which have (i) complex behavior that is unrealistic to completely model a priori, (ii) noisy outputs, (iii) high cost of bad adaptation decisions. We assume that the CES-to-be-adapted is abstracted as a black-box model of the essential input and output parameters. Input parameters (knobs) can be set at runtime to change the behavior of the CES. Output parameters are monitored at runtime to assess whether the CES satisfies its goals. Noisy outputs refer to outputs whose values exhibit high variance, and thus may need to be monitored over long time windows. The cost of an adaptation decision (e.g. setting a new value to one of the knobs) refers to the negative impact of the adaptation decision on the CES.

Finding values of input parameters that optimize the outputs

Given the above assumptions, we focus on finding the values of the input parameters of a CES that optimize (maximize or minimize) its outputs. Our approach performs such optimization online, i.e. while the system is running, and in several phases [Gerostathopoulos et al. 2018]. In doing so, it explores and exemplifies (i) how to build system models out of observations of noisy system outputs; (ii) how to (re-)use these models to optimize the system at runtime, even in the face of newly encountered situations; and (iii) how to incorporate the notion of cost of adaptation decisions in the above processes. Compared to related approaches, it focuses on providing statistical guarantees (in the form of confidence intervals and p-values) in different phases of the optimization process.

1.2 A Self-Optimization Approach for CES

A self-optimization approach for CES needs to be (i) efficient in finding an optimal or close-to-optimal configuration fast, and (ii) safe in not incurring high cost of adaptation decisions. To achieve these goals, in our approach, we use prior knowledge of the system (the K in the MAPE-K loop for self-adaptive systems [Kephart and Chess 2003]) in order to guide the exploration of promising configurations. We also measure the cost of adaptation decisions in the optimization and stop the evaluation of bad configurations prematurely to avoid incurring high cost.

Self-optimization by finding the best configuration

Formally, the self-optimization problem we are considering consists of finding the minimum of a response or output function $f: X \rightarrow R$, which takes n input parameters $X_1, X_2, \ldots, X_n$, which range in domains $Dom(X_1), Dom(X_2), \ldots, Dom(X_n)$ respectively. X is the configuration space and corresponds to the Cartesian product of all the parameters domains $Dom(X_1) \times Dom(X_2) \times \ldots \times Dom(X_n)$. A configuration C assigns a value to each of the input parameters.

Based on the above definitions, our approach for self-optimization of CES relies on performing a series of so-called online experiments. An experiment changes the value of one or more input parameters and collects values of the outputs. This allows for assessing the impact of the input parameter change on the outputs. The experiment-driven approach consists of the following three phases, depicted also in Figure 1-1 (where the CES is depicted on the upper right corner):

- Phase #1: Generation of system model
- Phase #2: Runtime optimization with cost handling
- Phase #3: Comparison with baseline configuration

These phases run consecutively; in each phase one or more experiments are performed. An optimization round consisting of the three phases may be initiated via a human (e.g. an operator) or via the system itself, if it is able to identify runtime situations where its behavior can be optimized. At the end of the optimization round, the system has learned an optimal or close-to-optimal configuration and decides (as part of phase #3) to use it instead of its current configuration or not.

In the rest of the section, the three phases are described.

Using factorial analysis of variance for building knowledge models

"Generation of system model" phase deals with building and maintaining the knowledge needed for self-optimization. Here we use factorial analysis of variance (ANOVA) to process incoming raw data and automatically create a statistically relevant model that is used in the subsequent phases. This model describes the effect of

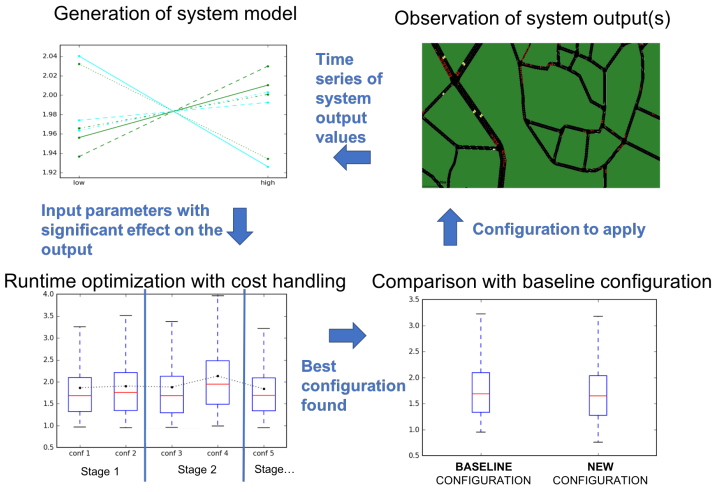

Figure 1-1. Overview of online experiment-driven learning and adaptation approach.

changing a single input parameter on the output, while ignoring the effect of any other parameters. It also describes the effects of changing multiple input parameters together on the output. This phase is run both prior to deploying the system using a simulator (to bootstrap the knowledge) and while the system is deployed in production using runtime monitoring (to gradually collect more accurate knowledge of the system in the real settings).

