IEEE TRANSACTIONS ON
EMERGING TOPICS IN COMPUTING

Received 31 July 2018; revised 12 April 2019; accepted 19 April 2019.
Date of publication 10 July 2019; date of current version 4 June 2021.

Digital Object Identifier 10.1109/TETC.2019.2919801

An Efficient Monte Carlo-Based Probabilistic Time-Dependent Routing Calculation Targeting a Server-Side Car Navigation System

EMANUELE VITALI, DAVIDE GADIOLI, GIANLUCA PALERMO, (Member, IEEE), MARTIN GOLASOWSKI, JOÃO BISPO, PEDRO PINTO, JAN MARTINOVIČ, KATEŘINA SLANINOVÁ, JOÃO M. P. CARDOSO, (Senior Member, IEEE), AND CRISTINA SILVANO, (Fellow, IEEE)

E. Vitali, D. Gadioli, G. Palermo, and C. Silvano are with the Dipartimento di Elettronica, Infomazione e Bioingegneria, Politecnico di Milano, Milano 20133, Italy
M. Golasowski, J. Martinovič, and K. Slaninová are with the IT4Innovations, VŠB - Technical University of Ostrava, Ostrava 708 00, Czech Republic
J. Bispo, P. Pinto, and J.M.P. Cardoso are with the Faculty of Engineering (FEUP), University of Porto, Porto 4099-002, Portugal
CORRESPONDING AUTHOR: G. PALERMO (gianluca.palermo@polimi.it)

ABSTRACT Incorporating speed probability distribution to the computation of the route planning in car navigation systems guarantees more accurate and precise responses. In this paper, we propose a novel approach for selecting dynamically the number of samples used for the Monte Carlo simulation to solve the Probabilistic Time-Dependent Routing (PTDR) problem, thus improving the computation efficiency. The proposed method is used to determine in a proactive manner the number of simulations to be done to extract the travel-time estimation for each specific request, while respecting an error threshold as output quality level. The methodology requires a reduced effort on the application development side. We adopted an aspect-oriented programming language (LARA) together with a flexible dynamic autotuning library (mARGOt) respectively to instrument the code and to make decisions on tuning the number of samples to improve the execution efficiency. Experimental results demonstrate that the proposed adaptive approach saves a large fraction of simulations (between 36 and 81 percent) with respect to a static approach, while considering different traffic situations, paths and error requirements. Given the negligible runtime overhead of the proposed approach, the execution-time speedup is between 1.5x and 5.1x. This speedup is reflected at the infrastructure-level in terms of a reduction of 36 percent of the computing resources needed to support the whole navigation pipeline.

INDEX TERMS High performance computing, approximate computing, adaptive applications, smart cities, vehicle routing

I. INTRODUCTION

In smart cities, the trend is to combine and automate several common tasks to ease the life of citizens. Among these tasks, traffic estimation and prediction play a central role not only to avoid traffic congestion, thus enabling predictable travel times, but also reducing car emissions. Considering the rising wave of self-driving cars, the amount of car navigation requests will increase rapidly together with the need of real-time updates and processing on large graphs representing the urban network. This trend imposes large and powerful computing infrastructures based on HPC resources.

Concerning the algorithmic problem, car navigation is one of the main problems of the applied theoretical research. The Dijsktra's shortest path algorithm is used to find the optimal path between two vertices in a weighted graph representing a road network. Apart from the single navigation between two points, navigation algorithms are used in various systems for solving larger optimization problems, like route planning for a fleet of package delivery vehicles, waste collection management and traffic optimization in a smart city [1]. Definition of the optimal path is based on the type of used weights of the graph edges. The shortest path is based on the geographical distance between two adjacent vertices of a graph. The fastest path is based on the time needed to cross a particular edge. There might be more complex criteria, however their description is out of the scope of this paper. Time needed to cross a particular stretch of road can be affected by various elements, such as accidents, traffic congestion, road works and so on. At the basic level, an upper legal limit of speed is used, based on the assumption that each vehicle travels with the same speed. This can be vastly inaccurate due to the intrinsic natural behavior of the traffic.

With the increasing availability of historical traffic monitoring data, there are several research efforts to determine the average speed on road networks by using statistical analysis and various models. However, a single speed value is still not very useful as it does not reflect the stochastic behavior of the traffic. The probability distribution of the speed at a certain time enables to incorporate low-probability real-world events that can cause major delays and affect traffic over wide areas. By adding the probability distribution to the computation, the system can compute the probability of arrival time within a certain time frame that can be useful for a more precise route planning. This problem is called *Probabilistic Time-Dependent Routing* (PTDR).

A scalable algorithm for solving the PTDR problem based on Monte Carlo simulations has been presented in [2], [3], and it represents the basis of our work. In particular, this algorithm uses probability distributions of the travel time for the individual graph edges to estimate the distribution of the total travel time and it is integrated with an experimental server-side routing service. This service is deployed on an HPC infrastructure to offer optimal performance for a large number of requests as needed by the smart city context. The PTDR algorithm used in this work simulates a large number of vehicles driving along a determined path in a graph at a particular time of departure. The speed of vehicles on each road is sampled from the speed probability distribution (also called speed profile) associated to each graph edge. The number of samples is a parameter that affects directly the informational value of the output as well as its computational requirements. Given the large amount of requests to be served, even small changes in the workload can affect the overall HPC system efficiency. While the original version of the algorithm [2] was based on a worst-case tuning of the number of samples, and given that a reactive approach [4] is not a viable solution due to the overheads, in this paper we present a proactive method for adapting dynamically the number of samples for the Monte Carlo (MC) simulations based on the PTDR algorithm.

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

- We propose a methodology for self-adapting the PTDR algorithm presented in [2], [3] to the input data in a proactive manner, maximizing its performance, while respecting the output quality level;
- We propose a probabilistic error model used to correlate the input data characteristics with the number of samples used by the Monte Carlo algorithm;
- We adopted an aspect-oriented programming language to keep separated the functional version of the application from the code needed to introduce the adaptivity layer.

The remainder of this paper is organized as follows. Section II provides an overview of the related works, while Section III provides an introduction to the Monte Carlo approach to solve the PTDR problem. Sections IV and V describe the proposed methodology respectively from the adaptivity and code integration point of view. Finally, Section VI describes the experimental campaign to validate the proposed methodology, while Section VII concludes the paper.

II. RELATED WORK

Determining the optimal path in a stochastic time-dependent graph is a well-studied problem which has many formulations [5]. Our approach is close to the *Shortest-path problem with on-time arrival reliability* (SPOTAR) formulation. It can be seen as a variant of the *Stochastic on-time arrival* (SOTA) problem, for which a practical solution exists as shown in [6]. These algorithms have the objective of maximizing the probability of arriving within a time budget and are related to optimal routing in stochastic networks. However, there are not so many solutions for the time-dependent variant of both of the problems. In [5], authors show practical results for the time-dependent variant of SOTA, while in [7] authors elaborate on the complexity of existing theoretical solutions of the SPOTAR problem and show how it can be extended with time dependency. There are many other papers which show various theoretical approaches for the SOTA problem, including some practical applications [6]–[10]. The solution to the SPOTAR problem based on *policy-based* SOTA as a heuristic is presented in [9]. However, the authors make the assumption that the network is time-invariant, which is not true in real cases whether considering long paths. The solution is also unusable in online systems as its scalability to graphs representing real-world routes is not sufficient.

Our approach follows the same philosophy presented in [2], [3], where the authors provide an approximate solution of the time-dependent variant of the SPOTAR problem based on Monte Carlo simulations. As shown in Section III, our approach uses the k-shortest paths algorithm [11]–[13] to determine the paths for which the travel time distribution is estimated. This separation allows us to implement the approach in an online system which provides adaptive routing in real-time. Given the Monte Carlo nature of the algorithm, to improve the efficiency of the PTDR calculation, we have two main alternatives [14]. The first is sampling efficiency, while the second is sampling convergence. In both cases, the algorithm optimization is given by exploiting the iterative nature of the Monte Carlo simulation. To obtain sampling efficiency, several techniques have been proposed to determine what is the next sample to be evaluated to maximize the gathered knowledge [15], [14]. However, in the implementation under analysis, this has been discarded because our goal is to exploit the parallelism of the underlying HPC architecture [3] that excludes any iterative approach to the Monte Carlo. For the same reason, the approaches based on a statistical property evaluation after every iteration [4], checking if the error is acceptable, are not acceptable.

Both approaches would be too time-consuming and, as already analyzed in [2], for the specific problem the number of samples must be chosen a priori in a proactive rather than in a reactive manner.

In this paper, we adopted autotuning techniques to face with the Monte Carlo efficiency problem. Classical autotuning approaches are based on code refactoring and loop parameterization to optimize extra-functional properties (e.g., power and performance) on a given target architecture (e.g., A-tune [16], SPIRAL [17], ATLAS [18], PowerDial [19]). Concerning a Monte Carlo application, in [20] the authors developed a framework to select automatically the optimal parameters for GPU accelerated kernels, such as block size, number of threads and other variables of the CUDA runtime. All those approaches are orthogonal to the proposed work, which is more focused on reducing the number of samples rather than tailoring the computation for a specific platform.

