Noname manuscript No.
(will be inserted by the editor)

Performance-driven Analysis for an Adaptive Car Navigation Service on HPC Systems

Leonardo Arcari · Marco Gribaudo · Gianluca Palermo · Giuseppe Serazzi

the date of receipt and acceptance should be inserted later

Abstract Car navigation system recently established as an imperative utility for modern navigation on road networks. The rising wave of self-driving cars along with an increasing demand for real-time traffic data is expected to generate massive growth of routing requests and processing time on large graphs representing the urban networks. Therefore larger and more powerful computing infrastructures are required. In the context of smart cities, new, dynamic solutions are needed in order to deliver high-quality car navigation services, powered by municipal traffic monitoring data, capable of handling such a vast expected demand with reasonable employment of financial resources.

In this work we introduce an adaptive car navigation system and its performance model used to tune the size of the computing infrastructure as a function of the characteristics of the environment considered. The model has been validated for a smart city environment using the data collected on Milan urban area. [1]

Keywords Adaptive car navigation system · Queueing Networks · Petri Nets · Multiformalism models · Performance-driven model.

1 Introduction

The automotive navigation systems market size is expected to observe an exponential growth in the next eight years, with an anticipated compound annual growth rate of 9.95% during the forecast period. Most important driving factors are the increase in the number of vehicles worldwide and the raising demand for real-time traffic data.

Politecnico di Milano, DEIB, Milano Italy
E-mail: {firstname.lastname}@polimi.it

[1] 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)

Indeed, both passengers and commercial vehicle drivers share the same interest to reach their destinations as soon as possible. People are putting more and more trust in car navigation systems, relying on computer-suggested routes as opposed to their driving experience. Such a scenario will be even more realistic with the expected worldwide spread of self-driving cars. Indeed, drivers will eventually turn into actual passengers, leaving to their car, and hence to its car navigation system, the burden of choosing a path for them.

In this context, there are various opportunities to transform the current traffic situation into a better one and to exploit the availability of traffic data to optimize driving conditions. For instance, smart cities might employ their historical traffic monitoring data to provide citizens with an accurate car navigation service that would make local drivers happier and the whole city less congested.

This paper describes an adaptive car navigation system designed to target future smart cities. The design of the functional aspects of the system has been done together with a performance model capable to capture its adaptivity degrees. An hybrid approach composed by multi-class Queueing Networks and colored Petri Nets formalisms has been employed to study how to size the IT infrastructure for smart city requirements.

The remainder of this paper is organized as follows. Section 2 provides an overview of the target adaptive car navigation system, while Section 3 describes the performance model developed. Section 4 describes the experimental validation of the proposed methodology, and Section 5 concludes the paper.

2 The Adaptive Car Navigation System

This section introduces the adaptive car navigation system designed to meet the possible needs of a smart city that wants to provide this service to its citizens.

2.1 The smart city perspective

Considering the rising wave of self-driving cars [9], the amount of car navigation requests will increase rapidly [8] together with the need for real-time updates and the processing on large graphs representing the urban network. Currently, even drivers of cars, taxis and motorcycles rely on navigation systems to reach their destinations within the city to discover the routes with less traffic. Moreover, mail, packages and food delivery services experience their business rising [13] as a result of the non-stopping trend of online goods purchasing. The depicted scenario causes increasingly complex problems in the management of traffic jams and levels of air pollution levels in some areas of the city.

The proposed adaptive car-navigation system addresses the problems of effective route planning and traffic optimization. The municipality will provide a service significantly improving the navigation experience, either from

more reliable and faster paths in terms of expected travel time, and from discounts on other expenses like parking tickets, fines or permissions to access limited-traffic areas. Upon a request for a path to some destination, the service would respond with a single path that is optimal according to some quality metric, subject to a number of constraints set by the municipality, for instance minimizing the number of traveling vehicles around the city center. The ideal target of the service would be autonomous-cars that would follow passively any computed route. Indeed, human drivers can still decide to take detours from the suggested path, either because different from the one they are used to, or because they believe they know a better path. This behavior, if shared among a considerable number of drivers, would compromise the effectiveness of traffic-splitting service. As a solution, the city might produce a rewarding plan for well-behaving drivers that follow the proposed paths and obtain discounts on urban taxes.

