Multi-Objective Microservice Orchestration: Balancing Security and Performance in CCAM

Stefano Berlato[1,2], Silvio Cretti[2], Domenico Siracusa[2], Silvio Ranise[2,3]
[1]Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa, Genoa, Italy
[2]Center for Cybersecurity, *Fondazione Bruno Kessler*, Trento, Italy, {sberlato,scretti,dsiracusa,ranise}@fbk.eu
[3]Department of Mathematics, University of Trento, Trento, Italy

***Abstract*—We (devise and) demonstrate the benefits of a methodology and a toolset for orchestrating Cloud-native applications to balance the minimization of risks due to the presence of security threats and the achievement of service performance requirements — expressed on, e.g., computational resources, network throughput and latency. The demo proves the effectiveness of the methodology in orchestrating a set of microservices implementing a prominent Cooperative, Connected and Automated Mobility (CCAM) service.**

***Index Terms*—Orchestration, Cloud, Security, CCAM**

I. Introduction

Cooperative, Connected and Automated Mobility (CCAM) allows vehicles to communicate and make collective driving decisions. The 5G-CARMEN [1] project investigated CCAM, proposing a Cooperative Lane Change (CLC) service in which vehicles install a front-end (FE) to broadcast their position, speed and trajectory through Cooperative Awareness Messages (CAMs); a message broker, called data manager (DM), groups CAMs by areas of interest (e.g., highway acceleration lanes). Hence, FEs can build an internal representation of nearby traffic conditions, making lane change maneuvers easier and safer. CLC considers using Cryptographic Access Control (CAC) to guarantee the confidentiality and the integrity of CAMs while allowing authorized vehicles only to participate. With CAC, one or more proxies (PRs) en/decrypt CAMs generated by FEs before transmission to or from the DM, while a database — called metadata manager (MM) — stores cryptographic keys; we summarize the data flow in Figure 1. In CLC, FEs, PRs, DM and MM can be implemented as microservices managed by Kubernetes (K8s) — chosen for its popularity and features such as resiliency, load balancing, and zero-touch deployment — across 4 types of computing regions: Public Cloud (e.g., Azure), Private Cloud, Edge and vehicles.[1] The Private Cloud is in the premises of the organization offering CLC, while the Edge is (logically) co-located with the road-side infrastructure.

Problem Characterization. The orchestration (and especially the definition of the placement into regions) of microservices is not trivial when considering the peculiarities of CLC. Indeed, like many CCAM services, CLC is marked by a delicate trade-off between performance and security [4]. On the one hand, CLC requires end-to-end (E2E) vehicle-to-vehicle latency to be $\leq$ 100ms for the exchange of up-to-date CAMs [1, §D4.1]; bandwidth is not a concern given the limited size of CAMs ($<$ 300 bytes). Similarly, CLC microservices require a certain amount of computational resources for the timely derivation of driving decisions. On the other hand, the placement of the microservices must consider the (possible) presence of a heterogeneous set of threats (e.g., external attackers, malicious insiders, partially trusted service providers) to the security (i.e., confidentiality, integrity, and availability) of CAMs [4].

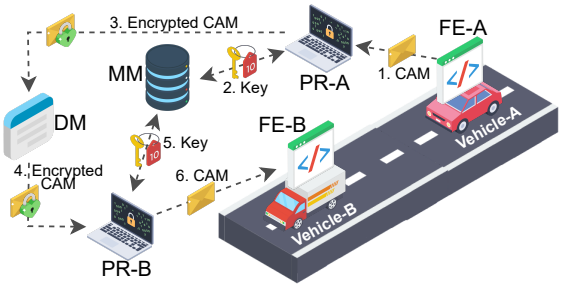

Fig. 1: Data Flow in CLC

Solution and Demo. We (devise and) demonstrate the benefits of a methodology and a toolset that map the problem of finding the placement of CLC microservices to regions of a K8s cluster into a Multi-Objective Combinatorial Optimization Problem (MOCOP) [5] which considers the fulfillment of service performance requirements plus CAMs security. We implement our methodology — named MOMO (short for Multi-Objective Microservice Orchestration) — as a placement algorithm for K8s by integrating it into FogAtlas [6], a framework to manage the Cloud-to-Thing Continuum. Then, we propose a demo over the CLC service to showcase the effectiveness of MOMO with respect to 3 other basic placement strategies.

II. Multi-Objective Microservice Orchestration

Given the sets of microservices $\mathcal{M}$ and regions $\mathcal{R}$, MOMO derives the set of all placements $\mathcal{PL} = \{pl \in \mathbb{P}(\mathcal{M} \times \mathcal{R}) : |\{m : (m, -) \in pl\}| = |\mathcal{M}|\}$ ($\mathbb{P}$ is the power set and $|\cdot|$ the set cardinality). Then, MOMO evaluates placements against a set of *objective functions*, where an objective function $g_o : \mathcal{PL} \mapsto \mathbb{Z}$ measures how much a placement attains an objective $o \in \mathcal{O}$; the internal definition of g_o (i.e., how to compute its value)

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union — NextGenerationEU. Also, we gratefully thank Marco Centenaro (orcid.org/0000-0003-1664-8015) for his valuable feedback on this article.

