A Multivariate Characterization and Detection of Software Performance Antipatterns

Alberto Avritzer[1], Ricardo Britto[2,3], Catia Trubiani[4], Barbara Russo[5], Andrea Janes[5], Matteo Camilli[5], André van Hoorn[6], Robert Heinrich[7], Martina Rapp[8], Jörg Henß[8]

[1] eSulab Solutions, Princeton, NJ
[2] Ericsson AB, Karskrona, Sweden
[3] Blekinge Institute of Technology, Karlskrona, Sweden
[4] Gran Sasso Science Institute, Italy
[5] Free University of Bozen-Bolzano, Italy
[6] University of Stuttgart, Germany
[7] Karlsruhe Institute of Technology, Germany
[8] FZI Forschungszentrum Informatik, Germany

ABSTRACT

Context: Software Performance Antipatterns (*SPAs*) research has focused on algorithms for the characterization, detection, and solution of antipatterns. However, existing algorithms are based on the analysis of runtime behavior to detect trends on several monitored variables (e.g., response time, CPU utilization, and number of threads) using pre-defined thresholds.

Objective: In this paper, we introduce a new approach for *SPA* characterization and detection designed to support continuous integration/delivery/deployment (CI/CDD) pipelines, with the goal of addressing the lack of computationally efficient algorithms.

Method: Our approach includes *SPA* statistical characterization using a multivariate analysis approach of load testing experimental results to identify the services that have the largest impact on system scalability. More specifically, we introduce a layered decomposition approach that implements statistical analysis based on response time to characterize load testing experimental results. A distance function is used to match experimental results to *SPA*s.

Results: We have instantiated the introduced methodology by applying it to a large complex telecom system. We were able to automatically identify the top five services that are scalability choke points. In addition, we were able to automatically identify one *SPA*. We have validated the engineering aspects of our methodology and the expected benefits by means of a domain experts' survey.

Conclusion: We contribute to the state-of-the-art by introducing a novel approach to support computationally efficient *SPA* characterization and detection in large complex systems using performance testing results. We have compared the computational efficiency of the proposed approach with state-of-the-art heuristics. We have found that the approach introduced in this paper grows linearly, which is a significant improvement over existing techniques.

ICPE '21, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8194-9/21/04…$15.00
https://doi.org/10.1145/3427921.3450246

CCS CONCEPTS

• **Software and its engineering → Software performance**.

ACM Reference Format:
Alberto Avritzer[1], Ricardo Britto[2,3], Catia Trubiani[4], Barbara Russo[5], Andrea Janes[5], and Matteo Camilli[5], André van Hoorn[6], Robert Heinrich[7], Martina Rapp[8], Jörg Henß[8]. 2021. A Multivariate Characterization and Detection of Software Performance Antipatterns. In *Proceedings of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3427921.3450246

1 INTRODUCTION

The performance assessment and improvement of large distributed systems is challenging, because of the need to systematically assess a complex dynamic ecosystem [1]. The identification of scalability choke points is often expensive and it involves load testing and complex analysis by performance experts.

The extensive body of knowledge has addressed several aspects of Software Performance Antipatterns (*SPAs*), as for example *SPA* classification and solution [2], early detection/solution at the design phase [3], methodologies to rank *SPAs* occurring in design models [4], detection/solution during the testing or operational phases [5], load testing and profiling to detect *SPAs* in Java applications [6], and an automated approach for detection in load testing and production [7].

The state-of-the-art algorithms [3, 6–8] used to detect *SPAs* employ a search over the candidate *SPAs* and all the monitored performance data. The existing approaches use searching for several *SPAs* for each load test result. In addition, for every evaluated *SPA*, a heuristic is executed to evaluate the load test data. Therefore, if there are N load tests, M *SPAs* to be evaluated, and given a worst-case heuristic cost of H_M among the evaluated *SPAs*, the computational complexity of the state-of-the-art algorithms is $O(H_M \cdot N)$. Even though polynomial, the computational effort required for each load test could be expensive. Therefore, the lack of computationally efficient methods for *SPA* characterization limits the adoption of *SPA* detection in continuous integration/delivery/deployment (CI/CDD) pipelines of large and complex systems [9].

In summary, the research gap addressed in this paper is the need to integrate automated *SPA* characterization into CI/CDD pipelines. These approaches need to be computationally efficient, because CI/CDD pipelines are expected to be executed several times per day. In addition, the automated characterization and detection of *SPA*s need to provide useful support to software engineers working on software refactoring in large complex systems.

To deal with this issue, we introduce a novel approach that leverages on the Partial Order Scalogram Analysis by Coordinates (POSAC) [10] approach to partition the *SPA* domain into regions, using the structure induced by the measured variables. Then, we use a multivariate coordinate system [11] for classification and detection of *SPAs*. We characterize *SPAs* using queuing models that were parameterized with measurements from a large telecommunication system. In addition, we apply the response time based scalability approach introduced in our previous work [9], to define the multivariate coordinate system that is used for *SPA* characterization. Our approach reduces the state-of-the-art complexity to linear time, i.e., $O(\Phi_n \cdot N)$, where Φ_n is a value that represents the length of the time series to be processed per load test, and N is the number of load levels to be evaluated per load test, which can usually be bound by a small constant.

In this work, we consider the following *SPA*s: *Application Hiccups* (i.e., repeated violations of the baseline response time requirement); *Continuous Violated Requirements* (i.e., continuous violations of the baseline response time requirement, for every evaluated load); *Traffic Jam* (i.e., high variability in the externally observed system response times); *The Stifle* (i.e., a software service or component that issues many short database calls); *Expensive Database Call* (i.e., few long database calls); *Empty Semi Trucks* (i.e., a transaction that issues many short messages); *The Blob* (i.e., a component or service that acts as a central processor).

In this paper, we pose the following research questions to evaluate the feasibility and usefulness of the proposed approach in CI/CDD pipelines:

RQ1: How to develop an approach to characterize and detect *SPA*s that can be integrated into CI/CDD pipelines?
RQ2: What is the computational complexity of the proposed approach?
RQ3: What is the perceived usefulness of the developed approach?

Our approach has been empirically evaluated by means of an industrial case study conducted at Ericsson. The case is a large real-time telecommunication system that is expected to serve millions of users per second.

The main contributions of this paper can be summarized as follows:

- A new approach for the characterization of *SPAs* using multivariate coordinates.
- A computationally efficient *SPA* detection layered approach that leverages on the multivariate characterization.
- An illustration of the proposed approach using performance data from a large complex telecommunication system.
- A static validation based on expert opinion that quantifies pros and cons of our approach, and highlights the actual needs of developers.

The remainder of the paper is organized as follows: Section 2 provides relevant background and related literature. Section 3 presents the research design and describes the telecommunication system case study used to illustrate and evaluate our approach. Section 4 presents the detailed *SPA* characterization and the empirical results of the application of the *SPA* detection approach to the telecommunication system. Section 5 contains the evaluation of the proposed approach, while Section 6 discusses threats to validity, followed by Section 7, which presents our conclusions and challenges ahead.

2 BACKGROUND AND RELATED WORK

In this section, we present relevant background and an overview of the reviewed work, describing *SPA* detection at different stages of the software development process.