Two interesting works facing the autotuning problem considering the output accuracy (and based on Monte Carlo simulations) are presented in [21] and [22]. These works adopt an ad-hoc DSL language as front-end for the developer, giving to the compilation framework the customization and the related execution. Our approach tries to face the problem from a different perspective. First, we intend to reduce as much as possible the intrusiveness on the target code, thus reducing the load on the application developer. Second, our approach relies more on run-time and data-aware decisions, avoiding compile-time decisions on performance-accuracy trade-offs.

A two-step approach for solving the Monte Carlo problem has been envisioned in [23]. Similar to our work, the authors suggest to have a first step with a reduced number of samples to provide an initial approximate solution as fast as possible, and then to refine the output towards the required accuracy in successive iterations. In the proposed context, this idea suffers from two main problems. First, it is suitable for scientific work-flows, where an intermediate solution is used to trigger the next computations, but this is not our case. Second, in the iterative phase, it suggests a reactive approach rather than a proactive one, that we already discussed to be necessary for the specific PTDR problem. Another two-step approach has been presented in [24]. In this case, the approach suggests to build from each input a *small canary* – a statistically representative subset of the actual input – that is used for a parameters' exploration at runtime. In the specific PTDR case, the approach suffers from a large overhead due to the canary extraction and approximation selection. In the proposed approach, we moved the two operations at design-time by generalizing the approximation selection according to the *unpredictability function* computed at runtime. Moreover, the proposed approach has been designed to guarantee statistically the compliance with an output error level.

Finally, the work in [25] presents a proactive control of the application tuning as a constrained optimization problem, which can be solved at the run-time according to profiling information. The paper introduces a Bayesian model to learn the accuracy of the computation according to data features that are mainly related to the data size. Despite of the similarity of this previous work with one of the components used for the integration (i.e., the mARGOt autotuner [26], [27]), the proposed approach is mainly related to the data-aware error prediction model for PTDR, where SLA constraints are considered, and to the seamless integration of the adaptivity concepts.

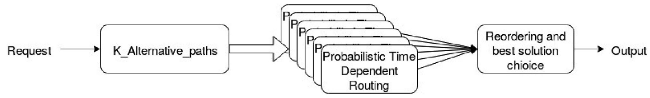


FIGURE 1. Navigation infrastructure for serving a single request.

III. MONTE CARLO APPROACH FOR PROBABILISTIC TIME-DEPENDENT ROUTING

So far, many theoretical formulations and several algorithms have been developed for solving the problem of computing the travel time distribution [5]. In this paper, we consider a *path-based* approach (*SPOTAR*), where the paths are known *a-priori* and travel-time distributions are determined subsequently for each path [28].

In the context of the traffic navigator application shown in Figure 1, our focus is on the efficient estimation of the arrival time distribution (*PTDR - Probabilistic Time-Dependent Routing* phase). More in detail, the three main steps of the application can be described as follows:

(i) The first step consists of selecting *K alternative paths* to be passed to the next steps. In the navigation scenario, the identification of the shortest path is not enough to determine a good solution, if no traffic information have been considered. Thus alternative routes, derived by the *k-Short Paths* algorithms with limited overlap, must be adopted in this step [11]–[13]. This first phase is out of the scope of this paper;

(ii) For each path selected by the previous step, the computation of the travel time is done by using the Probabilistic Time-Dependent Routing module. While the exact solution to the travel-time estimation (PTDR) has an exponential complexity, in this work we approximate efficiently the solution of the SPOTAR problem by adopting a Monte Carlo sampling approach [2];

(iii) The final step gathers timing information provided by the k instances of the PTDR module for each single request and selects the best path for the user. Actually, this phase does not provide a single route, but reorders the list of k paths according to the user preference and to the timing distributions determined in the previous phase [29].

This three-step approach of the navigation application enable us to obtain an approximate solution to the SPOTAR problem, which can be used as an online system to serve a large volume of routing requests.

Our definition of a probabilistic road network is similar to the definition of the stochastic time-dependent network as described by Miller-Hooks [28], with the exception of the segment travel times, which has been substituted by the speed probability distribution (*speed profile*) for a given time of departure within a week. Formally, let $G = (V, E)$ be a well-connected, directed and weighted graph, where V is the set of vertices and E is the set of edges. Each vertex represents a junction or some important point corresponding to geospatial properties of the road, while edges represent the individual road segments. Each path selected by the first phase of the application (i.e., K-Alternative paths) can be formally represented as a vector of graph edges $S = (s_1, s_2, \ldots, s_n)$, while $S_p \subseteq E$ and n is the number of road segments in the path.

Using a travel time estimation function, we are interested in estimating the travel time θ as $\hat{\theta}_{S,t,P_S}$ where S is the given path, t is the departure time and P_S are the probabilistic speed profiles for the segments in S. More in detail, $t \in T$ is a departure time within a set of possible discrete departure times $T = \{t : t = n \cdot \phi, n \in \mathbb{N}\}$ [30], where the length of the interval ϕ is determined by input data. P is the set of probabilistic speed profiles for the entire graph edges E, where $P_S \subseteq P$. Each speed profile $p \in P$ is represented by a set of discrete speed values and assigned probabilities. The number of speed values depends on the method used for deriving the profiles from historical traffic monitoring data, while the minimum and maximum values represent respectively the congestion speed and the free flow speed. In this work, the discrete time interval and the number of speed values have been set respectively to $\phi = 900$ s (15 minutes) and to 4 speed levels according to the characteristics of the available input data.

Focusing on the SPOTAR problem, we are not interested in a single travel time value θ, but we have to calculate the probability distribution of the arrival time. Given the previous formalization of the problem, the travel time distribution can be estimated by traversing the path segments together, while considering the speed profile distribution. In particular, we can define a tree, where each layer represents a segment in the selected path [2]. The tree root is the starting segment, while the end segment is on the leaves. Each node in all layers of the tree has l children, where l is a number of the discrete speed values for each segment, and the tree depth corresponds to the number of selected path segments $|S|$. Each edge in the tree is annotated by the discrete speed value, its probability, and by the length of the considered segment. A travel time can be computed by a depth-first search (DFS), while selecting an arbitrary child node at each level of the tree. The travel time value is then the sum of the time spent in each segment (length/speed), while the probability of that value is the product of the probability on each edge of the traversal. Each traversal corresponds to a single car traveling along the entire path. The exact solution is obtained by an exhaustive search over all the possible paths between the root node and all the leaves. This approach is clearly not efficient, because it scales exponentially with the number of segments in the path.

A Monte Carlo-based approach can be successfully employed in this case. By generating a large number of random tree traversals, enough samples can be obtained to estimate the final distribution. We define this final distribution,

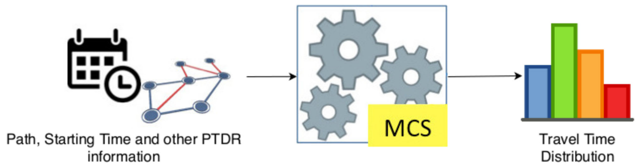


FIGURE 2. The original approach for probabilistic time-dependent routing based on Monte Carlo simulations to derive the travel time distribution.

which is a collection of θ values $(\theta_1 \ldots \theta_x)$ obtained through the Monte Carlo simulation $MCS(x, i)$, where x is the number of random tree traversals, and i is the input set of the $\hat{\theta}$ function (i.e., S, t, P_S).

Given that travel times usually have a long tailed distribution due to inherent properties of the traffic (e.g., rare events such as accidents), a large number of samples is needed to estimate the travel time distribution with a sufficient level of precision. Regarding the definition of the number of samples for the Monte Carlo simulation, the specific implementation of the PTDR kernel cannot rely on a run-time stability analysis of the output. Each tree traversal (a sample of the Monte Carlo simulation) is totally independent to the others, thus this problem is perfectly suitable for parallel computing architectures, such as modern CPUs or accelerators. To exploit efficiently the parallelism, it is necessary to know a-priori the number of travel time estimations required to build the final distribution.

To summarize, the PTDR algorithm can be seen as in Figure 2, where all the information regarding the request are provided to the Monte Carlo simulation (MCS) to get the predicted travel time distribution for the given route.

IV. THE PROPOSED APPROACH

The Monte Carlo simulation is designed to use a given number of samples x for every run. Based on a conventional approach, this number is selected according to the worst-case analysis and it is given by the lowest number of samples to reach the target precision [3]. In this section, we present the proposed technique to select at runtime the number of samples for the Monte Carlo simulation according to the input data characteristics.

Before moving to the methodological part, let us better contextualize the problem. Even if we are interested in the travel time distribution, our goal is to know a value τ_i to guarantee, with a certain probability, that the travel time will be within that value: $P(\theta < \tau_i) \geq y$ where i has been defined as the input set of the travel-time function. The value τ_i is the output of the PTDR phase. In the following, we characterize that value with an additional property $\tau_{i,y}$, where y is the probability that the travel time will be lower than τ.

Based on Monte Carlo simulation, we can estimate the value of $\tau_{i,y}$ by using x samples as follows: $\hat{\tau}^x_{i,y} = MCS(x, i, y)$. We estimate the value $\hat{\tau}^x_{i,y}$ by selecting the yth percentile of the finite-sample distribution obtained from the Monte Carlo simulation (i.e., if $y = 95\%$ then $\hat{\tau}^x_{i,y}$ is the 95th percentile of the distribution).