From the municipality perspective, the service should have a low cost, in terms of amount of resources required, and should satisfy the whole load of requests. Furthermore, it should absorb peaks of unexpected requests that could occur in the presence of accidents, harsh climate conditions or natural disasters that would affect the viability. As for the end-user viewpoint, a good service would provide high-quality results, i.e., a route with best travel time, with a realistic response time, possibly tuning the path along the way through re-routing requests.

2.2 System Architecture

The proposed adaptive car-navigation system is composed of a three-stage pipeline, shown in Figure 1, called: (i) Alternative Route Planning (ARP), (ii) Probabilistic Time Dependent Routing (PTDR), and (iii) Reordering phase, respectively.

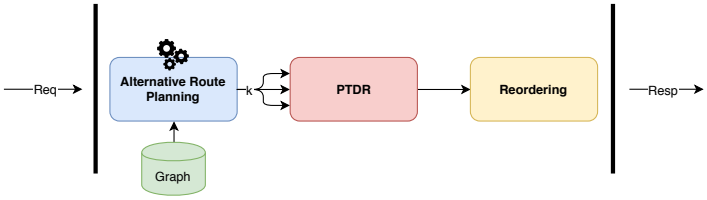

Fig. 1 A graphical representation of the proposed car-navigation system pipeline.

The pipeline is considered adaptive since there is the possibility to play with several configuration parameters to customize the computation according to the dynamic traffic conditions and the constraints of the navigation provider.

Alternative Route Planning (ARP). The first stage consists of determining a number k of alternative paths from source to destination to be passed to the

next step. This problem is known in the literature as *k-Shortest Paths with Limited Overlap* (k-SPwLO) [4], since the alternative paths should overlap less than a given threshold (θ). In the context of navigation, identifying the shortest path is not enough to discover a good solution since the traffic characteristics must be taken into account. A shortest path in terms of distance or average traveling time might not be the optimal one in every time frame, therefore an expected arrival time should be computed considering the current speed profile on the given routes to select the best path. The ARP stage is the most important and it is also the one that requires the most computation time. This module has been parameterized to balance the time between the solution and the quality of the results. In particular, we included two ARP algorithms from literature, ONEPASS+ [5] and Penalty [3,17], both parameterized in terms of maximum overlap among the paths and number of alternative paths. Both algorithms are based on heuristics. While ONEPASS+ generates alternative paths shorter than Penalty, much closer to the shortest path, it takes longer to compute the solution especially for small values of maximum overlap.

Probabilistic Time Dependent Routing (PTDR). In the second stage, for each one of the k alternative paths, the travel time is estimated using the PTDR [18] module. This module compute the probability distribution of the arrival times taking into account the traffic status on the road networks. Since the exact computation of the expected travel time distribution has exponential complexity, a Monte Carlo approach is used to efficiently approximate the solution of the problem [19,7]. The PTDR module implements the adaptability of the navigation system to the varying traffic conditions. Different routes are suggested for the same source-destination pair, based on an in-time computed expectation of travel time. The more alternative routes are computed in the ARP stage, the higher the chance to find a better one.

Reordering stage For every single request, this stage gathers the timing information provided by the k instances computed by the PTDR module together with other features related to the selected paths, and reorder them according to a cost function. The goal is to return to the user not only the best route in terms of traveling time, but also taking into account other municipality policies. For instance, penalizing the paths that cross the city center, which are close to hospitals and schools, or that pass in an area under investigation for high level of pollution.

2.3 System Adaptability

System adaptivity has been considered in many works. For example [20,21] give an overview of several adaptive dynamic allocation algorithms that take decisions based on on-line and up-to-date measurements to achieve desirable QoS levels. The context of the system proposed in this paper is to provide a traffic optimization service for a smart city. The navigation service has been

parameterized for being able to modulate the quality of the result with the amount of resources needed to support the service. In addition to module parameterization, the application flow has been also adapted to consider a couple of more constraints derived by the specific service. First, there is the need to limit the amount of time to derive the best route. This avoids long waiting time on the user side (*timeout*). Second, there is the need also to manage unexpected peaks of requests much larger than planned.