2.1 Multivariate Classification Approaches

A *partially ordered set* (or simply poset) consists of a set along with a binary relation indicating that, for certain pairs of elements, one of the elements is greater than the other one in the ordering. Namely, we say that two elements are numerically ordered, i.e., $e_1 > e_2$ if each variable value in e_1 is greater than the correspondent variable value in e_2.

A Hasse diagram [12] is a graph structure used to represent a poset. A Hasse diagram represents a convenient visualization method when the multiple values associated with the nodes in the poset can be numerically ordered. In this paper, we use Hasse diagrams based on two variable values that are numerically ordered to prioritize the software components' impact on the software scalability of the system under study.

The theory of scalogram algebra and associated heuristics known as Multiple Scaling by Partial Order Scalogram Analysis by Coordinates (POSAC) was introduced in [10] to generalize the concept of the one-dimension Gutman scale [13] to several dimensions. When a one-dimension variable can be numerically ordered, it is called a Guttman scale [13]. In this paper, we apply concepts derived from the POSAC [10] methodology to partition the performance antipattern domain into regions using the structure induced by the measurement variables.

2.2 Literature Review

Logic-based Description of SPAs. Performance antipatterns have been defined in the literature as bad practices leading to performance flaws [2, 14]. The detection and solution of performance antipatterns can be performed at different stages of the software development process, and approaches can be reviewed accordingly.

In the case of early detection/solution, i.e., during the design phase: (i) in [3] a first-order logic representation of performance antipatterns is provided, specifically a set of rules express the system properties under which antipatterns occur; (ii) in [4] a methodology to rank performance antipatterns occurring in Palladio-based design models [15] is proposed and applied to optimize (i.e., by reducing the number of design alternatives to be analyzed) the solution process.

In the case of late detection/solution, i.e., during the testing or operational phases: (i) in [5] a performance antipatterns detection approach is presented, more specifically a decision tree specifying

the performance problems hierarchy is used to capture the root causes of the identified performance problems; (ii) in [6] load testing and profiling data is exploited to detect performance antipatterns in Java applications, and an industrial case study demonstrates the usefulness of detecting and solving antipatterns for system performance improvement.

SPA Detection in Real-time Systems. In [9, 16] a methodology for the quantitative assessment of micro-service architecture deployment alternatives by automated performance testing was introduced. The methodology was developed to be integrated with CI/CDD pipelines, as it assesses scalability by incrementally incorporating results from individual tests that are run for a fixed short time-frame, as for example, 30 minutes.

In [7], response time requirements, measured response time, CPU, network and database utilization were used to detect *SPAs*. A set of algorithms was also introduced in [7] to detect *SPAs* by analyzing the trends of the monitored performance signatures. In contrast, in [3] a logic-based representation of *SPAs* is presented and a more detailed set of performance signatures is used for monitoring. Moreover, the performance signatures being monitored are compared with predefined thresholds. The set of performance signatures that are proposed to be monitored in [3] extends the set proposed in [7] and also includes the number of objects created/deleted, number of connections, number of messages, and other related variables.

In [6, 8], the logic-based representation of performance antipatterns was applied to a large complex Java-based system. The approach presented in [3] was applied to the following *SPAs*: *Extensive Processing*, *Circuitous Treasure Hunt*, and *Wrong Cache*. The approach served as the basis for the implementation of a tool named *PADProf*. The authors have concluded that the logic-based approach was effective to automatically detect *SPAs* in the large complex system studied. However, additional research and testing was required to generalize the application to other systems.

In [17], the feasibility of injecting *SPAs* in a system under study was evaluated. It provided an example of the implementation of a proposed performance antipattern framework, where the *Ramp* and the *One Lane Bridge* implementation were demonstrated. The authors introduced an *SPA* injection framework that was designed to inject problems related to response time and memory use.

Summary of Related Research. The vision of integrating automated detection of *SPA*s into CI/CDD pipelines [9] poses several challenges, as CI/CDD pipelines might be executed several times a day, for many components, and they have a specific performance budget for completion time.

The state of the art on automated detection of *SPA*s [3, 6–8] contains algorithms that analyze detailed monitoring data to detect trends on several monitored variables (e.g., response time, CPU utilization, number of threads) and using pre-defined thresholds.

The research gap addressed in this paper is the lack of a computationally efficient approach to integrate *SPA* characterization and detection into CI/CDD pipelines. We address the aforementioned research gap by introducing a new approach to characterize and detect *SPA*s using multivariate analysis.

3 RESEARCH DESIGN

To address our research questions, we conducted an industrial case study. In this section, we describe the case and the research design of our investigation.

3.1 The Case and Unit of Analysis

The case and unit of analysis of our case study is a large and complex real-time telecommunication system developed by Ericsson. The system is composed by more than 20 subsystems, which are developed using a Service-Oriented Architecture (SOA). The system is developed by many distributed teams using agile software development practices.

The system has hard performance requirements and is expected to handle millions of users per second. Furthermore, the system shows many performance indicators. Therefore, it is challenging to monitor the system's performance through a manual approach. The scale of operation makes the system particularly interesting for our investigation. Furthermore, it represents a relevant and representative example of a performance-critical application.

The main hardware and software performance-critical components of our target system are: network, network interface subsystem, processor subsystem, and database. The system receives requests from a network, which are received by a load balancer and then forwarded to the two subsystems of interest. The network interface subsystem provides services that are invoked by the network. The processor subsystem processes requests that are forwarded by the network interface subsystem.

3.2 Data Collection

We collected two types of data to answer our research questions:

- **Performance data** – Data associated with the performance of the investigated system. This data was used to answer RQ1 and RQ2.
- **Questionnaire data** – Data related to the perceived usefulness of our approach from the point of view of Ericsson experts. This data was used to answer RQ3.

The performance data was the result from a 21-hour long load testing session of the system investigated in this paper. During the testing session, the load varied between 40% and 100% (where 100% represents a load of 16k transactions per second), in steps of 10%. Based on the operational data, the frequency of occurrence of such workload intensity values is as follows: 40% = 0.24, 50% = 0.05, 60% = 0.14, 70% = 0.10, 80% = 0.14, 90% = 0.24, and 100% = 0.10. The data was collected in December 2019 during a load testing session. It includes data associated with 12 services provided by the case software system.

4 THE PROPOSED APPROACH

In this section, we describe our approach that addresses **RQ1** (namely, how to develop an approach to characterize and detect *SPA*s that can be integrated into CI/CD pipelines?). Figure 1 shows a high-level schema of the proposed approach for *SPA* characterization and detection. The shaded boxes represent our contributions. The major boxes are briefly described below.