In the context of this work, we are interested in minimizing the execution time of the function *MCS*, while limiting the prediction error defined as $error^x_{i,y} = \frac{|\tau_{i,y} - \hat{\tau}^x_{i,y}|}{\tau_{i,y}}$. The target problem can be expressed as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & cpu\_time^x_i \\ \text{subject to} \quad & error^x_{i,y} \leq \epsilon, \end{aligned} \tag{1}$$

where ϵ represents the upper bound on the computation error. We want this error be relative to the output of the MCS, which is the desired percentile of the predicted travel time. In this way, we can abstract from the actual path. Given the tight correlation between the execution time and the used number of samples x, the previous problem can be simplified by considering the minimization of x instead of the *cpu_time*. According to the Monte Carlo's properties [31], we can derive that $\tau_{i,y} \equiv \hat{\tau}^{\infty}_{i,y}$, where $\hat{\tau}^{\infty}_{i,y}$ is the output of the *MCS* function computed by using an infinite number of samples. Thus, we can reformulate the error as

$$error^x_{i,y} = \frac{|\hat{\tau}^{\infty}_{i,y} - \hat{\tau}^x_{i,y}|}{\hat{\tau}^{\infty}_{i,y}}. \tag{2}$$

Moreover, the value $\hat{\tau}^x_{i,y}$ is a random variable, asymptotically normally distributed with mean $\mu_{\hat{\tau}^x_{i,y}}$ and standard deviation $\sigma_{\hat{\tau}^x_{i,y}}$. In particular, according to the central limit theorem [32], the mean value does not depend on the number of Monte Carlo simulations, and the standard deviation decreases by increasing the number of samples. Given that, we can define the error as characterized by a normal distribution with mean 0 and a standard deviation $\sigma_{\hat{\tau}^x_{i,y}}/\mu_{\hat{\tau}^x_{i,y}}$. In the following, we refer to the standard deviation of the error as $\nu_{\hat{\tau}^x_{i,y}}$. This expression is the same as the coefficient of variation (relative standard deviation) of the result of the Monte Carlo simulation.

According to the probabilistic nature of the problem, we cannot guarantee that the error will be always less than ϵ. However, this can be done by relaxing the error constraint by introducing a confidence interval (CI) level. Given the normal distribution of the error, the selected confidence interval can be correlated with the expected error

$$P(error^x_{i,y} \leq \epsilon) \geq CI \Longrightarrow \hat{error}^x_{i,y} \leq n(CI) \times \nu_{\hat{\tau}^x_{i,y}} \leq \epsilon, \tag{3}$$

where $n(CI)$ is a value that express the confidence level (e.g., n(68%) = 1, n(95%) = 2 and n(99.7%) = 3 derived from the 1-3 σ-intervals of the normal distribution). Thus, if we decrease the number of Monte Carlo simulations used to derive $\hat{\tau}^x_{i,y}$, we decrease the execution time of the application, but we reduce the accuracy of the results, with a larger value for the coefficient of variation $\nu_{\hat{\tau}^x_{i,y}}$.

An additional problem is given by the fact that $\hat{\tau}^x_{i,y}$ is input-dependent. Therefore it is not possible to predict the possible Monte Carlo error for unknown paths, according to the number of samples. To deal with this problem, we found a feature u_i of the inputs i that can be used to quickly estimate the number of samples necessary to keep the error below the threshold ϵ. The idea is to evaluate the error by using u_i instead of the actual i, so that we can transform the original problem as

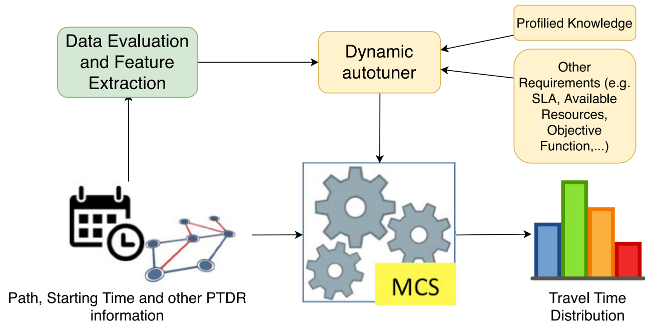


FIGURE 3. **Proposed adaptive approach for PTDR routing based on Monte Carlo simulations and dynamic choice of number of simulations.**

$$error^x_{i,y} \leq n(CI) \times \nu_{\hat{\tau}^x_{u_i,y}}. \tag{4}$$

The feature u_i has been called *unpredictability*, since it represents a set of input characteristics i (e.g., road and starting time) that provides information about how complex is the prediction of $\tau_{i,y}$. Therefore it is related on how many samples are required to satisfy a certain error and confidence level. More details on the unpredictability feature are presented in Section IV-A.

Given that the error is no more related to the specific input set i, but only to the feature u_i, the number of samples needed to satisfy the constraint can be easily extracted by $\nu_{\hat{\tau}^x_{u_i,y}} \leq \frac{\epsilon}{n(CI)}$. A profiling phase on a set of representative inputs can be used to extract the values of $\hat{\nu}_{\hat{\tau}^x_{u_i,y}}$, that will be used to determine the correlation between the unpredictability function and the error. More details on the profiling phase including the prediction function are presented in Section IV-B.

To summarize, the proposed methodology adds an adaptive layer on top of the Monte Carlo simulation (see Figure 3) to determine at runtime the right number of samples for each request satisfying the required accuracy. A feature-extraction procedure estimates the unpredictability value from the input data of the request (path, starting time and segment speed-profiles). The dynamic autotuner combines this data feature with the profiled knowledge and the extra-functional requirements to configure the Monte Carlo simulation.

A. UNPREDICTABILITY FEATURE

Given that the extraction of the data feature from the inputs should be done at runtime, its computation should not be a costly operation. Otherwise, the benefit of speeding up the computation phase by reducing the number of Monte Carlo samples would be reduced by the data feature extraction overhead, eventually making the whole approach meaningless. From the experimental data, we have found that a measure of the unpredictability of the path can be extracted by a simple statistical property of the set of travel times θ extracted by a quick Monte Carlo simulation: the coefficient of variation.

Intuitively, the more the results are spread out, the more the route is hard to predict. Therefore, to get a precise estimation of the distribution, and in particular the percentiles, we need a higher number of samples.

The unpredictability function is defined as $u_i = \sigma^x_{\theta_i} / \mu^x_{\theta_i}$ where σ_{θ_i} and μ_{θ_i} are evaluated on a MCS done with the minimum number x of samples allowed at runtime. It is important to note that σ_{θ_i} is the variance of the travel times extracted by a single Monte Carlo simulation on the minimum number of samples. We calculate the unpredictability function together with the first set of Monte Carlo samples to further reduce the overhead introduced by the data feature extraction. In particular, we will use this first short Monte Carlo run to determine if there is a need of further samples (and how many) to satisfy the error constraint.

To validate the usage of u instead of i, we used the Spearman correlation test [33] between the unpredictability value and the value of $\nu_{\hat{\tau}^x_{i,y}}$ used in the calculation of the expected error for different values of x and y over a wide range of inputs sets i. In all cases, the correlation values were larger than 0.918 showing a p-value almost 0. These correlations confirm our hypothesis and the p-values prove that the results are statistically significant.

B. ERROR PREDICTION FUNCTION

To predict the expected error for a specific configuration according to the data feature u, we need to extract $\hat{\nu}_{\hat{\tau}^x_{u_i,y}}$ from profiling data. We run the Monte Carlo simulation several times for each configuration in terms of number of samples. In particular, we decided to use values ranging from 100 samples up to 3,000. The two numbers are derived from the observation that 100 samples are the minimum to estimate the percentile for the distribution, while 3,000 is the number of samples found to be good enough to satisfy the worst case conditions in the previous work [34]. In this range of values, we also selected 300 and 1,000 samples by considering that the Monte Carlo error decreases as $1/\sqrt{n}$ [35]. Thus in our case at each sampling level, we have that the error is almost halved.

We run each set of Monte Carlo simulation with the same configuration in terms of the number of samples on a large set of inputs i, and we extract $\nu_{\hat{\tau}^x_{u_i,y}}$ and u_i from each configuration. Then we create a predictor $\hat{\nu}_{\hat{\tau}^x_{u_i,y}}$ as the *quantile regression* [36] over the extracted data. The use of quantile regression improve the robustness of the model in the context of its use. Indeed, we are not interested in predicting an average value as final result, but we want to use it for the inequality formula $\hat{\nu}_{\tau^x_{u_i,y}} \leq \frac{\epsilon}{n(CI)}$. In this case, a higher value of the quantile with respect to 50th (the purely linear regression), guarantees a higher robustness in satisfying the previous inequality. The quantile used for the regression is an additional parameter that can be explored to trade-off robustness and performance.

V. INTEGRATION FLOW

While the previous section introduced the proposed methodology from the end-user perspective, thus considering the execution time and the elaboration error, this section focuses on the application-developer perspective by presenting the integration flow proposed to enhance the target application with limited effort. The proposed integration flow enforces a separation of concerns between the functional and extra-functional properties by using an Aspect-Oriented Programming Language to inject the code to introduce the adaptivity layer in the target source code.

On one side, we used the mARGOt framework [26], [27] to tune dynamically the application, thus implementing the adaptivity concepts presented in Section IV and highlighted in Figure 3. mARGOt is an open-source dynamic autotuning library that enhances an application with an adaptation layer. In particular, it provides to the application the most suitable configuration of the software knobs, according to the application requirements and to the execution environment evolution. The configuration selection is done by relying on application knowledge gathered at profile time. In this context, we used mARGOt to select the number of samples to minimize the execution time, while keeping the error below a certain threshold. In particular, the selection is done by considering the unpredictability value of the current path, and using as application knowledge the design-time model described in Section IV-B.