Concretely, first, the designer needs to discretize the domain of each input parameter in two or more values—an offline task. When the phase starts, the system derives all the possible configurations given the parameter discretization (e.g. for three input parameters with two values each, it will derive 8 possible configurations capturing all possible combinations). This corresponds to a full factorial design in experimental design terminology [Ghosh and Rao 1996]. Third, for each configuration, an online experiment is performed and output values are collected. Once all experiments are performed, between-samples factorial ANOVA is used to analyze the output datasets corresponding to the different configurations. The output of this phase is a list of input parameters ordered by decreasing effects (and corresponding significance levels) on the output.

Using Bayesian optimization for finding optimal configuration

"Runtime optimization with cost handling" phase evaluates configurations via online experiments in a sequential way to find a configuration in which the system performs the best, i.e. the output function is maximized or minimized. Instead of pre-designing the

experiments to run as in phase #1, we use an optimizer that selects the next configuration to run based on the result of the previous experiment. In particular, the optimizer we have used so far employs Bayesian optimization with Gaussian processes [Shahriari et al. 2016]. The optimizer takes as input the output of phase #1, i.e. a list of input parameters. For each parameter in the list, it selects a value from its domain (its original domain, not its discretized one used in phase #1) and performs and online experiment to assess the impact of the corresponding configuration to the system output. Based on the result of the online experiment, the optimizer selects another input parameter value, performs another online experiment, and so on. Before the start of the optimization process, the design sets the number of online experiments (iterations of the optimizer) that will be run in phase #2. The outcome of this phase is the best configuration found by the optimizer.

We assume that configurations are rolled out incrementally in the system. If there is evidence that a configuration incurs high cost, its application stops and the optimizer moves on to evaluate the next configuration. So far, we assume that cost is measured in terms of ratio of bad events, e.g. complaints. Under this assumption, we use binomial testing to determine (with statistical significance) whether a configuration is not worth exploring anymore because of cost overstepping a given threshold. A binomial test is a statistical procedure that tests whether, in a single sample representing an underlying population of two categories, the proportion of observations in one of the two categories is equal to a specific value. In our case, a binomial test evaluates the hypothesis that the predicted proportion of issued "bad events" is above a specific value—our "bad events" maximum threshold.

Using statistical testing to compare optimal with default configuration

"Comparison with baseline configuration" makes sure that a new configuration determined in the second phase is rolled out only when it is statistically significantly better than the existing configuration (baseline configuration). In order for the new configuration to replace the baseline, it has to be checked that (i) it indeed brings a benefit to the system (at a certain statistical significance level), (ii) the benefit is enough to justify any disruption that may result out of applying the new configuration to the system. The last point recognizes the presence of primacy effects, which pertain to inefficiencies caused by a new configuration to the users.

Concretely, in this phase, the effect of the (optimal) configuration outputted by phase #2 is compared to the default configuration of

the system. Such default configuration is provided offline by the system designers. To perform the comparison, the two configurations are rolled out in the system and values of system output are collected. In other words, two online experiments are performed corresponding to the two configurations. Technically, the effect of the experiments is compared by means of statistical testing (so far we have used t-test) on the corresponding datasets of system outputs. This way, we can deduce whether the two configurations have a statistically significant difference (at a particular significance level *alpha*) on their effect on the system output.

1.3 Illustration on CrowdNav

Application of the approach on traffic testbed

We illustrate our approach on the CrowdNav self-adaptation testbed [Schmid et al. 2017], whose goal is to optimize the duration of car trips in a city by adapting the parameters of the routing algorithm used for navigating the cars. CrowdNav is released as an open-source project[1].

In CrowdNav, a number of cars is deployed in the city of Eichstädt with approx. 450 streets and 1200 intersections. Each car navigates from an initial (randomly allocated) position to a randomly chosen destination in the city. When a car reaches its destination, it picks another one at random and navigates to it. This process is repeated forever.

To navigate from point A to point B, a car needs to ask a router for a route (series of streets). There are two routers in CrowdNav: (i) the built-in router provided by SUMO (the simulation backend of CrowdNav) and (ii) a custom-built parametric router developed in our previous work. A certain number of cars ("regular cars") use the built-in router; the rest use the parametric router—we call these "smart cars".

The parametric router can be configured at runtime; it provides the seven configuration parameters depicted in Figure 1-2. Each parameter is an interval-scaled variable that takes real values within a range of admissible values, provided by the designers of the system. Intuitively, certain configurations of the router's parameters yield better overall system performance.