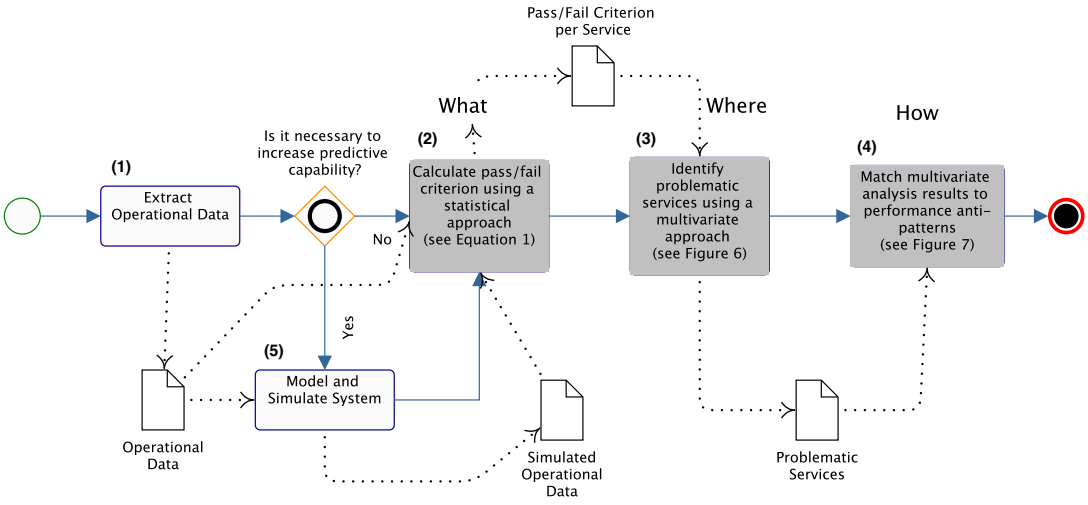

Figure 1: Proposed approach

(1) Extract operational data. This step was described in our previous research [9, 18]. It aims at creating a model of operational usage based on the most-likely usage scenarios.
(2) *What*. Calculate pass/fail criteria based on automatically computed baseline requirements from load test response time measured under small workload.
(3) *Where*. Define a multivariate approach that can be used to create a *partially ordered set* (*poset*) of the evaluated services. Identify a set of services, in the poset, that have the largest impact on system scalability.
(4) *How*. Match multivariate analysis results to performance anti-patterns. Multivariate analysis is used to characterize and detect *SPA*s.
(5) *Model and Simulation*. Conduct simulation activities to validate modeling assumptions. This is left as future research.

Our approach uses a multi-layer decomposition approach based on performance modeling and multivariate characterization of *SPA*s. In doing so, it decouples the performance cost of *SPA* characterization, which can be done offline, from the cost of executing the (*SPA*) detection procedure, which needs to be efficient enough to be integrated into the CI/CDD pipelines. In particular, the cost of *SPA* detection in our approach consists of the execution of the multivariate characterization of load test results; it is not a function of the number of evaluated *SPA*s.

The *induced partition of the studied domain* creates a profile based on the two-coordinate values: (1) *slope* of the fitted linear regression line of the maximum response time measurement for each of the evaluated loads in the y-axis vs. load level in the x-axis; and (2) *normalized distance* between the maximum response time and the baseline performance/scalability requirement.

Intuitively, the slope of the maximum response time regression line is an indication of the performance degradation as a function of the offered load. The normalized distance is an indication of the system's ability to meet the scalability requirements and is related to the customer perception of system performance. In other terms, we rely on the *slope* that aims to express the severity of the performance degradation experienced when evaluating a specific load. The *normalized distance* is instead meant to quantify how much the performance indices deviate from the expectation that is fixed through requirements, such as the system response time.

The normalized distance, *nd*, is evaluated through Equation 1, where M is the measurement and B the baseline.

$$nd = 2 \cdot \frac{M}{M+B} \tag{1}$$

The normalized distance is evaluated to 1 when the measurement is equal to the baseline, < 1 when the measurement is lower than the baseline, and > 1 when the measurement is larger than the baseline.

The remainder of this section is composed of four parts. Section 4.1 presents the performance modeling approach used to characterize *SPA*s. Section 4.2 describes the approach used for performance modeling parameterization using measurements from the case telecom software, as described in Section 3.1. Section 4.3 describes the application of the POSAC procedure to characterize *SPA*s, to define a two-dimensional space to create a framework for the system of observations, and to determine the empirical structure of the modeled universe. Finally, Section 4.4 illustrates the approach using the data from the investigated system.

4.1 SPA Characterization

In this section, we describe how the single server $M/G/1$ model, and its analytical P-K (Pollaczek–Khinchine) [19] solution, can be used to characterize the impact of *SPA*s on the system under study performance, as defined by the average response time and variance.

In the single server $M/G/1$ queueing model, M represents the Poisson process arrivals assumption and G stands for general, i.e., any service time distribution of the single server [19]. $M/G/1$ is the appropriate model to represent *SPA*s, because the most common impact of *SPA*s is to introduce single service bottlenecks [20].

Table 1: SPA modeling parametrization approach

SPA	model	service used	calibration approach	$\overline{X}$	$\overline{X^2}$
Application hiccups	$M/G/1$ with vac.	control	add vacation variability per control	5.01	88.1
Continuous violated req.	$M/D/1$	interrogation + c	add c=5 to baseline	26.95	0
Traffic jam	$M/G/1$	control	double variability	5.01	177.6
The stifle	$M/E_k/1$	interrogation	use 10 stages	99.5	1089.4
Expensive DB call	$M/G/1$	enquiry	double $\overline{X}$	7.2	327.6
Empty semi-trucks	$M/E_k/1$	control	use 10 stages	50.1	276.1
The Blob	$M/G/1$	database management	double $\overline{X}$	6.6	71.98

The $M/G/1$ P-K (Pollaczek–Khinchine) formula [19] for average waiting time in the system, W, and residence time, T, as a function of the first and second moments of the service time distribution (average and variance), $\overline{X}$, and $\overline{X^2}$, and system arrival rate, λ is given by [19]:

$$W = \frac{\lambda \overline{X^2}}{2(1-\lambda \overline{X})} \tag{2}$$

and $T = \overline{X} + W$.

We propose to use a model with Poisson Arrivals (M) and general service distributions (G) for several reasons. First, it allows us to model service variability by the parameterization of G, as shown in Table 1. Second, the Poisson arrivals assumption has been shown to be a reasonable approximation for the arrivals process, in several domains [21], where the independence assumption for the user arrival process can be modeled [19]. Finally, $M/G/1$ is the most general model for which we have a simple closed form analytical solution for the average waiting time (W) [19].

In the following, we describe how the $M/G/1$ model can be used to characterize the performance of the *SPAs* considered in this paper.

4.1.1 Application Hiccups. The *Application Hiccups* [7] is characterized by repeated violations of the scalability requirement. These violations may occur as a consequence of different events, such as transitions from idle to busy state, processing of preemptive tasks (e.g., garbage collection), or database backups. These tasks might reduce available system capacity, therefore impacting system performance. There are several approaches for modeling the *application hiccups*, such as, $M/G/1$ queues with vacations that occur when the system transitions to the idle state, and using queues with preemptive or non-preemptive priorities. In this section, we model *Application Hiccups* by using the $M/G/1$ with vacations model, described below. Using the same notation as in Equation (2), and using as the first and second moments of the vacation time distribution, $\overline{V}$, and $\overline{V^2}$, the $M/G/1$ with vacations average waiting time in the system is given by [19] and shown in Equation 3:

$$W = \frac{\lambda \overline{X^2}}{2(1-\lambda \overline{X})} + \frac{\overline{V^2}}{2\overline{V}} \tag{3}$$

4.1.2 Continuous Violated Requirements. The Continuous Violated Requirements [7] is characterized by the continuous violations of the scalability requirement, for every evaluated load. The approach used in this section, for modeling the Continuous Violated Requirements is to use an $M/D/1$ queue, with an average service time larger than the computed baseline requirement. D stands for deterministic and represents a constant service time.