Figure 2 shows the application flow in the three cases: *regular*, regular with *timeout*, and *peak* of requests. If there is a regular number of requests, the ARP module is configured with the ONEPASS+ algorithm that guarantees the highest quality of results for the K paths. In case the ARP module is able to finalize the computation within the timeout, all the paths are sent to the next stages. Otherwise, in case we reach the configured timeout, only a part of the results are calculated using ONEPASS+, the others are quickly derived using Penalty. On the opposite, in case of peak of requests, the ARP module is configured only to provide the shortest path. Indeed, we do not consider dropping requests a valid option.

In order for our fast lane to be effective, it should process requests considerably faster than the main one, to cope with the workload increase. Nonetheless, we want to keep serving valid responses to navigation users, maybe degrading solution quality, but we do not consider dropping requests a valid option. Indeed, request drop might be interpreted by users, or clients, as momentary service unavailability, causing recurrent, successive queries to the system, viciously further increasing the arrivals rate

The system adaptivity can be further exploited considering also that the navigation system manages two types of requests. The first type includes the *first-routing* requests. Those are used to initialize the client with the route to follow. The second type includes the *re-routing* requests, that are issued by the client periodically seeking for better alternative routes. The difference between the two types of requests is mainly on the user constraints. Indeed, while the *re-routing* requests are hidden to the driver and thus do not require a tight time-out, the *first-routing* requests should be served in shorter time.

Moreover, given the complexity of providing an always updated graph representing the traffic status, the service has been deployed on an HPC infrastructure. Indeed, this paper concentrates only on the part of the system related to the management of the routing requests, hiding the details of the part of the system managing the graphs updates. The advantage of the HPC infrastructure is its the capability to support fast communications and parallel updates on a large shared graph. On the other side, in HPC context there is a certain rigidity in resource acquisition and release, which may result in both slow resource scaling in presence of unexpected workload increase and money wasting in presence of a workload significantly lower than expected.

The tuning of such a system and in particular an accurate planning of the resources is a key point for having an efficient solution. On one hand, a navigation service provider would employ historical data to characterize its workload, for example categorizing it with respect to daytime time slots. On

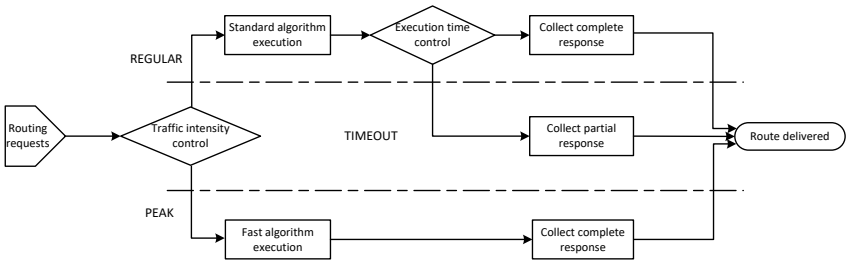

Fig. 2 Navigation adaptivity controls

the other hand, understanding when we have to consider that there is a peak of requests is fundamental to avoid critical cases.

3 Modeling

To briefly recap, we are considering a software system implementing a pipeline of three stages. As first stage, *ARP* stage computes a number of alternative paths with limited overlapping. In the second stage, *PTDR* module estimates the expected travelling time for each alternative, according to the current traffic conditions. Finally, *Reordering* stage elects the best path to follow according to arrival time and other arbitrary policies set by the navigation service provider. In this section we will describe how to model the three considered components in order to correctly size the system in order to achieve the desired QoS, while reducing as much as possible the running costs.

Many modelling techniques can be used to model the considered system. For example, G-Net [6] allows to consider shared resources and batch services. In this work we have preferred to use a multi-formalism modelling technique [10] to exploit the features of two different modelling approaches, each one better suited to describe the different characteristics of the various components of the system. In particular, the first module of the navigation service (the alternative routing planning stage) is described with the multi-formalism modelling environment supported by JMT [2], that combines Multi-class Queueing Networks (QNs) [11] and Colored Generalized Stochastic Petri Nets (GSPNs) [14]. The combination of QNs and GSPNs allows to describe both the resource allocation and the queuing behavior of the processes execution. The other two stages are instead modeled using conventional fork/join and other convenrional QN components, since they are not characterized by a complex batch and time-out structure.