On the other side, we have hidden to the application developer the code manipulation complexity by using LARA [37] as a language to describe user-defined strategies, and its Clava source-to-source compiler[1] for code analysis and transformation. LARA is a Domain Specific Language inspired by the Aspect-Oriented Programming concepts. It enables the user to capture specific points in the code based on structural and semantic information, and then analyze and act on those points. This generates a new version of the application, leaving the original unchanged and separating the main functional concerns from those specified in LARA. Clava is a C/C++ source-to-source compiler, where the code analysis and transformation steps are guided by scripts written in the LARA language. In this work, we used LARA and Clava for two main tasks. First, to instrument the original source code with the autotuner glue code. Second, to configure the autotuner library according to the application requirements.

Figure 4 outlines the integration framework, highlighting the two main LARA aspects used (Listings 1 and 2) and the automatic generation process from the original application (Listing 3) to the final enriched code (Listing 4).

The code in Listing 1 shows the aspect needed to configure mARGOt, generating an autotuning library tailored according to the application requirements. This represents the *mARGOt Configuration aspect* shown in Figure 4. At lines 9–16, we define the *num_samples* tunable software knob, the *unpredictability* feature that we want to observe, the *error* metric and the goal (i.e., the Service Level Agreement, *error* $< 3\%$). Once the knobs, metrics and data features are defined, we can proceed with the generation of the multi-objective constrained optimization problem (lines 19–21),

[1]Project repository: https://github.com/specs-feup/clava

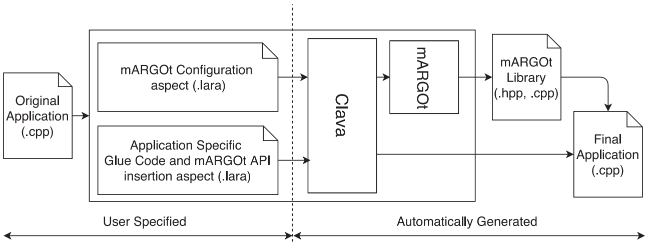


FIGURE 4. Integration flow outlining the two main LARA aspects and related actions: Original code enrichment and autotuner configuration.

called states in mARGOt (line 19). Lines 20–21 define respectively the objective function and the constraint for the target problem. Finally, line 24 builds the LARA internal structure *margotCodeGen_ptdrMC* that is then used to generate the mARGOt configuration file and its code generator.

The second aspect (shown in Listing 2) integrates the proposed methodology in the target application. This is the *Application Specific Glue Code and mARGOt API insertion aspect* shown in Figure 4. It takes as input (line 3) the mARGOt code generator given by the previous aspect (Listing 1) and the number of samples needed to evaluate the unpredictability feature. At line 6, we query the original code (shown in Listing 3) to identify the statement (*stmt*) including the Monte Carlo function call (*mc.RunMonteCarloSimulation*) as target join point to be manipulated. Lines 7–17 contain the actual manipulation actions done on the selected join point *stmt* of the target code. There are two different types of operations. First, to integrate the mARGOt calls for initializing the library and updating the software knob (Lines 10 and 14). Second, to *insert* the glue code (LARA *codedef*) for calculating the unpredictability (line 12 and lines 21–25), and to *replace* the original Monte Carlo call with the optimized one that does not repeat the unpredictability samples (line 16 and lines 28–31).

```
1  aspectdef McConfig
2    /* Generated Code Structure*/
3    output codegen end
4
5    /* mARGOt configuration */
6    var config = new MargotConfig();
7    var ptdrMC = config.newBlock('ptdrMC');
8
9    /* knobs */
10   ptdrMC.addKnob('num_samples','samples','int');
11   /* data features */
12   ptdrMC.addDataFeature('unpredictability','float','>=');
13   /* metrics */
14   ptdrMC.addMetric('error','float');
15   /* goals */
16   ptdrMC.addMetricGoal('my_error_goal','<=', 0.03, 'error');
17
18   /* optimization problem */
19   var problem = ptdrMC.newState('problem');
20   problem.minimizeKnob('num_samples');
21   problem.subjectTo('my_error_goal');
22
23   /* creation of the mARGOT code generator for the following code
         enhancement (McCodegen aspect) */
24   margotCodeGen_ptdrMC = MargotCodeGen.fromConfig(config, '
         ptdrMC');
25 end
```

LISTING 1. LARA aspect for configuring the mARGOt autotuner.

```
1  aspectdef McCodegen
2    /* Target function, mARGOt code generator from McConfig aspect, #
         samples for feature extraction */
3    input margotCodeGen_ptdrMC, unpredictabilitySamples end
4
5    /* Target function call identification */
6    select stmt.call{'mc.RunMonteCarloSimulation'} end
7    apply
8      /* Target Code Manipulation */
9      /* Add mARGOt Init*/
10     margotCodeGen_ptdrMC.init($stmt);
11     /* add unpredictability code */
12     $stmt.insert before UnpredictabilityCode(unpredictabilitySamples);
13     /* Add mARGOt Update */
14     margotCodeGen_ptdrMC.update($stmt);
15     /* Add Optimized Call Code */
16     $stmt.insert replace OptimizedCall(unpredictabilitySamples);
17   end
18 end
19
20 /* Unpredictability extraction code */
21 codedef UnpredictabilityCode(unpredictabilitySamples) %{
22   auto travel_times_feat = mc.RunMonteCarloSimulation([[
         unpredictabilitySamples]], startTime);
23   ResultStats feat_stats(travel_times_feat, {});
24   float unpredictability = feat_stats.variationCoeff;
25 }% end
26
27 /* Optimized MonteCarlo call */
28 codedef OptimizedCall(unpredictabilitySamples) %{
29   auto run_result = mc.RunMonteCarloSimulation(samples − [[
         unpredictabilitySamples]], startTime);
30   run_result.insert(run_result.end(), travel_times_feat.begin(),
         travel_times_feat.end());
31 }% end
```

LISTING 2. LARA aspect for inserting the application-specific glue code (unpredictability extraction) and the required mARGOt calls.

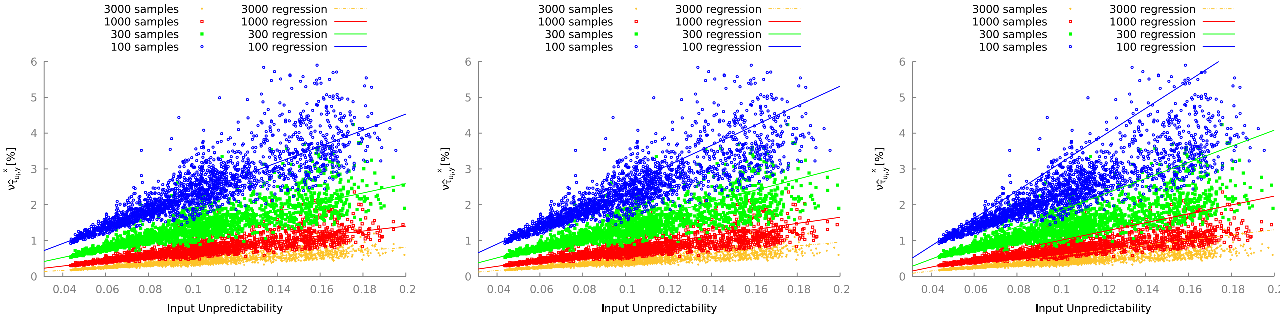
```
/* Load data */
Routing::MCSimulation mc(edgesPath, profilePath);
auto run_result = mc.RunMonteCarloSimulation(samples, startTime);
ResultStats stats(run_result);
Routing::Data::WriteResultSingle(run_result, outputFile);
return 0;
```

LISTING 3. Original source code before integrating the adaptivity layer.

```
/* Load data */
Routing::MCSimulation mc(edgesPath, profilePath);
/* BEGIN LARA INSERT codedef UnpredictabilityCode */
auto travel_times_feat = mc.RunMonteCarloSimulation(100, startTime);
ResultStats feat_stats(travel_times_feat, {});
float unpredictability = feat_stats.variationCoeff;
/* END LARA INSERT codedef UnpredictabilityCode */
/* BEGIN LARA margotCodeGen_ptdrMC.update */
if(margot::ptdrMC::update(samples, unpredictability)) {
  margot::ptdrMC::manager.configuration_applied();
}
/* END LARA margotCodeGen_ptdrMC.update */
/* BEGIN LARA REPLACE codedef OptimizedCall */
auto run_result = mc.RunMonteCarloSimulation(samples − 100, startTime);
run_result.insert(run_result.end(), travel_times_feat.begin(),
      travel_times_feat.end());
/* END LARA REPLACE codedef OptimizedCall */
ResultStats stats(run_result);
Routing::Data::WriteResultSingle(travel_times_new, outputFile);
return 0;
```

LISTING 4. Target source code after the integration of the adaptivity layer.

Listing 4 shows the outcome of the integration flow after all the code manipulation steps, as highlighted in Figure 4. The code inserted or replaced with LARA is marked with *BEGIN-END* comments.



(a) Quantile regression using the 50^{th} perc. (b) Quantile regression using the 75^{th} perc. (c) Quantile regression using the 95^{th} perc.

FIGURE 5. Training of the error model by using different number of samples and quantile regressions.