4.1.3 Traffic Jam. The Traffic Jam software antipattern represents high variability in the externally observed system response times that is caused by queuing at software resources. In this section, we model the Traffic Jam *SPA* by using the $M/G/1$ queuing system, and modeling the increase in the system variability in the departure process function G.

4.1.4 The Stifle. The Stifle software antipattern [7] represents a software component that issues many short database calls to implement a service. In this section, the Stifle *SPA* is modeled by using the $M/G/1$ queueing system, where G is modeled by an Erlangian distribution with $k - stages$ [19], $M/E_k/1$. Each of the k states in the Erlangian distribution is used to model the Stifle fan-out of one call to k serial database calls.

4.1.5 Expensive Database Call. The Expensive Database Call software antipattern [7] represents few long database calls and can be modeled by using an $M/G/1$ queuing system with a long tail distribution for service times.

4.1.6 Empty Semi Trucks. The Empty Semi Trucks software antipattern [7] represents a transaction that issues many short messages in series to implement the transaction. Therefore, this antipattern can be modeled similarly to the *Stifle* antipattern, by using a $M/G/1$ queueing system, where G is modeled by an Erlangian distribution with $k - stages$ [19], $M/E_k/1$.

4.1.7 The Blob. The Blob antipattern [7] represents a component that manages most of the overall messages in the system. As a consequence of being the focus of messages, message processing performance degradation is observed. Therefore, the Blob software antipattern can be modeled similarly to the Expensive Database Call software antipattern. In Section 4.2 we describe the approach that was used for the parameterization of the *SPA* models just described, by using measurements derived from the large telecom software, as described in Section 3.1.

4.2 Analytical Model Parametrization

In this section, we present an illustration of one possible *calibration* that was used for the parameterization of the *SPAs* performance models.

The *SPA* model parametrization approach shown in Table 1 presents the model used, the service from the case software selected for calibration, the calibration approach, and the first and second moments selected to represent the *SPA*.

Table 2: Partition induced by slope and normalized distance by the *SPA*s, for the load 90%

	normalized distance	slope
Application hiccups	1.74	130.42
Continuous Violated Requirements	1.08	0.00
Traffic Jam	1.72	260.84
The Stifle	1.60	80.54
Expensive Database Call	1.59	334.51
Empty Semi-trucks	1.47	40.55
The Blob	1.51	79.56

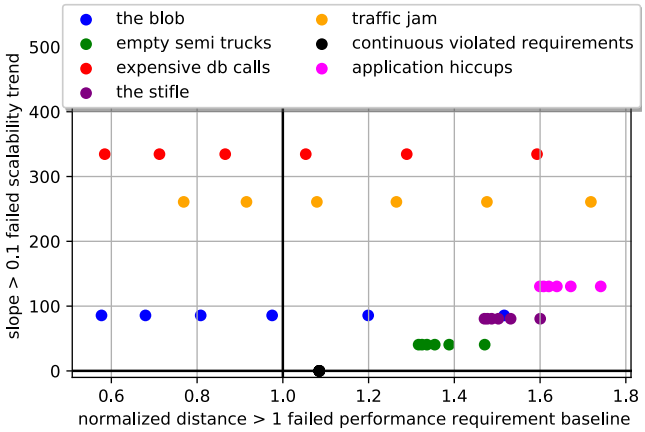

Figure 2: Partition induced by slope and normalized distance by the *SPA*s analyzed, for the different loads set to 40%, 50%, 60%, 70%, 80%, 90%

The results obtained for each *SPA* multivariate characterization are presented in Table 2 and Figure 2. They show the two-dimensional (x, y) coordinates, i.e., the normalized distance and slope, for each of the evaluated *SPAs*. Table 2 shows the multivariate pair for the 90% load experiment, while Figure 2 shows the multivariate pairs associated with the evaluated *SPAs*, for loads varying from 40% to 90%.

It is worth considering the slope and normalized distance coordinates for the analyzed *SPA* in Figure 3. The figure includes some antipatterns that can be clearly identifed (e.g., expensive DB calls), while other points appear to be quite close to each other in the solution space (e.g., the Blob and the Stiffle). The strength of the proposed approach is the computational efficiency of detecting *SPA* using multivariate analysis. There is a need for extending our approach by evaluating accuracy of detection and near neighbours *SPA* detection. One option for further research is to consider *SPA detection probability.*

4.3 SPA Detection

In [11], a step-by-step procedure is presented to illustrate the application of Multiple Scaling by Partial Order Scalogram Analysis by Coordinates (POSAC) in the behavioral sciences domain. POSAC

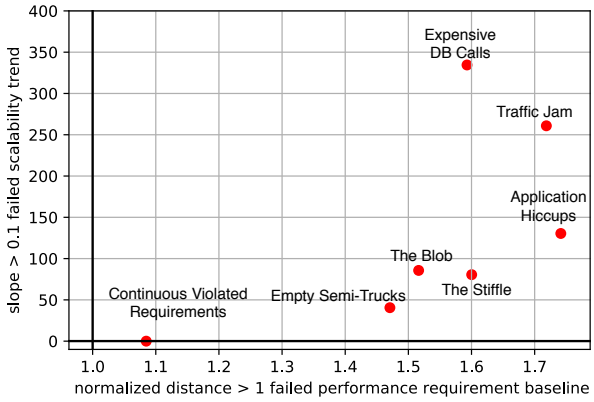

Figure 3: Partition induced by slope and normalized distance by the *SPA*s analyzed, for loads = 90%

induces a partition of the studied space, because profiles are distinguished by high/low values of the x, y coordinates. In the application of POSAC described in [22] the interpretation of the POSAC coordinates represents the subjective concern in the target research community. For example, in the assessment of quality of life, a two-dimensional space representing intelligence and well-being was used.

In this section, we present the application of POSAC to the *SPA* detection domain. The x coordinate, the normalized distance of the performance requirement, represents the impact of response time requirement violation to the user, or user well-being [22]. In addition, the y coordinate, the slope of the linear regression of response time as load levels increase, represents the risk that the user will be impacted by the response time degradation, or system intelligence [22].

The notion of POSAC can be used to partition the *SPA* domain by means of the following steps:

(1) Define a mapping sentence to create a framework for the system of observations.
(2) Determine the empirical structure of the content universe.
(3) Execute the POSAC algorithm using the (x, y) coordinates of the location of their profile in space.
(4) Interpret the POSAC coordinates to understand the induced partition on the POSAC space.

To interpret the POSAC coordinates and to map the induced partition on the POSAC domain from measurements of load testing results to the associated *SPA*s, we define the multivariate mapping using as (x, y) coordinates the normalized distance and slope of the response time.