To model the different types of requests, *first-routing and re-routing*, we use two different type of entities, that are mapped into two open classes for what concerns the queuing network primitives, and to colored tokens when dealing with Petri nets. To simplify the notation, in the following we will call

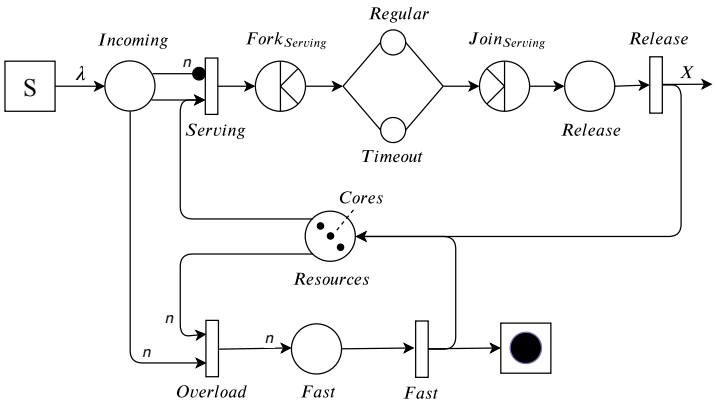

Fig. 3 The multi-formalism model of the *Alternative Route Planning* stage with *Fast lane*.

these different type of entities *classes* regardless of the fact that they refer to QNs or GSPNs primitives.

3.1 ARP-module

The first part of the pipeline represents the most complex stage of the processing pipeline since it relies on several degrees of adaptivity in the service implementation. Figure 3, illustrates the proposed Multi-formalism model for ARP stage. It is mainly composed by two separate and parallel processing pipelines that share the same computational resources. The one used for serving requests in the normal operation case is shown in the upper part of the model, while the second one that is employed only in case of system overload (peaks of requests) to reduce the waiting time before they are served by the ARP module is represented on the lower part of the figure.

Place *Resources* models the number of parallel *servers* available for the ARP stage. The initial marking of this place determines the number of nodes available in the system. A special class, named *Cores* is assigned to the tokens in this place: this type of entities is necessary to keep track of the number of busy processing units, that corresponds to cores of the HPC infrastructure. Routing requests arrive at the system from the *Source* node, represented the box S in Figure 3. Incoming requests, here represented by Petri net tokens, are queued in place *Incoming* and are served whenever at least token of class *Cores* is available in place *Resources*. Resource acquisition is modelled by immediate transition *Serving*, which is enabled by one token of either $first-routing$ or $re-routing$ class token and one token of *Cores* class. As a result of the firing of transition *Serving*, a new customer with the same class as the class of the token acquired from place *Incoming* is generated in the $Fork_{Serving}$ queuing station. This node, represented with a "K" shaped surrounded by a circle, corresponds to a *Fork* node, that splits the incoming job into two

tasks, sent respectively to the two infinite servers queuing stations *Regular* and $Timeout$, each represented with a single small circle.

Regular servers process each task according to their effective service time distribution, while $Timeout$ servers follow a deterministic distribution that represents the deadline before which a task must be completed. Both servers route tasks to the $Join_{Serving}$ station, that is represented with a mirrored "K" shaped pattern surrounded by a circle, that implements a *Join* QN component characterized by a *Quorum* policy: the job is considered completed when a fraction of the tasks in which it has been split have been completed. In this case, it is enough that just one of the two tasks is completed to consider the job ended, which corresponds either to a timeout or to a regular completion. The second task completing, will be simply discarded when reaching the join node. The infinite server semantic of the two queuing nodes *Regular* and $Timeout$ can be used since the number of servers running the tasks is controlled by place *Resources*. Jobs departing from $Join$ station are sent in *Release* place, enabling *Release* transition which moves the routing request to the next pipeline stage and restores a token of $Cores$ class in *Resources* place. In the model this is represented by an arrow that does not have any destination nodes: the meaning is that jobs exiting from this point will continue being served by the other stages of the system.