Overall, in this instance of integration, we used 53 lines of LARA to generate 221 lines of C++ code. However, the advantage does not only consists in saving lines of code ($>$ 4x). There are three main reasons to justify this approach. First, the user does not need to know the details about the mARGOt configuration files and low-level C++ API, but he can instead focus on the high-level interface available in LARA that is more declarative on the target problem (as shown in Listings 1 and 2). Second, the proposed approach reuses information through different steps of integration. There is mARGOt-specific information that should be provided to the user at several steps, such as in the configuration files and when using the autotuning API (e.g., the name of the autotuner block and the knobs and data features). By using the high-level LARA aspects, the user defines this information only once, thus saving time and possibly introducing fewer errors. Third, the proposed approach leverages on the separation of concerns between the original code (functional description) and the autotuning code (extra-functional optimizations). Therefore the extra-functional optimizations, including problem definition (optimization targets and constraints), are kept separated and the user does not have to modify the original source code. In this way, the original code developer does not need to be involved in the optimization process, even enabling the functional development and extra-functional optimization to run in parallel.

VI. EXPERIMENTAL RESULTS

In this section, we show the results of applying the proposed methodology to the PTDR algorithm. The platform used for the experiments is composed of several nodes based an the Intel Xeon E5-2630 V3 CPUs (@2.8 GHz) with 128 GB of DDR4 memory (@1866 MHz) on a dual channel memory configuration. In Section VI-A, we show the results of the model training for estimating the expected error. Then, in Section VI-B, we validate the approach by verifying the error constraint ϵ. In Section VI-C, we compare the proposed approach with respect to the original version taking a static decision on the number of samples, while in Section VI-D, we discuss the overhead introduced. Finally, in Section VI-E, we evaluate the optimization impact when considering the navigation service at system-level.

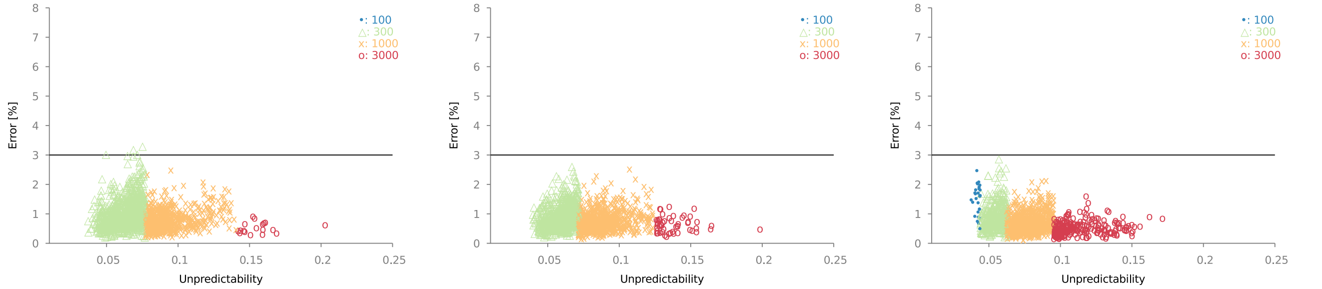
A. TRAINING THE MODEL

The first phase of the methodology is done off-line and consists of training the error model ($err\hat{o}r^x_{i,y}$) presented in Section IV-B by using a different number of samples. For training the quantile regression, we used profiling data extracted by running the PTDR algorithm on a training set. This training dataset has been built by using random requests done on 300 different paths across the Czech Republic in different time-slots, thus considering different speed profiles for each segment of the paths. All these requests have been done for all the 4 levels of sampling used in this paper (i.e., 100, 300, 1000 and 3000, as described in Section IV-B). The output of the model training is shown in Figure 5. The points in the three plots represent the results obtained from the profiling runs. The lines represent the quantile regression lines, thus the model that will be used at runtime. The three subfigures are different in terms of the quantile value used for the regression. Figures 5(a), 5(b) and 5(c) represents the regression results by using respectively the 50th, 75th and 95th quantile. The three regressions are slightly different because we pass from a more permissive one in Figure 5(a), where almost half of the points are below the corresponding regression lines, to the most conservative one in Figure 5(c), where only a few points are above. We can see that the coefficients of the lines of the quantile regression are almost doubled passing from 75th to 95th percentile (e.g., for 100 samples, the coefficients pass from 0.27 to 0.38, while for 3,000 samples they pass from 0.049 to 0.071).

The extracted models are now ready to be used at runtime by the dynamic autotuner to select the minimum number of samples that satisfy the error constraint for the given unpredictability value. The results shown in Section VI-B will demonstrate the effectiveness of the proposed method at runtime.

B. VALIDATION RESULTS

The set of validation results presented in this section are reported to demonstrate how the dynamic tuning of the number of samples satisfies the error constraints. We randomly generated 1,500 requests for the enhanced PTDR module for routes on the Czech Republic at different starting times. These requests are different from the ones used in the training phase of the model. We validated the approach by using



(a) Quantile regression using the 50^{th} perc. (b) Quantile regression using the 75^{th} perc. (c) Quantile regression using the 95^{th} perc.

FIGURE 6. Validation of the proposed approach by using 3 percent as target error and different percentiles for the quantile regression.

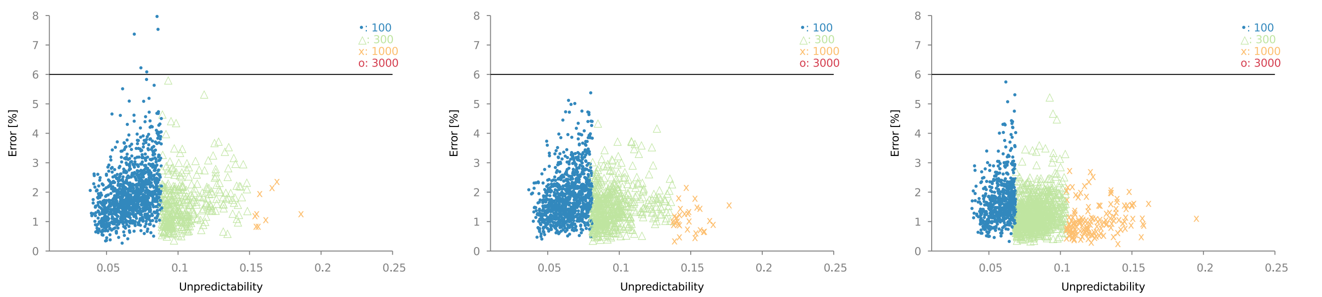
three different quantile regressions (on 50th, 75th and 95th quantile), two different target errors ϵ (3 and 6 percent) and a confidence interval (CI) for the error constraint equal to 99 percent (i.e., $n(99\%) = 3$). The error is derived by considering a run of Monte Carlo simulation on the same input set by using 1 million of samples, enough to be considered a good estimation of the actual travel time distribution. Then, we selected as error the maximum among different key percentiles: 5th, 10th, 25th, 50th, 75th, 90th and 95th percentile.

The results are reported in Figures 6 and 7 respectively for an error constraint ϵ equal to 3 and 6 percent. The two figures show the error results for each run with respect to the unpredictability feature extracted on the path. Each dot in the plots represents a PTDR request, while its shape depends on the number of samples used for the Monte Carlo simulation. In most of the cases, the actual error is below the target error. As it was expected by considering the same value of the error constraint ϵ, the more conservative is the quantile regression, the less are the points that violate the constraint error. For the data we processed, the number of times that the error constraint is not respected is within the selected CI (99 percent). At the same time, moving from a less conservative quantile regression (e.g., 50th percentile) towards a more conservative one (e.g., 95th percentile), it is possible to note how the threshold values for selecting the same number of samples shift to the left. By considering an error constraint $\epsilon = 3\%$ (see Figure 6), the maximum unpredictability value for 300 samples moves from 0.075 to less than 0.06 respectively when considering the quantile regression from the 50th percentile, up to the 95th quantile. Similar is the case when we consider an error constraint $\epsilon = 6\%$ (see Figure 7), where the same threshold moves from an unpredictability of 0.15 to 0.14 and 0.11 when using the 50th, the 75th and the 95th as quantile value for the regression. Finally, we can notice the difference in terms of number of samples between the two cases with different ϵ values. Indeed, while for ϵ equal to 3 percent (Figure 6) only a very small fraction of the cases use 100 samples and there is a not negligible fraction of cases where 3,000 samples are employed. For ϵ equal to 6 percent (Figure 7), in some cases only 100 samples are required.

C. COMPARISON RESULTS WITH STATIC APPROACH

In this section, we demonstrate the advantages of the proposed approach with respect to the baseline version [2], where the number of samples is defined *a priori*. To provide a fair comparison, we extracted the number of samples to be used for the baseline version by using the training dataset.

For the 4 levels of sampling used in this paper (i.e., 100, 300, 1000 and 3000, as described in Section IV-B), we analyzed the cumulative distributions of the expected error (see Figure 8). We selected the minimum sampling level that passes a certain threshold of the cumulative value before reaching the error constraint value ϵ. This threshold value has almost the same *robustness* meaning of the quantile regression value used in our approach. In the following, we compare the proposed approach, where the quantile regression model has been built over a certain percentile, with a static tuned version, where the same percentile is used as threshold for the cumulative. If we use for the proposed approach the quantile regression at 95 percent, we compare



(a) Quantile regression using the 50^{th} perc. (b) Quantile regression using the 75^{th} perc. (c) Quantile regression using the 95^{th} perc.