In the following, we instantiate the *Antipattern Characterization and Detection* approach by outlining the specific methods proposed for SPA characterization and detection in the context of our research:

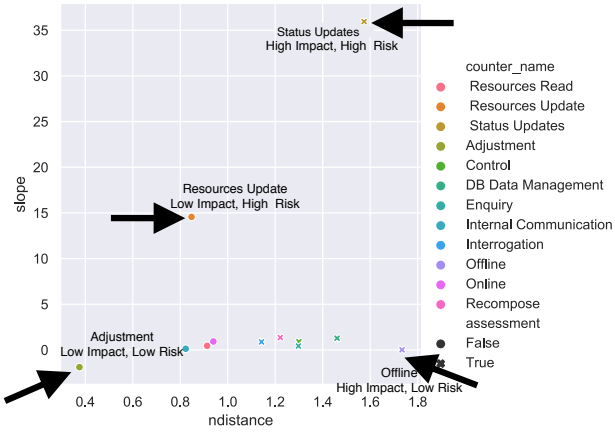

Figure 4: System services for load = 90%

(1) Define a scalability requirement baseline, as introduced in [9, 16] that is used to provide an automated scalability assessment, i.e., the pass/fail criteria of load tests. Figure 4 illustrates the pass/fail assessment of the load tests analyzed in the case study.
(2) Define a multivariate approach, based on the load test response time measurements that can be used to create a partial order of the evaluated services.
In this paper, we have used as the multivariate system of coordinates: (1) the slope of the linear regression of response times per load, sl, and, (2) the normalized distance, nd, between the measured response time and the defined baseline requirement, as shown in Equation 1.
(3) Define a function $f(sl, nd)$ to charaterize SPAs by using the defined multivariate approach. An example of the multivariate characterization of the considered *SPAs* is shown in Figure 3.
(4) Identify a set of services, $s_i(sl, nd)$, in the poset, that have the largest impact on system scalability. The set $s_i(sl, nd)$ can be found among the members of the top two rows in Figure 5. The detailed values used to create the Hasse diagram are shown in Table 3. For instance, we can notice that the *Status Updates* service shows the largest slope (i.e., 35.95), indicating that it is the service experiencing the highest performance degradation rate. The members of the example set, $s_i(sl, nd)$, for the telecom application are: Status-updates, Offline, Resources-update, DB-Data-Management, and, Recompose.
(5) Define a detection function:

$$D(f(sl, nd), (s_i(sl, nd))) \quad (4)$$

to detect SPAs characterized by $f(sl, nd)$ that occur in the set $s_i(sl, nd)$. An example of the evaluation of the detection function, for the telecom example shown in Figure 6, is the euclidean distance, where Status Updates is detected to belong to the Expensive Database Call *SPA*.
(6) SPA detection is implemented by evaluating the function D in Equation 4.

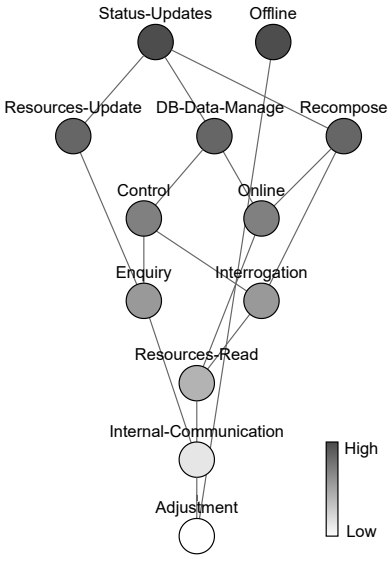

Figure 5: Hasse Diagram using performance testing results for a specific load, i.e., equal to 90%

Table 3: Multivariate data for each system service for a specific load equal to 90%

service	normalized distance	slope
Adjustment	0.38	-1.88
Enquiry	1.30	0.43
Interrogation	1.14	0.88
Resources Read	0.91	0.45
Resources Update	0.85	14.56
Status Updates	1.57	35.95
Control	1.30	0.90
DB Data Management	1.46	1.28
Internal Communication	0.82	0.13
Offline	1.73	0.01
Online	0.94	0.92
Recompose	1.22	1.35

4.4 Illustration of the Proposed Approach

We use the data obtained from the investigated system to illustrate our approach. The computation of the pass/fail criteria (performance/scalability requirement) uses the approach proposed in [16], which calculates the pass/fail criteria as the *average* + 3 · *standard deviations* of the no-load response time measurement. Figure 4 shows a plot with slope (y axis) and normalized distance (x axis) for the load = 90%.

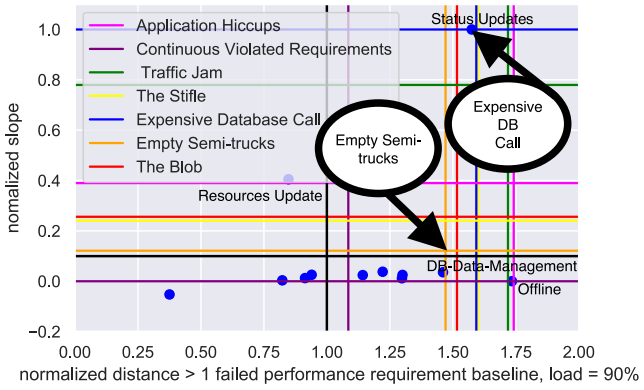

Figure 6: Partition induced by the normalized slope and normalized distance coordinates, for a specific load, i.e., equal to 90%

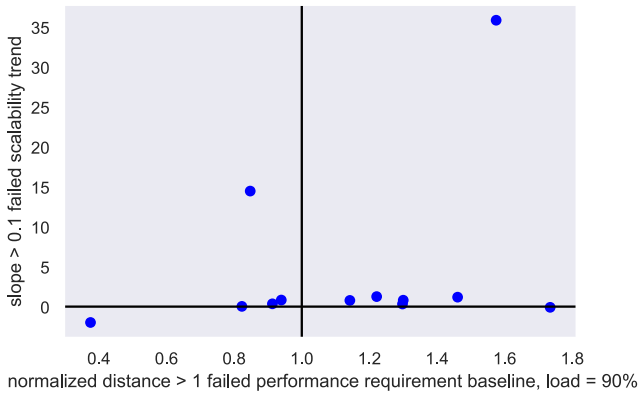

Figure 7: Partition induced by slope and normalized distance on the investigated system services for load = 90%

Using a risk assessment approach, we can observe that services of the investigated system can be grouped into four risk partitions (see Figure 7 for an example of risk assessment for the 90% load in the case software):

(1) **Low risk / Low Impact** — Performance requirements met and no scalability degradation detected. The services that lie in the lower left quadrant have a slope less than 0.1 and a normalized distance less than 1.
(2) **High risk / Low Impact** — Performance requirements met and scalability degradation detected. The services that lie in the upper left quadrant have a slope greater than 0.1 and a normalized distance less than 1.
(3) **High risk / High Impact** — Performance requirements not met and scalability degradation detected. The services that lie in the upper right quadrant have a slope greater than 0.1 and normalized distance greater than 1.
(4) **Low risk / High Impact** — Performance requirements not met and no scalability degradation detected. The services that lie in the lower right quadrant have slope less than 0.1 and normalized distance greater than 1.

SPA detection — Figure 6 shows how the load testing induced partition shown in Figure 7 can be overlayed over the partition induced by the *SPA*s, as shown in Figure 3. *SPA* detection is implemented by using slope normalization, and the definition of a distance function between the load test (x_1, y_1) coordinates, and the *SPA* characterization (x_2, y_2) coordinates.

5 EVALUATION

In this section, we present the adopted evaluation method and the evaluation results.

5.1 Evaluation Method

We evaluated our approach using data from the system described in Section 3.1. The evaluation has two parts:

(1) We have compared the computational efficiency of the proposed approach with some of the algorithms presented in [7] (RQ2). This step used the performance data.
(2) We have presented our approach to experts in Ericsson, along with the results of employing it in the investigated system to answer RQ3. Then, we asked the participants to answer a questionnaire that aimed at identifying the usefulness of the proposed approach from the point of view of potential users. This step used both the performance and the questionnaire data.