Indeed, while scaling policies are in place to issue resource acquisition requests from the HPC provider in case of excessive arrivals rate, within the timeframe between request and actual resource allocation, incoming requests could saturate all available service stations by increasing the system response time beyond an acceptable threshold. To approach this problem, we introduce a fast lane processing in parallel to the main pipeline to solve these issues. The fast lane computes a single shortest path and return it to the user without any further processing. This particular path is chosen whenever the number of customers waiting in the ARP queue exceed a given threshold; otherwise the main pipeline is preferred. This heavy load behavior is modelled by the $Overload$ immediate transition, which is enabled whenever the marking of the $Incoming$ place reaches the threshold level n regardless of their original class. Threshold n is a parameter which must be fine tuned depending on the arrival rate and the service times. In order to avoid excessive fluctuations in the selection of the routes by continuously performing switches between the two alternatives, the system waits to have enough resources available to serve a batch of n jobs whenever the threshold is reached. This is modelled by the arc of weight n (tokens of $Cores$ class) that connects place *Resources* to transition $Overload$. Moreover, to prevent jobs following the light load route when the threshold is reached, an *inhibitor arc* of weight n jobs (of either service classes $first{-}route$ or $re{-}route$) connects place $Incoming$ to transition $Serving$. Then, jobs are forwarded to the $Fast$ place to be served by the infinite server $Fast$ timed transition. When job processing completes, the job exits the system, responding to waiting user, and a token with the $Cores$ class is restored in place *Resources*.

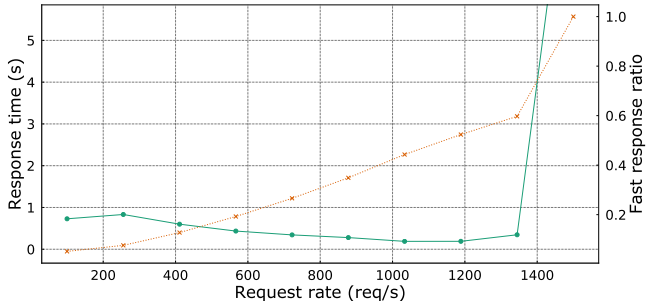

Fig. 4 Overloading scenario of a system tailored to support a $\lambda = 100$. In green, solid lines we plot the system response time. In orange, dotted lines we plot the rate of requests routed to fast lane to cope with the excessive workload.

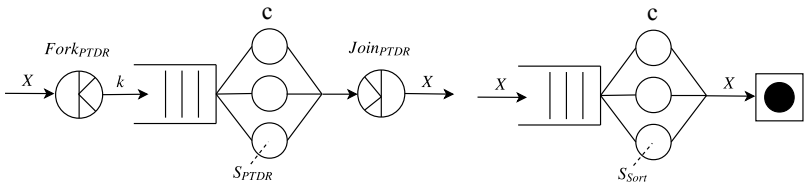

Fig. 5 PTDR and Reordering stages

To quantitatively show the effect of the fast lane, Figure 4 shows the response time of the ARP module when increasing the number of requests above the value used for dimensioning the infrastructure. In particular, the infrastructure has been tailored for a request rate $\lambda = 100$ $[req/sec]$. As expected, by increasing the number of requests per second, we see a growing percentage of requests being served by the *fast-lane*. Different is the behaviour of the response time. It remains below 1 second up to $\lambda = 1300$ $[req/sec]$, then it grows dramatically. This is the point in which also the fast-lane service is no longer able to cope with the heavy load and the system inevitably saturates. Note however that this phenomenon occurs only when the incoming load is more than thirteen times larger than the arrival rate for which it was sized. Note that in the intermediate heavy load region, that is with $100 < \lambda < 1300$, the average response time decreases: this occurs because a larger fraction of requests are served with the fast-lane. Please note that users served by the fast-lane are provided with lower quality answers: with this design, as the load increases the system privileges speed over quality, giving a better performance at the expense of the effectiveness of the suggested routes.

3.1.1 PTDR and Reordering stages

The other two stages are simpler w.r.t. ARP stage. Indeed, alternative routes discovered in ARP stage are forwarded to PTDR module to compute a travel time estimate on each of them. Since this process can be performed in parallel

this is modelled with a simple fork-and-join queuing model. In particular, each ARP solution produces k alternative routes, and it is then spawn into k parallel tasks each one running the PTDR algorithm on a different part of the solution coming from the previous stage. The left part of Figure 5 shows the corresponding Fork-Join model. In particular, jobs arriving at this stage (represented by an arrow that does not start from any node) are forked into k tasks for each job by the $Fork_{PTDR}$ station. Generated tasks are all sent to an M/M/c queue, serving tasks according to a Poisson process of rate $1/S_{PTDR}$. $Join_{PTDR}$ station, then, applies a *Standard Join* strategy to wait for all the k-tasks to complete, and then forwards jobs to the Reordering stage (represented by an arrow that does not have any destination node).