FIGURE 7. Validation of the proposed approach by using 6 percent as target error and different percentiles for the quantile regression.

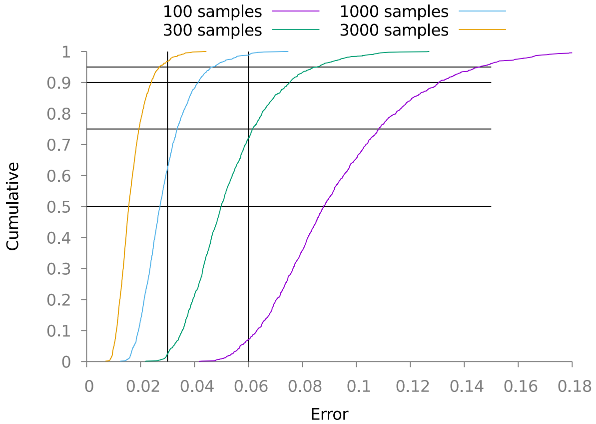


FIGURE 8. Cumulative distribution of the error by varying the number of samples over the training set.

with the statically tuned version, where the number of samples is defined by looking at the cumulative curve that reaches at least 95 percent before to the target error constraint. In Figure 8, we can notice that for an error constraint $\epsilon = 6\%$ the static tuning is set to 1,000 samples for the percentile interval between 72th and 98th. For values larger than 98th and smaller than 72th percentile (down to 7th) we have to consider respectively the configuration using 3,000 and 300 samples. On the other side, for $\epsilon = 3\%$ we select 3,000 samples within the percentile interval 72th-97th, 1,000 samples for percentile values smaller than 72th (down to 5th), while we need more than 3,000 samples if the request is very tight on a percentile larger than 97th.

Table 1 shows the comparison results obtained by using the proposed adaptive technique with respect to the original version (*baseline*) with the statically defined number of samples obtained with the previously described analysis. In particular, Table 1 presents the average number of samples and gain with respect to the baseline for different values of error constraint ϵ and different percentiles used to build the predictive model and for the static tuning of the baseline. The results are obtained by a large experimental campaign over randomly selected pairs of Czech Republic routes and starting times, different from those used for the training. While the routes have been randomly selected, we used a more realistic distribution of the starting time emulating the actual road usage [38], [39].

In the considered cases, the proposed approach reduces the number of samples by at least the 36 percent, and up to the 81 percent. As expected, the average number of samples for the proposed approach is smaller when we relaxed either the error constraint (i.e., 6 percent) or the percentile used for building the model (e.g., 50th percentile). The lower gain for the configurations using the 50th percentile with respect to the cases using the 75th-95th percentile is due to the fact that in the former case the baseline requires a smaller number of samples with respect to the latter case (i.e., 1,000 versus 3,000 for $\epsilon = 3\%$ and 300 versus 1,000 for $\epsilon = 6\%$). Looking at absolute numbers, it is possible to detect that even if the percentage gain seems to be higher with more conservative regressions

TABLE 1. Average number of samples for the validation set using different quantile regression values (columns) and different error constraints

		Average Number of Samples		
ϵ		50th perc.	75th perc.	95th perc.
3%	baseline	1000	3000	3000
	adaptive	632 (−36%)	754 (−74%)	1131 (−62%)
6%	baseline	300	1000	1000
	adaptive	153 (−49%)	186 (−81%)	283 (−71%)

The results are reported for the baseline *and proposed* adaptive *versions.*

(75th-95th), the actual average number of samples used is smaller with the more permissive quantile (50th).

The reduction in terms of the number of samples is directly reflected on the execution time because there is a linear dependency. We observed an execution time speedup between 1.5x and 5.1x. A more detailed analysis on the overhead is presented in Section VI-D.

To further analyze the benefits of the proposed methodology, Figure 9 shows the number of samples selected by the adaptive Monte Carlo simulation, when the same request in terms of target path is done every 15 minutes during the entire week. The temporal interval is derived from the smallest time granularity (ϕ) of the database containing the speed profiles. The two plots (a) and (b) have been generated by using the 3 and 6 percent respectively as maximum target error and for both experiments a quantile regression on the 75th percentile.

By looking at the number of samples requested by the adaptive version of the Monte Carlo simulation, we can easily recognize well-known traffic behaviors in both plots. The daily distribution on the weekdays is characterized by two main peaks determined by less predictable situations. The first peak is around the 7–8 am timeslot, while the second one is around the 4–5 pm. During the weekend, the morning peak is a bit postponed, while the afternoon peak almost disappears. It is also clearly visible how the evening hours are the most predictable ones.

The dynamic behavior captured by the enhanced version of the algorithm cannot be exploited by using the original (baseline) version. Following the same philosophy adopted in Table 1, the original version must be tuned by considering 3,000 samples for the experiment in Figure 9(a) ($\epsilon = 3\%$) and 1,000 samples for the experiment in Figure 9(b) ($\epsilon = 6\%$). In both cases, the static tuning requires a higher number of samples than the proposed techniques, which instead requires such a large number only when strictly required (traffic peaks). Also considering the static tuning to the *average* case (i.e., 1,000 samples for the experiment in Figure 9(a) and 300 samples for the experiment in Figure 9(b)) is not a viable solution. This is because there are still a lot of sampling reduction possibilities in predictable moments that are not captured, and more importantly, the travel time predictions are not able to satisfy the algorithm output quality during the most unpredictable periods. Finally, a fixed time-slot policy is sub-optimal, given that

(a) Error constraint ϵ=3%

(b) Error constraint ϵ=6%

FIGURE 9. Number of samples selected by the proposed adaptive method when the same request is performed every 15 minutes during the week.

the unpredictability strongly depends not only on the time of the request, but also on the path characteristics (e.g., urban or countryside path, close or far from the congested areas) and length (e.g., when it is expected the arrival in a congested area).

D. OVERHEAD ANALYSIS

Even if we already described in Section V how we reduced the integration overhead from the application developer point of view, this section explains the time-overhead introduced to obtain the proposed adaptivity. In particular, the additional computations that we add are related to the calculation of the $\nu_{\tilde{\tau}^x_{i,y}}$ and to the autotuner calls used to determine the right number of samples to be used. The initial 100 Monte Carlo samples, required to extract the data feature, are not part of the overhead given that they are reused (and thus discounted) to calculate the expected travel time (see Listing 4).

Figure 10 shows the overhead introduced by the proposed methodology compared to a set of Monte Carlo calculation by using a different number of samples (from 100 to 300, and 1 M) over a set of paths among different locations in the three main cities of the Czech Republic. As expected, it is evident that the execution time is strictly correlated to the number of samples used for the travel time computation. The different paths used are in a range between 300 and 800 segments. When we fixed the number of samples, the different number of segments are the main reasons of variability for the computing time of the Monte Carlo simulation.

Even if needed for every request, the overhead introduced by our approach is almost negligible, being more than two orders of magnitude smaller than the smaller Monte Carlo simulation with 100 samples. In particular, the execution times of the data feature extraction and mARGOt calls are comparable to the evaluation of a single sample of the Monte Carlo run on a path composed of 200 road segments.

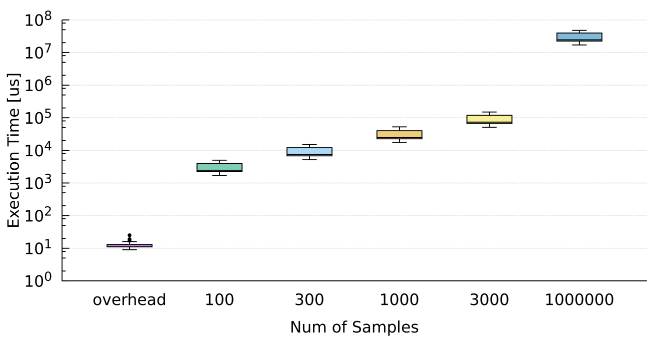


FIGURE 10. Execution time overhead due to the additional code for the proposed method with respect to the target Monte Carlo simulation by varying the number of samples.

E. SYSTEM-LEVEL PERFORMANCE EVALUATION

To quantify at system-level the effects of the proposed adaptive method, in this section we present an analysis done when considering that the efficient PTDR module is included in the full navigation pipeline shown in Figure 1. We built a performance model of the navigation pipeline by using the simulation environment Java Modeling Tools (JMT) [40]. JMT is an integrated environment for workload characterization and performance evaluation based on queuing models [41]. It can be used for capacity planning model simulation, workload characterization and automatic identification of bottlenecks. In particular, to build the simulation model of the queuing network, we considered one station for each of the modules composing the navigation pipeline and we added a fork-join unit to model the parallel PTDR evaluations of each alternative path found in the first stage.

The model, shown in Figure 11, was annotated with values derived by the profiling of each module (K-Alternative path, PTDR, and reordering) and considering a value for K (the number of alternative paths) equal to 10. Moreover, we made a resource allocation according to a load produced by up to 100 K cars producing a request every 2 minutes. The latter

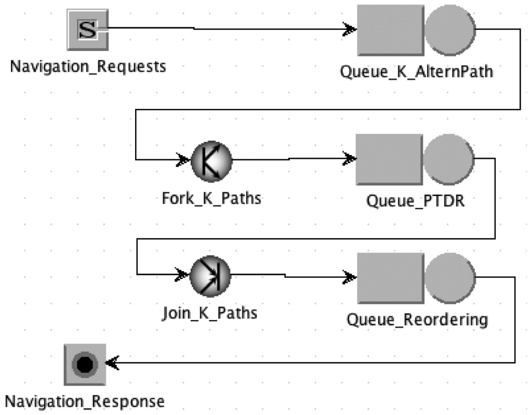


FIGURE 11. **Navigation pipeline modeled using JMT.**

number is in line with the consideration of having self-driving cars continuously connected with route planner, while the former has been derived by a simple estimation considering a Smart City such as the Milan urban area. In this area, there is a population of around 4 Million people and every day it is estimated that there are more than 5 Million trips, of which only less than 50 percent are done by using public transports [42], [43].