Next we show more details about each of the evaluation steps.

Computational Efficiency. The analysis of algorithms approach (big-O) used in this paper [23], estimates an upper bound on the required number of computations that are executed by the evaluated algorithm, when presented with a large enough number of data inputs. The big-O methodology consists of expressing an upper bound on the algorithm complexity growth function. More formally [24],

Definition of big-O: If $a(n)$ and $b(n)$ are two positive valued functions, we define that $a(n) = O(b(n))$, if there exists a constant, K, which satisfies the condition $a(n) \leq Kb(n)$ for all, but a finite number of instances of n.

In Section 5.2.1, we present results of the application of the big-O [23] methodology summarized above to compare the computational efficiency of the proposed approach with some of the algorithms presented in [7].

The notation used in the computation efficiency assessment is presented in Table 4.

Perceived Usefulness — Questionnaire. We have developed a questionnaire to obtain the perceived usefulness of our approach from the point of view of potential users. First, we conducted a workshop in June 2020 to present our approach and associated results to experts in Ericsson. Thirty participants attended the workshop (e.g., testers and developers). Second, we asked the participants to answer a questionnaire and obtained answers from eight different respondents. The questionnaire has 10 questions. Eight questions are related to demographics and the opinion of the respondents regarding performance antipattern detection and characterization in general. We asked these questions to have a better understanding of the respondents' demographics and contexts. The remaining two questions are about the usefulness of our approach.

Table 4: Computation efficiency notation

Notation	Meaning
N	Number of load levels evaluated, one per load test
Φ_n	Length of response time series to be processed by *SPA* heuristics, per load test
σ_n	Length of sql response time series to be processed by *SPA* heuristics, per load test
δ_n	Length of messaging dataset to be processed by *SPA* heuristics, per load test
τ_n	Length of message tracing dataset to be processed by *SPA* heuristics, per load test
M	number of *SPA* to be evaluated
$O(s)$	big-O upper bound on the algorithm complexity of function s
$O(\mathcal{H}_\mathcal{M})$	the upper bound on the algorithm complexity of the worst-case *SPA* heuristic

5.2 Evaluation Results

In this section, we present the results associated with the evaluation of the proposed approach, where each evaluation is grouped by the corresponding research question.

5.2.1 Computational Efficiency (RQ2). In this subsection, we compare the computational efficiency of the algorithmic approach presented in previous work [7] with the computational efficiency of the statistical approach introduced in this paper. This subsection addresses **RQ2**: *What is the computational complexity of the proposed approach?*

The methodology for detection of *SPAs* is composed of two parts: 1) an offline training phase, and, 2) an online prediction phase.

In the analyzed literature, the training phase consists mostly of threshold definitions for each of the analyzed *SPAs* [3, 6–8], and the prediction phase employs specific algorithms defined for each of the candidate *SPAs*. Therefore, the ultimate online detection of the specified *SPAs* requires the execution of the specified prediction algorithms to detect one or more *SPA*.

In the novel approach introduced in this paper, the training phase consists of *SPA* parametrization using analytical modeling to identify the partitions of the *SPA* domain that are induced by the measurement variables. This training phase methodology is illustrated in Figure 3, where the partitions induced by the normalized slope, and, the performance requirements normalized distance, for load = 90% are shown. In the statistical approach, the online detection methodology consists of mapping load test results into the specified *SPA* partitions, as illustrated by the blue dots in Figure 6.

Algorithmic approach from previous research. The offline training of the heuristics presented in [7] consists of determining the parameters to be used in the *SPA* prediction heuristics, and is out of scope of this section, because the focus of this analysis is on determining the computational complexity of online prediction. The state of the art approaches presented in [7] execute heuristics to detect *SPAs*, for each load test result. Therefore, if there are N load tests, M *SPAs* to be evaluated, and given that the computational complexity of the worst-case heuristic is defined as $O(H_M)$, the computational complexity of the state of the art algorithms is $O(H_M \cdot N)$. In this section we present a summary of the results obtained from the analysis of the heuristics presented in [7].

Online prediction. We illustrate the approach for the computational complexity evaluation of the online prediction heuristics presented in [7], by using as example, the application hiccups and continuous violated requirement *SPAs*. These *SPA* prediction heuristics are organized as two nested loops. Therefore, the computational complexity of these two anti-patterns is estimated as $O(\Phi_n^2)$. The outer loop scans the response time series of length Φ_n, ordered by timestamp, and passes a starting pointer to the inner loops to execute a linear search on the series, and perform calculations to detect *SPAs*. We present in Table 5 the justifications for the estimated worst-case computational complexity for each of the *SPA* prediction heuristics introduced in [7]. Adding all the computational complexities in Table 5, we get:

Table 5: SPA Heuristics Computational Complexity [7] summary

SPA Heuristic	O	Justification
Application hiccups	Φ_n^2	two nested loops
Continuous violated req.	Φ_n^2	two nested loops
Traffic jam	Φ_n^2	outer loop on response time series and inner loop for linear regression
The stifle	$\sigma_n \cdot \Phi_n$	two nested loops on response time and SQL response time series
Expensive DB call	$\sigma_n \cdot \Phi_n$	two nested loops on response time and SQL response time series
Empty semi-trucks	τ_n	loop on message tracing dataset
The blob	δ_n	loop on messaging dataset

$$3\Phi_n^2 + 2\sigma_n \cdot \Phi_n + \tau_n + \delta_n \tag{5}$$

Therefore, we have found that

$$O(\mathcal{H}_\mathcal{M}) = \Phi_n^2 + \sigma_n \cdot \Phi_n + \tau_n + \delta_n \tag{6}$$

Statistical approach introduced in this paper. In this section, we evaluate the computational complexity of *SPA* detection for the approach introduced in this paper.

Offline training. The number of *SPAs* to be evaluated in offline training is M. The approach introduced in this paper calls for analytical modeling, parametrization, and solution of the M *SPAs* for N load levels. Therefore, the computational complexity of the offline

training approach introduced in this paper, can be computed using the following steps:

(1) Parametrization and solution of Equation 3, for each SPA and each evaluated load level. Therefore, the computational complexity of this step is $O(M \cdot N)$.
(2) Computation of normalized distance, for each *SPA*, for the evaluated load level. For example, in Figure 6, we used the load level of 90%. Therefore, the computational complexity of this step is $O(M)$.
(3) The computational complexity of linear regression of one variable, slope in this case, is linear $O(N)$ [25]. Therefore, the computational complexity of obtaining the slope of the response vs. load curve using linear regression, for each *SPA* is $O(M \cdot N)$.

In summary, the computational complexity of the offline training approach introduced in this paper, can be computed as $O(M \cdot N)$, which is $O(KM) = O(M)$, because, for *SPA* detection, we can assume N is $O(K)$, where K is a small constant. The number of *SPAs* to be evaluated is likely to grow, but the number of load levels evaluated is usually a number less than 10.

Online prediction. The computational complexity of the online training approach introduced in this paper, can be computed as $O(\Phi_n \cdot N)$, using the following steps:

(1) Compute the normalized distance for the evaluated load level. Therefore, the computational complexity of this step is $O(\Phi_n \cdot N)$.
(2) Compute the slope of the response time vs. load level plot using linear regression of the load testing results. Similarly to the offline computation, the computational complexity of linear regression of one variable, slope in this case, is linear $O(\Phi_n \cdot N)$.