[1] https://github.com/Starofall/CrowdNav

Id	Name	Range	Description
1	route randomization	[0-0.3]	Controls the random noise introduced to avoid giving the same routes
2	exploration percentage	[0-0.3]	Controls the ratio of smart cars used as explorers[2]
3	static info weight	[1-2.5]	Controls the importance of static information (i.e. max speed, street length) on routing
4	dynamic info weight	[1-2.5]	Controls the importance of dynamic information (i.e. observed traffic) on routing
5	exploration weight	[5-20]	Controls the degree of exploration of the explorers
6	data freshness threshold	[100-700]	Threshold for considering traffic-related data as stale and disregard them
7	re-routing frequency	[10-70]	Controls how often the router should be invoked to re-route a smart car

Figure 1-2. Configurable (Input) Parameters in CrowdNav's Parametric Router.

Trip overhead is a prime example of noisy output

To measure the overall system performance, CrowdNav relies on the metric of trip overhead. A trip overhead is a ratio-scaled variable whose values are calculated by dividing the observed duration of a trip versus the theoretical duration of the trip, i.e. the hypothetical duration of the trip if there were no other cars, the smart car travelled in maximum speed and did not stop in intersections or traffic lights. Only smart cars report their trip overheads at the end of their trips (we assume that the rest of the cars act as noise in the simulation, so their effect can only be indirectly observed). Since some trips will have larger overhead than others no matter what the router configuration is, the data set of trip overheads exhibits high variance—it can be thus considered a noisy output.

Driver complaints model "bad events"

Together with the trip overhead, each smart car reports at the end of each trip a complaint value, i.e. a Boolean value indicating whether the driver is annoyed. The complaint value is generated based on the trip overhead and a random chance, so that some of the "bad trips" would generate complaints (but not all). To measure the cost of a bad configuration in CrowdNav, the metric of complaint rate is used: the ratio of issued complaints to total number of observed (trip overhead, complaint) tuples.

Finally, CrowdNav resides in different situations depending on two context parameters that can be observed, but not controlled: the number of regular (non-smart) cars and the number of smart cars. In particular, each context parameter can be in a number of predefined ranges. For example, the number of smart cars can be in one of the

following ranges or states: 0-100, 100-200, 200-300, …, 700-800, >800. All the possible situations are defined as the cartesian product of the states of all context variables. At each situation, a different configuration might be optimal. The task of self-optimization in CrowdNav then becomes one of quickly finding the optimal configuration for the situation the system resides in and applying it.

In this context, quickly finding a configuration of parameters that minimizes the trip overhead in a situation, while keeping the number of complaints in check, entails understanding the effect a configuration has on both the trip overhead (the output we want to optimize for) and the complaint rate (the "bad events" metric).

Generalizing from this scenario, the problem to solve is: "Given a set of input system parameters X, an output system parameter O with values exhibiting high variance, an environment situation S, and a cost parameter C, find the values of each parameter in X that optimize O in S without exceeding C, in the least number of tries."

Our approach focuses the optimization on the important input parameters

We have evaluated the applicability of our experiment-driven self-optimization method on CrowdNav. Compared to performing optimization with all the input parameters (essentially skipping phase #1), our approach is able to reduce the optimization space, and consequently converge faster, by only optimizing the input parameters that have a strong effect on the output (trip overhead in the case of CrowdNav) [Gerostathopoulos et al. 2018].

1.4 Conclusions

In this chapter, we presented an approach for runtime optimization of CES. Our approach relies on the concept of online experiments which consist of applying an adaptation action (changing a configuration) of a running system and observing the effect of the change in the system output. It consists of three stages that together combine optimization with statistical guarantees that come in the form of confidence intervals and observed effect sizes. We have applied the approach on a self-adaptation testbed where the routing of cars in a city is optimized at runtime based on tuning the configuration of the cars' parametric router. Our approach can be used in any system that can be abstracted as a black-box model of the essential input and output parameters.

1.5 Literature

[Gerostathopoulos et al. 2018] I. Gerostathopoulos, C. Prehofer, T. Bures, Adapting a System with Noisy Outputs with Statistical Guarantees, in Proceedings of the 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2018). ACM, 2018, pp. 58–68.

[Kephart and Chess 2003] J. Kephart D. Chess, The Vision of Autonomic Computing, Computer, vol. 36, no. 1, pp. 41–50, 2003.

[Schmid et al. 2017] S. Schmid, I. Gerostathopoulos, C. Prehofer, T. Bures, Self-Adaptation Based on Big Data Analytics: A Model Problem and Tool, in Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2017). IEEE, 2017, pp. 102–108.

[Ghosh and Rao 1996] S. Ghosh, C. R. Rao, Eds., Handbook of Statistics 13: Design and Analysis of Experiments, 1 edition. Amsterdam: North-Holland, 1996.

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

[Shahriari et al. 2016] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, 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.