The reordering stage, shown in the right hand side of Figure 5, is the fastest and the easiest stage to model. At this step, arbitrary sorting rules are applied according to service provider policies. For instance, paths can be penalized if flowing across the city center, or nearby some city event. This stage is modeled as a Poisson process of rate $1/S_{Sort}$. Therefore, we employ a simple M/M/c station processing jobs arriving from PTDR module and returning a final response to service customers. The termination of the process is modelled by the sink node, represented by a square with a black circle inside. Node that jobs being served by the fast-lane, since they return a single alternative, and do not need to pass through the PTDR and reordering stage, terminates immediately as modelled by the sink node shown in the bottom right corner of Figure 3.

4 Case study: The Milan Urban Area

We applied the proposed car-navigation system to the traffic optimization of a real smart city. We have considered the traffic of Milan urban area by analyzing the requirements of such a smart city and applying step by step our methodology to provide a realistic service for this municipality. We solved the multi-formalism performance model of the navigation pipeline by using the Java Modeling Tools (JMT) [1,2].We performed our experiments on the region shown in Figure 6 and retrieved by OpenStreetMap [16] database, with coordinates ranging from $[45.3743, 45.5509]$ latitude to $[9.0519, 9.3507]$ longitude (EPSG:4326/WGS84 reference system).

The population of this area is about 4 Million people. Every day, local agencies estimate to have around 5 Million trips, and only less than 50% are done using public transportation. [15,12]. Thus, we consider to have 2.5 Million trips per day.

First-routing and re-routing requests have different performance objectives defined by the following parameters. In both cases the number of alternative paths is fixed to $k = 5$ with an overlap threshold $\theta = 50\%$, while the timeout has been configured to 1 second for the first-routing and 3 seconds for the re-routing requests, respectively.

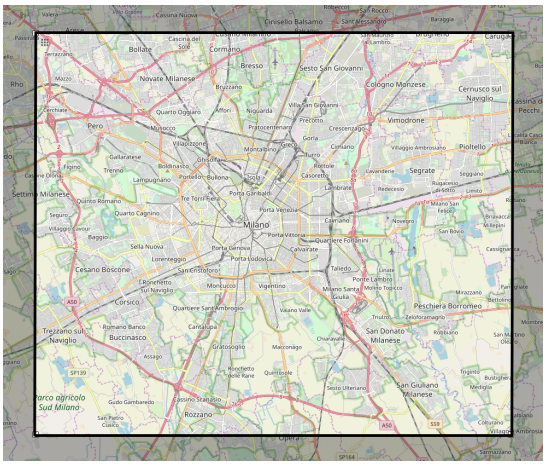

Fig. 6 The considered Milan urban area map.

Table 1 Simulation vs. Actual average response times

Performance Index	Simulation	Actual	rel. err.
First-routing Response Time	$0.498 \div 0.502$ sec.	$0.518 \div 0.522$ sec.	3.8%
Re-routing Response Time	$0.646 \div 0.654$ sec.	$0.655 \div 0.665$ sec.	1.5%

4.1 Model Validation

We have tested a prototype of the proposed application on a 16 cores HPC node, based on an *Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz and 20MB Cache*, which was handling a reduced workload with respect to the one of the whole city. In particular, we used a first-routing requests arrival rate of $\lambda_f = 4$ reqs/sec, and a re-routing requests arrival rate of $\lambda_r = 40$ reqs/sec. The experiment was run until 200K requests considering source-destination pairs within the urban area were collected. The service times of the components of the models in Figure 3 and 5 have been computed by fitting their corresponding service time distributions against half of the data collected during the experiment. Then the models have been analyzed using discrete event simulation by the JMT tool, and performance metrics have been collected. The other half of the data collected has been used to compute average measurements against which compare the results obtained with the model.