As a result of these steps, load test results can be mapped into the specified *SPA* partitions, as illustrated by the blue dots in Figure 6.

5.2.2 Perceived Usefulness (RQ3). This subsection addresses **RQ3**: *What is the perceived usefulness of the developed approach?*

The results associated with the demographics/context questions are presented next. Figure 8 shows that the majority (38%) of the questionnaire respondents are testers.

Figure 9 shows that the majority of the respondents (63%) has more than 10 years of IT experience, while Figure 10 shows that most of them have between one to five years of experience regarding software performance testing.

Most of the respondents answered that they manually set the pass/fail criteria used in their performance testing efforts (63%, Figure 11), while there is an almost equal number of respondents that believe that the raw log (37%) and the service (38%) levels are the minimum required granularity level to identify performance problems and enable reengineering activities (Figure 12).

The perceived usefulness questionnaire results are presented in Table 6. We have also asked about the main challenges faced by the respondents when carrying out performance testing. Table 7 shows that the main challenges faced by the majority of the respondents are Analyzing the Performance Test Outcomes (16%), Defining a Testing Strategy (15%), and End of Project Life cycle (15%).

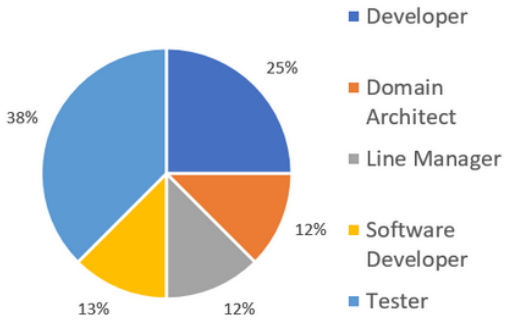

Figure 8: Role of the respondents

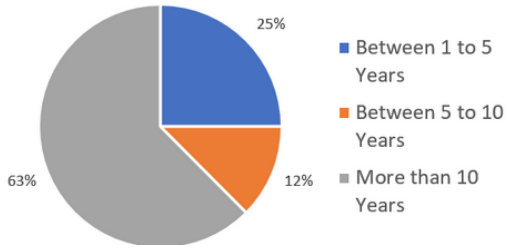

Figure 9: Years of experience of the respondents in IT

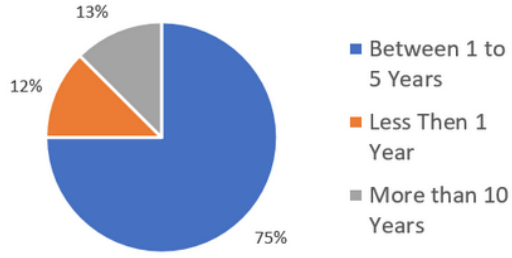

Figure 10: Years of experience of the respondents in performance testing

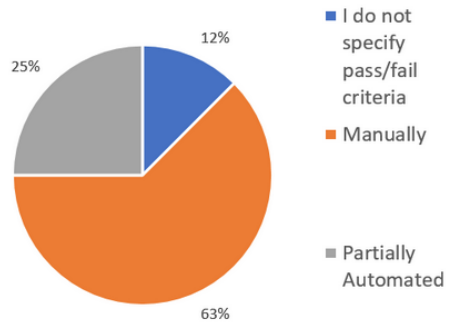

Figure 11: How the respondents set pass/fail criteria

Table 6: Perceived Usefulness

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Reduce performance testing cost	25%	62%	0	13%	0
Reduce deployment cost	38%	37%	13%	12%	0
Reduce time to deploy	38%	37%	25%	0	0
Increase software quality	75%	12%	0	13%	0
Increase software power consumption efficiency	13%	62%	25%	0	0
Increase software reliability	63%	25%	0	12%	0

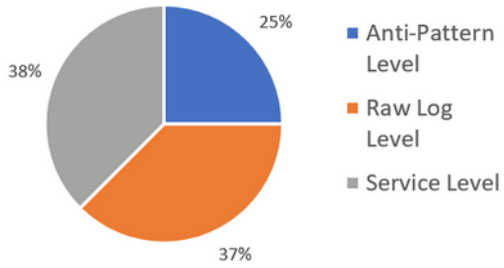

Figure 12: The minimum required granularity level for software performance problem detection to improve reengineering activities

Table 7: The biggest challenges faced by the respondents regarding software performance testing

Challenge	Percentage
Analyzing the Performance Test Outcomes	16%
Defining a Testing Strategy	15%
End of Project Life cycle (Testing is done at the end, when it's late)	15%
Lack of Available Human Resources	9%
Lack of Domain Competence	9%
Lack of Recognition of the value of performance testing	9%
Limited Time and Budget	6%
Model Scenario and Assertions	6%
System Complexity	6%
Test Coverage	6%
To do performance testing on a system that are in functional development	3%

Table 8: Impact of the investigated SPAs

	Small	Moderate	Large	Don't know
The Blob	0	75%	0	25%
Empty Semi-Trucks	0	38%	12%	50%
Expensive DB Calls	0	0	75%	25%
The Stifle	0	25%	50%	25%
Traffic Jam	0	0	75%	25%
Consistent Violated Requirements	0	13%	50%	37%
Application Hiccups	0	25%	50%	25%

Finally, Table 8 shows the opinion of the respondents on the investigated performance antipatterns. The antipatterns that were considered by the majority of the respondents (75%) as having a large impact are Traffic Jam and Expensive DB Calls. These results are summarized in our conclusions, see Section 7.

6 THREATS TO VALIDITY

In this section, we discuss the threats to the validity of our investigation using the categories reliability, internal, construct, and external validity described by Runeson and Höst [26].

We mitigated *reliability* threats by involving several researchers in the design and execution of our investigation. Moreover, our results were verified with the help of Ericsson experts to avoid false interpretations. However, the approach we used to assess the usefulness of our approach was highly dependent on the questionnaire respondents, i.e., it might be difficult to obtain the same values with other respondents.

Regarding *internal* validity, the performance of the investigated system may be impacted by factors that were not present in the performance data used to illustrate our approach and to conduct part of the evaluation. This is a limitation that might have affected the related evaluation steps. We plan to mitigate this threat by applying the methodology to additional projects and by creating a PCM-based simulation model.

Regarding the questionnaire-based evaluation, the main internal validity threats is respondent bias. To mitigate this threat, we invited people with different roles in Ericsson (data triangulation).

The main threat to *construct* validity is that we used only one method to measure a construct. To mitigate this threat in the case study, we collected data from different sources to evaluate our approach (data triangulation).

Regarding *external* validity, although our method was developed using an analytical approach and is expected to be quite generic, the evaluation used data from just one system from one company. Furthermore, only experts from Ericsson were asked to give their opinion about the usefulness of our approach. As a consequence, the evaluation results are strongly bound by the context of our investigation. In addition, the investigated case involved only one product in one company. To mitigate this threat, we made an attempt to describe the context of our study in as much detail as possible. This makes it easier to relate the present case to similar cases, facilitating the use of our approach in other contexts.

7 CONCLUSION AND FUTURE WORK

In this paper, we have introduced a new approach for the characterization and the detection of *SPAs* that was designed to be integrated into CI/CDD pipelines, since its implementation is computationally efficient.