Under these conditions and considering the configuration with $\epsilon = 6\%$ and $95th$ *percentile*, we found that by adopting the proposed technique we obtained a 36 percent reduction in terms of number of resources needed to satisfy the target workload. In particular, we can distinguish two cases. The first case considers the number of resources needed to satisfy the steady-state conditions, and thus that the throughput in terms of input requests should be satisfied by all the stages. In this case, without the proposed optimization, we would have needed at least 777 computing resources (cores). Among them, 400 cores (52 percent of the entire set) should be dedicated to PTDR. By applying the proposed technique only 497 cores are needed, reducing to 120 (24 percent of the entire set) those required for the PTDR stage. The second case considers a more dynamic environment, where it is suggested to keep the average utilization rate of each station below 70 percent. While respecting this rule of thumb [44], the distribution of the system response time (the time passing from the navigation request to the response) is narrow, thus better to react to a burst of requests. In this second case, without the proposed optimization, we would have needed 1,010 cores to allocate the entire pipeline. The 572 of them (57 percent of the entire set) should be dedicated to the PTDR stage. By applying the proposed technique, 646 cores are enough to allocate the pipeline and out of them only 172 (26 percent of the entire set) are dedicated to the PTDR.

VII. CONCLUSIONS

In this paper, we presented an innovative approach to select dynamically the number of samples used in a Monte Carlo simulation to solve the Probabilistic Time-Dependent Routing problem. The proposed method samples quickly the data to extract an unpredictability feature to determine in a proactive manner the number of simulation to be executed while satisfying a certain error threshold. The runtime decision is based on a probabilistic error model – learned offline – correlating the unpredictability feature extracted from the data and the number of samples used by the Monte Carlo algorithm. Experimental results demonstrated that the proposed adaptive approach for the PTDR problem is able to save a large fraction of simulations (between 36 and 81 percent) with respect to a static approach while considering different traffic situations, paths and error requirements. Considering the entire navigation pipeline, composed also of the k-alternative path and reordering stages, the proposed technique guarantees a significative reduction in terms of computing resources. Finally, we adopted an aspect-oriented programming language (LARA) together with a flexible dynamic autotuning library (mARGOt) to reduce the effort necessary on the application developer for introducing the code needed to improve the execution efficiency.

ACKNOWLEDGMENTS

This work has been supported by European Commission under the grant 671623 FET-HPC-ANTAREX (AutoTuning and Adaptivity appRoach for Energy efficient eXascale HPC systems) and by The Czech Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science – LQ1602" and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center – LM2015070".

REFERENCES

[1] P. Toth and D. Vigo, *Vehicle Routing: Problems, Methods, and Applications*, vol. 18. Philadelphia, PA, USA: SIAM, 2014.
[2] R. Tomis, L. Rapant, J. Martinovič, K. Slaninová, and I. Vondrák, "Probabilistic time-dependent travel time computation using Monte Carlo simulation," in *Proc. Int. Conf. High Perform. Comput. Sci. Eng.*, 2015, pp. 161–170.
[3] M. Golasowski, R. Tomis, J. Martinovič, K. Slaninová, and L. Rapant, "Performance evaluation of probabilistic time-dependent travel time computation," in *Proc. IFIP Int. Conf. Comput. Inf. Syst. Ind. Manage.*, 2016, pp. 377–388.
[4] M. J. Gilman, "A brief survey of stopping rules in Monte Carlo simulations," in *Proc. 2nd Conf. Appl. Simulations*, 1968, pp. 16–20.
[5] A. Agafonov and V. Myasnikov, "Reliable routing in stochastic time-dependent network with the use of actual and forecast information of the traffic flows," in *Proc. IEEE Intell. Vehicles Symp.*, 2016, pp. 1168–1172.
[6] S. Samaranayake, S. Blandin, and A. Bayen, "A tractable class of algorithms for reliable routing in stochastic networks," *Procedia-Social Behavioral Sci.*, vol. 17, pp. 341–363, 2011.
[7] Y. M. Nie and X. Wu, "Shortest path problem considering on-time arrival probability," *Transp. Res. Part B: Methodological*, vol. 43, no. 6, pp. 597–613, 2009.
[8] M. Abeydeera and S. Samaranayake, "GPU parallelization of the stochastic on-time arrival problem," in *Proc. 21st Int. Conf. High Perform. Comput.*, 2014, pp. 1–8.
[9] M. Niknami and S. Samaranayake, "Tractable pathfinding for the stochastic on-time arrival problem," in *Proc. Int. Symp. Exp. Algorithms*, 2016, pp. 231–245.
[10] E. Nikolova, J. Kelner, M. Brand, and M. Mitzenmacher, "Stochastic shortest paths via Quasi-convex maximization," in *Proc. Eur. Symp. Algorithms*, 2006, pp. 552–563.

[11] A. Paraskevopoulos and C. Zaroliagis, "Improved alternative route planning," in *Proc. 13th Workshop Algorithmic Approaches Transp. Model. Optimization Syst.*, Sep. 2013, pp. 108–122. [Online]. Available: https://hal.inria.fr/hal-00871739

[12] C. Theodoros, B. Panagiotis, G. Johann, and L. Ulf, "Alternative routing: K-shortest paths with limited overlap," in *Proc. 23rd SIGSPATIAL Int. Conf. Advances Geographic Inf. Syst.*, 2015, pp. 68:1–68:4.

[13] T. Chondrogiannis, P. Bouros, J. Gamper, and U. Leser, "Exact and approximate algorithms for finding k-shortest paths with limited overlap," in *Proc. 20th Int. Conf. Extending Database Technol.*, 2017, pp. 414–425.

[14] H. Janssen, "Monte-Carlo based uncertainty analysis: Sampling efficiency and sampling convergence," *Rel. Eng. Syst. Safety*, vol. 109, pp. 123–132, 2013.

[15] Q. Xu, M. Sbert, M. Feixas, and J. Sun, "A new adaptive sampling technique for Monte Carlo global illumination," in *Proc. 10th IEEE Int. Conf. Comput.-Aided Des. Comput. Graph.*, Oct. 2007, pp. 191–196.

[16] C. A. Schaefer, V. Pankratius, and W. F. Tichy, "Atune-IL: An instrumentation language for auto-tuning parallel applications," in *Proc. Eur. Conf. Parallel Process.*, 2009, pp. 9–20.

[17] M. Püschel, J. M. F. Moura, B. Singer, J. Xiong, J. Johnson, D. Padua, M. Veloso, and R. W. Johnson, "SPIRAL: A generator for platform-adapted libraries of signal processing algorithms," *Int. J. High Perform. Comput. Applications.*, vol. 18, no. 1, pp. 21–45, 2004.

[18] R. C. Whaley and J. J. Dongarra, "Automatically tuned linear algebra software," in *Proc. ACM/IEEE Conf. Supercomput.*, 1998, pp. 1–27.

[19] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," in *Proc. 16th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2011, pp. 199–212.

[20] A. Paukštė, "Monte Carlo optimisation auto-tuning on a multi-GPU cluster," in *Proc. 2nd IEEE Int. Conf. Parallel Distrib. Grid Comput.*, Dec. 2012, pp. 894–898.

[21] V. Vassiliadis, C. Chalios, K. Parasyris, C. D. Antonopoulos, S. Lalis, N. Bellas, H. Vandierendonck, and D. S. Nikolopoulos, "Exploiting significance of computations for energy-constrained approximate computing," *Int. J. Parallel Program.*, vol. 44, no. 5, pp. 1078–1098, Oct. 2016.

[22] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, "Language and compiler support for auto-tuning variable-accuracy algorithms," in *Proc. Int. Symp. Code Generation Optimization*, Apr. 2011, pp. 85–96.

[23] J. S. Miguel and N. E. Jerger, "The anytime automaton," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, Jun. 2016, pp. 545–557.

[24] M. A. Laurenzano, P. Hill, M. Samadi, S. Mahlke, J. Mars, and L. Tang, "Input responsiveness: Using canary inputs to dynamically steer approximation," in *Proc. 37th ACM SIGPLAN Conf. Program. Language Des. Implementation*, 2016, pp. 161–176.

[25] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, "Proactive control of approximate programs," in *Proc. 21st Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2016, pp. 607–621.

[26] D. Gadioli, G. Palermo, and C. Silvano, "Application autotuning to support runtime adaptivity in multicore architectures," in *Proc. Int. Conf. Embedded Comput. Syst.: Archit. Model. Simul.*, 2015, pp. 173–180.

[27] D. Gadioli, E. Vitali, G. Palermo, and C. Silvano, "mARGOt: A dynamic autotuning framework for self-aware approximate computing," *IEEE Trans. Comput.*, vol. 68, no. 5, pp. 713–728, May 2019.