Table 1 compares the 99% confidence intervals of the response times for the two classes of requests: first-routing and re-routing. The results obtained from the JMT simulation of the model are very close to those actually measured for both types of requests, as shown by the relative error, that is less than 4%.

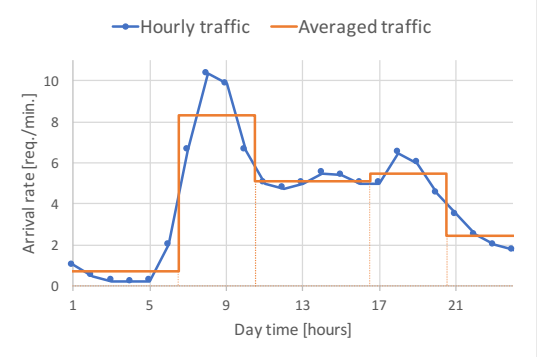

Fig. 7 Average daily traffic of routes requests per-hour of a working day in the urban area considered.

4.2 Model exploitation

In this section, we use the JMT model to identify the computational capacity, in terms of number of cores, required by the car-navigation service in the Milan urban area to achieve a given performance objective. In particular, we use the model to determine the minimum number of cores required to obtain a target average utilization of 70% for each of them. This threshold has been defined in order to account for several conditions that might increase the expected workload, such as peak of requests for unexpected human or natural events, or unavailability of infrastructural elements.

In Figure 7 we show the daily traffic distribution published in a study of the Milan municipality [15]. For each hour interval shown on the x-axis, the y-axis reports the percentage of car trips measured in the corresponding time slot. As expected, the traffic is minimal during the early hours of the day, and explodes up to the peak between 7 and 9 am. Then again, after some hours of stable values, we observe another peak between 5 and 7 pm, when most people drive back home. The daily traffic profile has been divided into 5 intervals having very different mean values. The intervals considered are: 1-6, 7-10, 11-16, 17-20 and 21-24. Each interval, shown in Figure 7 by an orange line, represents a different percentage of daily traffic. For each of them, we executed a what-if experiment on the number of cores to determine the lowest number of resources required to match the CPU utilization objective. We assume that the interarrival times of requests are exponentially distributed within each time slot. Initially, we evaluate the number of trips in each time slot as a parameter to determine the numbers of first routing requests. Then, since every trip has been measured to last slightly more than 20 minutes, and re-routing requests are sent every two minutes, we calculate the re-routing request rate considering a mean of 10 periodic requests for each first-routing query.

Table 2 summarizes the results obtained. Each row describes the optimal infrastructure size for each time slot. According to traffic-monitoring data,

Table 2 Number of cores required by each stage vs different time slots of the day.

Time Slot	ARP	PTDR	Reordering	Total cores
01 - 06	51	10	1	**62**
07 - 10	584	111	2	**697**
11 - 16	373	71	1	**445**
17 - 20	393	75	1	**469**
21 - 24	200	38	1	**239**

early hours in the morning and late hours in the night register the least traffic levels and, consistently, our analysis reports the least amount of resources required to run the navigation service, with just 62 cores. On the other hand, in correspondence to the maximum traffic peak, from 7 to 10 am, the amount of resources grows to reach the maximum value of 697 cores.

These results show an important characteristic of the proposed methodology in terms of economic return. While it is true that with 697 cores the navigation service can be provided without slowdowns throughout the day, our model has shown that this amount of resources is necessary only during the maximum peak of 4 hours, and is oversized for the other time slots. This means that for the rest of the day, a significantly smaller number of cores could be rented from supercomputing centers, drastically reducing the expenses for the municipality. In fact, a system running the whole day on 697 cores would cost the provider 16728 core-hours. By applying the results provided by our system model, we obtain the lowest value of 8662 core-hours, obtaining 49% *savings* on infrastructure rental costs.

5 Conclusions

The actual car navigation systems must provide a high level of adaptivity to cope with the varying characteristics of the smart cities. The degree of adaptivity of the proposed car-navigation system enables a variety of customizations to effectively tailor such a service on the provider needs. We have shown its validity considering the data collected in a real smart city, the Milan urban area.