We have compared the computational efficiency of the proposed approach with state-of-the-art heuristics, and we have found that the approach introduced in this paper grows linearly as $O(\Phi_n)$,

while the computational complexity of the state-of-the-art heuristics grows as $O(\Phi_n^2)$. Moreover, because the methodology characterizes each *SPA* using a multivariate approach that is based on the response time measurement, it can be efficiently implemented and deployed in industry, as response time measurements are normally logged in load testing experiments. In contrast, existing *SPA* approaches may require additional datasets for *SPA* analysis that might not be readily available. The approach has been successfully applied to a large and complex project at Ericsson.

The introduced approach can be applied to continuous development ecosystems that collect response time measurements logs, and are required to meet performance and scalability requirements. The application domains of interest are related to critical infrastructures [27], such as telecommunication and banking.

We have performed an analysis of the perceived usefulness of the proposed approach using a questionnaire that was responded by practitioners with significant development experience. Most of the responses are in the agree and strong-agree columns. In fact, the questions related to increase software quality, reduce performance testing cost, and increase software reliability, received an over 87% cumulative score of agree and strongly agree.

We are currently planning to extend this research in several ways. Our research agenda includes the following points:

- *Accuracy analysis* – We plan to assess accuracy by applying the approach to additional projects at Ericsson to obtain additional feedback from experienced software engineers. Specifically, we would like to investigate the sensitivity of the methodology to different approaches to further assess *SPA* model calibration.
- *Integration* – We plan to integrate the introduced approach for automated *SPA* detection into automated performance testing and analysis tools [9].
- *Modeling and Simulation* – We plan to validate the assumptions used in the *SPA* modeling presented in Section 4.1 by model-based simulation of the system's performance properties. Specifically, we plan to use Palladio [15] to model the case study presented in this paper and to assess the performance impact of *SPAs*. The Palladio Component Model (PCM) is a domain-specific language targeted at specifying and documenting software architectural knowledge. Furthermore, the simulation-based approach allows us to validate recall and precision of our approach by injecting arbitrary anti-patterns in the simulated models.

We expect the new methodology introduced in this paper to trigger research and development on new methods and tools to automatically characterize *SPA*s using the introduced multivariate characterization approach.

ACKNOWLEDGMENTS

This work has been partially funded by eSulab Solutions, the MIUR PRIN project 2017TWRCNB SEDUCE (Designing Spatially Distributed Cyber-Physical Systems under Uncertainty), and the European Union's Horizon 2020 research and innovation programme (grant no. 825040, RADON).

REFERENCES

[1] Microsoft, *Performance tuning a distributed application*, 2019 (accessed June 25, 2020), https://docs.microsoft.com/en-us/azure/architecture/performance/.
[2] C. U. Smith and L. G. Williams, "Software performance antipatterns for identifying and correcting performance problems," in *International Computer Measurement Group Conference*, 2012.
[3] V. Cortellessa, A. Di Marco, and C. Trubiani, "An approach for modeling and detecting software performance antipatterns based on first-order logics," *Software & Systems Modeling*, vol. 13, no. 1, pp. 391–432, 2014.
[4] C. Trubiani, A. Koziolek, V. Cortellessa, and R. H. Reussner, "Guilt-based handling of software performance antipatterns in Palladio architectural models," *J. Syst. Softw.*, vol. 95, pp. 141–165, 2014.
[5] A. Wert, J. Happe, and L. Happe, "Supporting swift reaction: automatically uncovering performance problems by systematic experiments," in *International Conference on Software Engineering (ICSE)*, 2013, pp. 552–561.
[6] C. Trubiani, A. Bran, A. van Hoorn, A. Avritzer, and H. Knoche, "Exploiting load testing and profiling for performance antipattern detection," *Information and Software Technology*, vol. 95, pp. 329 – 345, 2018.
[7] A. Wert, "Performance problem diagnosis by systematic experimentation," Ph.D. dissertation, 2015.
[8] A. Bran, "Detecting software performance anti-patterns from profiler data," B.S. thesis, 2017.
[9] A. Avritzer, V. Ferme, A. Janes, B. Russo, A. van Hoorn, H. Schulz, D. Menasché, and V. Rufino, "Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests," *Journal of Systems and Software*, 2020.
[10] S. Shye, *Multiple Scaling: The Theory and Application of Partial Order Scalogram Analysis.* Amsterdam: North-Holland, 1985.
[11] ——, *Partial order scalogram analysis by coordinates (POSAC) as a facet theory measurement procedure: how to do posac in four simple steps*, 01 2009, pp. 295–310.
[12] S. Skiena, *Hasse Diagrams.* §5.4.2 in Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Reading, MA: Addison-Wesley, p. 163, 169-170, and 206-208, 1990.
[13] F. Maggino, *Guttman Scale.* Dordrecht: Springer Netherlands, 2014, pp. 2626–2630.
[14] T. Parsons, J. Murphy *et al.*, "Detecting performance antipatterns in component based enterprise systems." *Journal of Object Technology*, vol. 7, no. 3, pp. 55–91, 2008.
[15] R. H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann, *Modeling and Simulating Software Architectures – The Palladio Approach.* MIT Press, 2016.
[16] A. Avritzer, V. Ferme, A. Janes, B. Russo, H. Schulz, and A. van Hoorn, "A quantitative approach for the assessment of microservice architecture deployment alternatives by automated performance testing," in *Proceedings of the 12th European Conference on Software Architecture (ECSA)*, 2018, pp. 159–174.
[17] P. Keck, A. van Hoorn, D. Okanović, T. Pitakrat, and T. F. Düllmann, "Antipattern-based problem injection for assessing performance and reliability evaluation techniques," in *Proceedings of the International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2016, pp. 64–70.
[18] A. Avritzer and E. J. Weyuker, "The automatic generation of load test suites and the assessment of the resulting software," *IEEE Trans. Softw. Eng.*, vol. 21, no. 9, 1995.
[19] D. Bertsekas and R. Gallager, *Data Networks (2nd Ed.).* USA: Prentice-Hall, Inc., 1992.
[20] A. Avritzer and E. J. Weyuker, "The role of modeling in the performance testing of e-commerce applications," *IEEE Transactions on Software Engineering*, vol. 30, no. 12, pp. 1072–1083, 2004.
[21] L. Kleinrock and K. M. R. Collection, *Queueing Systems, Volume I*, ser. A Wiley-Interscience publication. Wiley, 1974.
[22] S. Shye, *Systemic Quality of Life Model (SQOL)*, 03 2014, pp. 6569–6575.
[23] D. E. Knuth, *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms.* USA: Addison Wesley Longman Publishing Co., Inc., 1997.
[24] A. A. Nasar, "The history of algorithmic complexity," *The Mathematics Enthusiast*, vol. 13, no. 3, 2016.
[25] M. Hladík and M. Černý, "Total least squares and chebyshev norm," *Procedia Computer Science*, vol. 51, pp. 1791 – 1800, 2015, International Conference On Computational Science, ICCS 2015.
[26] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples.* John Wiley & Sons, 2012.
[27] A. Avritzer, F. D. Giandomenico, A. K. I. Remke, and M. Riedl, *Assessing dependability and resilience in critical infrastructures: challenges and opportunities.* Springer, 2012, pp. 41–63.