[28] E. Miller-Hooks and H. Mahmassani, "Path comparisons for a priori and time-adaptive decisions in stochastic, time-varying networks," *Eur. J. Oper. Res.*, vol. 146, no. 1, pp. 67–82, 2003.

[29] J. Martinovič, V. Snášel, J. Dvorský, and P. Dráždilová, "Search in documents based on topical development," in *Proc. Advances Intell. Web Mastering - 2*, 2010, pp. 155–166.

[30] M. Asghari, T. Emrich, U. Demiryurek, and C. Shahabi, "Probabilistic estimation of link travel times in dynamic road networks," in *Proc. 23rd SIGSPATIAL Int. Conf. Advances Geograph. Inf. Syst.*, 2015, Art. no. 47.

[31] J. M. Juritz, J. W. F. Juritz, and M. A. Stephens, "On the accuracy of simulated percentage points," *J. Amer. Statistical Assoc.*, vol. 78, pp. 441–444, 1983.

[32] D. C. Montgomery and G. C. Runger, *Applied Statistics and Probability for Engineers*. Hoboken, NJ, USA: Wiley, 2003.

[33] D. Zwillinger and S. Kokoska, *CRC Standard Probability and Statistics Tables and Formulae*. London, U.K.: Chapman & Hall, 2000.

[34] R. Tomis, J. Martinovič, K. Slaninová, L. Rapant, and I. Vondrák, "Time-dependent route planning for the highways in the Czech Republic," in *Proc. IFIP Int. Conf. Comput. Inf. Syst. Ind. Manage.*, 2015, pp. 145–153.

[35] P. P. Boyle, "Options: A Monte Carlo approach," *J. Financial Econ.*, vol. 4, no. 3, pp. 323–338, 1977.

[36] R. Koenker, *Quantile Regression*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[37] J. M. Cardoso, J. G. Coutinho, T. Carvalho, P. C. Diniz, Z. Petrov, W. Luk, and F. Gonçalves, "Performance-driven instrumentation and mapping strategies using the LARA aspect-oriented programming approach," *Softw.: Practice Experience*, vol. 46, no. 2, pp. 251–287, 2016.

[38] US Department of Transportation, Federal Highway Administration, US Department of Transportation, Federal Highway Administration – Traffic Report, 2014. [Online]. Available: https://www.fhwa.dot.gov/policy/publications.cfm

[39] UK Department for Transport, gov.uk, "Average annual daily flow and temporal traffic distributions," 2017. [Online]. Available: https://www.gov.uk/government/statistical-data-sets/tra03-motor-vehicle-flow

[40] M. Bertoli, G. Casale, and G. Serazzi, "JMT: Performance engineering tools for system modeling," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 10–15, Mar. 2009.

[41] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Upper Saddle River, NJ, USA: Prentice-Hall, 1984.

[42] Milano Agenzia Mobilita' Ambiente e Territorio, *Annu. Mobility Rep.*. 2015. [Online]. Available: https://www.amat-mi.it/it/documenti/

[43] Marco Bedogni, Milano Agenzia Mobilita' Ambiente e Territorio, *Road Traffic Measures the City Milan*. 2016. [Online]. Available: http://www3.gdos.gov.pl/Documents/Wizyty/W%C5%82ochy/Road%20Traffic%20Measures%20in%20the%20city%20of%20Milan.pdf

[44] M. Gribaudo, P. Piazzolla, and G. Serazzi, "Consolidation and replication of VMs matching performance objectives," in *Proc. Int. Conf. Analytical Stochastic Model. Techn. Appl.*, 2012, pp. 106–120.

EMANUELE VITALI received the MSc degree in computer engineering from Politecnico di Milano, Italy, in 2015. He is working toward the PhD degree in the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, since May 2017, where he was a research fellow. His main research interests include hardware architectures and application autotuning.

DAVIDE GADIOLI received the master of science degree in computer engineering, in 2013, and the PhD degree in computer engineering from Politecnico di Milano, Italy, in 2019. Currently, he is a postdoc with the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) of Politecnico di Milano. In 2015, he was a visiting student with IBM Research (The Netherlands). His main research interests include application autotuning, autonomic computing, and approximate computing.

GIANLUCA PALERMO received the master of science degree in electronic engineering, and the PhD degree in computer engineering from Politecnico di Milano, Italy, in 2002 and 2006. He is currently an associate professor with the Department of Electronics, Information and Bioengineering (DEIB) at the same University. Previously, he was consultant engineer with the Low-Power Design Group of AST - STMicroelectronics working on Network-on-Chip architectures, and research assistant with the Advanced Learning and Research Institute (ALaRI), Universita' della Svizzera Italiana (Switzerland). His research interests include design methodologies and architectures for embedded and HPC systems focusing on autotuning aspects. He is an active member of the scientific community serving in organizing and program committees of several conferences in his research areas. Since 2003, he published more than 100 scientific papers in peer-reviewed conferences and journals. He is member of the IEEE, ACM, and HiPEAC.

MARTIN GOLASOWSKI is working toward the PhD degree in the Advanced Data Analysis and Simulation Laboratory, IT4Innovations National Supercomputing Center of the Czech Republic, where he is a researcher. Topic of his research are high performance programming models for Monte Carlo methods and emerging heterogenous architectures. He participated in the H2020 FET project ANTAREX, H2020 ICT project LEXIS and in the research activities and development of FLOREON+ system for disaster management support. His other interests include parallel computing architectures, data processing, and visualisation. He has published more than 30 conference papers and several journal articles.

JOÃO BISPO received the bachelor's degree in 2006, and the PhD degree from the Instituto Superior Tcnico (IST), Lisbon, in 2012, with a thesis about automatic runtime migration of binary code to hardware. He is a post-doctoral researcher with the SPeCS Lab, Faculty of Engineering, University of Porto (FEUP). His research interests an on hardware synthesis from high-level descriptions and source-to-source compilation.

PEDRO PINTO received the MSc degree from the University of Porto, in 2012. He is currently working toward the PhD degree in the Faculty of Engineering, University of Porto. Since graduating, he has been involved in several research projects in the area of compilers. His main research interests include source-to-source compilation, application analysis and optimization, and code transformations, as well as broader topics such as programming languages, high-performance computing, and machine learning.

JAN MARTINOVIČ is currently head of Advanced Data Analysis and Simulation Lab, IT4Innovations National Supercomputing Center, VSB Technical University of Ostrava, Czech Republic. His research activities are focused on information retrieval, data processing, design and development of information systems, and disaster management. His activities also cover a development HPC as a Service Middleware which allows to use HPC infrastructure remotely by specific API. He is coordinator of the H2020 ICT project LEXIS (Large-scale Execution for Industry & Society). He had previous experience with coordination of the different contracted research activities and had responsibility for the technical coordination of the several national projects. He was the leader of IT4I as a partner of the two H2020-FETHPC-2014 projects ANTAREX and ExCAPE. He is also responsible for the research and development team of FLOREON+ system for disaster management support. He has published more than 100 papers in international journals and conferences.

KATEŘINA SLANINOVÁ received the doctoral degree in informatics from the VSB Technical University of Ostrava, Czech Republic. She is deputy head of Advanced Data Analysis and Simulations Lab, IT4Innovations National Supercomputing Center, VSB Technical University of Ostrava, Czech Republic. Her research interests include information retrieval, traffic analysis, vehicle routing problem, hyperparameter search, data mining, process mining, and complex networks. Her recent activities also cover cooperation with SMEs in areas such as traffic management, artificial intelligence, time series analysis, etc. She participated in H2020 ICT project LEXIS and H2020 FETHPC project ANTAREX. She worked within the team of the Center for the Development of Transportation Systems RODOS. She has published more than 70 papers in international journals and conferences.

JOÃO M. P. CARDOSO received the PhD degree in electrical and computer engineering from the IST/UTL (Technical University of Lisbon), Lisbon, Portugal, in 2001. He is full professor with the Department of Informatics Engineering, Faculty of Engineering, University of Porto, and a senior researcher with INESC TEC. Before, he was with the IST/UTL (2006-2008), a senior researcher at INESC-ID (2001-2009), and with the University of Algarve (1993-2006). In 2001/2002, he worked for PACT XPP Technologies, Inc., Munich, Germany. He has been involved in the organization and served as a Program Committee member for many International Conferences. He was co-scientific coordinator of the FP7-EU project REFLECT and technical manager of the H2020-EU project ANTAREX, and coordinator of various national funded projects. He has (co-)authored more than 200 scientific publications. His research interests include compilation techniques, domain-specific languages, reconfigurable computing, high-level synthesis and application-specific architectures, and high-performance computing with an emphasis in embedded computing. He is a senior member of the IEEE and ACM.

CRISTINA SILVANO is a full professor of computer architectures with the Department of Electronics, Information and Bioengineering (DEIB), Politecnico di Milano, Italy. Her main research interests include energy-efficient embedded systems, design space exploration of manycore architectures, and application autotuning for HPC. She has published more that 160 scientific papers in peer-reviewed journals and conferences, five books and she holds several patents in collaboration with Group Bull and STMicroelectronics. She was project coordinator of three European projects: H2020-ANTAREX, FP7-2PARMA and FP7-MULTICUBE. She has served in the organizing and program committees of several major conferences in computer architectures, embedded systems, and electronic design automation. She is associate editor of the *ACM Transactions on Architecture and Code Optimization* and the *IEEE Transactions on Computers*. She served as independent expert reviewer for the European Commission and for several science foundations. In 2017, she has been elevated to the grade of the IEEE fellow.