References

1. Bertoli, M., Casale, G., Serazzi, G.: Jmt: performance engineering tools for system modeling. SIGMETRICS Perform. Eval. Rev. **36**(4), 10–15 (2009). DOI http://doi.acm.org/10.1145/1530873.1530877
2. Casale, G., Serazzi, G., Zhu, L.: Performance evaluation with java modelling tools:: A hands-on introduction. SIGMETRICS Perform. Eval. Rev. **45**(3), 246–247 (2018). DOI 10.1145/3199524.3199567. URL http://doi.acm.org/10.1145/3199524.3199567
3. Chen, Y., Bell, M.G., Bogenberger, K.: Reliable pre-trip multi-path planning and dynamic adaptation for a centralized road navigation system. In: Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE, pp. 257–262. IEEE (2005)

4. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U.: Alternative routing: K-shortest paths with limited overlap. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '15, pp. 68:1–68:4. ACM, New York, NY, USA (2015)
5. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U.: Exact and approximate algorithms for finding k-shortest paths with limited overlap. In: 20th International Conference on Extending Database Technology: EDBT 2017, pp. 414–425 (2017)
6. Fourneau, J., Gelenbe, E., Suros, R.: G-networks with multiple classes of negative and positive customers. Theor. Comput. Sci. **155**(1), 141–156 (1996). DOI 10.1016/0304-3975(95)00018-6
7. Golasowski, M., Tomis, R., Martinovič, J., Slaninová, K., Rapant, L.: Performance evaluation of probabilistic time-dependent travel time computation. In: IFIP International Conference on Computer Information Systems and Industrial Management, pp. 377–388. Springer (2016)
8. Grand View Research: Global Positioning Systems (GPS) Market Size, Share & Trends Analysis Report By Deployment, By Application (Aviation, Marine, Surveying, Location-Based Services, Road), And Segment Forecasts, 2018 - 2025. https://www.grandviewresearch.com/industry-analysis/gps-market (2018)
9. Grand View Research: Self driving Cars and Trucks Market Size, Share & Trends Analysis Report By Application (Transportation, Defense), By Region (NA, Europe, APAC, South America, MEA), And Segment Forecasts, 2020 - 2030. https://www.grandviewresearch.com/industry-analysis/driverless-cars-market (2018)
10. Gribaudo, M., Iacono, M.: Theory and Application of Multi-Formalism Modeling, 1st edn. IGI Global, Hershey, PA, USA (2013). DOI 10.4018/978-1-4666-4659-9
11. Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1984)
12. Marco Bedogni, Milano Agenzia Mobilita' Ambiente e Territorio: Road Traffic Measures in The City of Milan (2016)
13. Market Realist: A Look at the Courier Service Industry in the United States. https://articles.marketrealist.com/2015/07/look-courier-service-industry-united-states/ (2015)
14. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets, 1st edn. John Wiley & Sons, Inc., New York, NY, USA (1994)
15. Milano Agenzia Mobilita' Ambiente e Territorio: Annual mobility report. https://www.amat-mi.it/it/documenti/ (2015)
16. OpenStreetMap contributors: Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org (2017)
17. Paraskevopoulos, A., Zaroliagis, C.: Improved alternative route planning. In: OASIcs-OpenAccess Series in Informatics, vol. 33. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2013)
18. Tomis, R., Rapant, L., Martinovič, J., Slaninová, K., Vondrák, I.: Probabilistic time-dependent travel time computation using monte carlo simulation. In: International Conference on High Performance Computing in Science and Engineering, pp. 161–170. Springer (2015)
19. Vitali, E., Gadioli, D., Palermo, G., Golasowski, M., Bispo, J., Pinto, P., Martinovic, J., Slaninova, K., Cardoso, J.M., Silvano, C.: An efficient monte carlo-based probabilistic time-dependent routing calculation targeting a server-side car navigation system. IEEE Transactions on Emerging Topics in Computing (2019)
20. Wang, L., Brun, O., Gelenbe, E.: Adaptive workload distribution for local and remote clouds. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2016, Budapest, Hungary, October 9-12, 2016, pp. 3984–3988 (2016). DOI 10.1109/SMC.2016.7844856
21. Wang, L., Gelenbe, E.: Adaptive dispatching of tasks in the cloud. IEEE Trans. Cloud Computing **6**(1), 33–45 (2018). DOI 10.1109/TCC.2015.2474406