[1]Deploying microservices of Cloud-native applications in smart vehicles is realistic and believed to be an integral part of the evolution of intelligent transportation systems in both academia [2] and the industry [3].

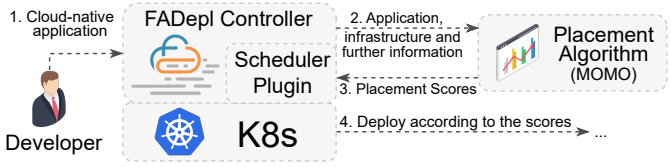

Fig. 2: Architecture of FogAtlas

depends on o. Finally, MOMO maps the problem of finding the placements yielding the maximum tuple of objective function values as the MOCOP $\max_{pl2PL}(g_{o_1}(pl), \ldots, g_{o_n}(pl))$. We use the *Pareto Dominance* relation as ordering relation for max: given $pl_1, pl_2 \in \mathcal{PL}$, pl_1 dominates pl_2 iff $(\forall o \in \mathcal{O}\ g_o(pl_1) \geq g_o(pl_2)) \wedge (\exists o \in \mathcal{O}\ g_o(pl_1) > g_o(pl_2))$. A $pl \in \mathcal{PL}$ is a *Pareto Optimal* (i.e., one of the best) solution to the MOCOP when no placement $pl^0 \in \mathcal{PL}$ dominates pl.

We apply MOMO to CLC (see Figure 1), thus $\mathcal{M} = \{$FE-A, FE-B, PR-A, PR-B, MM, DM$\}$ and $\mathcal{R} = \{$Vehicle-A, Vehicle-B, Edge, Private, Public$\}$ — for simplicity, we assume each region $r \in \mathcal{R}$ to have one node nr, i.e., a single (physical or virtual) machine on which microservices can run. As said in Section I, we identify two types of objectives, i.e., performance and security objectives: the former comprises computational resources, network bandwidth and E2E latency ($\mathcal{O}_P = \{CPU, MEM, BAN, LAT\}$), the latter confidentiality (C), integrity (I) and availability (A) of CAMs ($\mathcal{O}_S = \{C, I, A\}$); hence, $\mathcal{O} = \mathcal{O}_P \cup \mathcal{O}_S$. We define performance objective functions in Table I: a placement must be feasible according to the available computational resources (g_{CPU}, g_{MEM}) while E2E Latency (g_{LAT}) considers cumulative latency across the service data flow (i.e., the one in Figure 1) — network bandwidth (g_{BAN}) is not a concern in CLC. We use the security objective functions g_C, g_I, and g_A — measuring the achievement of the corresponding objective from 0 (low) to 2 (high) — defined in [7], where the authors propose a risk assessment to evaluate data security (e.g., CAMs) in services using CAC; the intuition is that, the more likely (or impactful) is a security threat to strike in a region or communication channel, the higher the risk associated with placing a microservice in that region or exchanging data over that channel. Unfortunately, for lack of space, we cannot report here more details, and refer the interested reader to [7].

As deriving driving decisions timely is crucial for safety, we seek the most secure among the placements that satisfy all performance objectives. Hence, we solve the MOCOP using a *bounded objective function method* [5] that maximizes a tuple of objective functions — (g_c, g_I, g_A), in our case — and uses others (i.e., $g_{CPU}, g_{MEM}, g_{BAN}$ and g_{LAT}) to form additional constraints; formally, $\max_{pl2PL}(g_C(pl), g_I(pl), g_A(pl))$ *subject to* $0 < g_{op}(pl) \leq 1\ \forall op \in \mathcal{O}_P$. We highlight that this is just one of the many methods to tackle MOCOPs (see [5]).

TABLE I: Definition of Performance Objective Functions

$g_{CPU}(pl) = 1$	if $(\sum_{(m;r)2pl}\ m_{CPU}) \leq nr_{CPU}\ \forall r \in \mathcal{R}$, else 0
$g_{MEM}(pl) = 1$	if $(\sum_{(m;r)2pl}\ m_{MEM}) \leq nr_{MEM}\ \forall r \in \mathcal{R}$, else 0
$g_{BAN}(pl) = 1$	(network bandwidth is not a concern in CLC [1, §D4.1])
$g_{LAT}(pl) =$	1 if $(\sum_{(m_1;m_2)2DF}\ LAT(r_1;r_2)) \leq$100ms — where $(m_1;r_1);\ (m_2;r_2) \in pl$ — else 0

m_{CPU} and m_{MEM} are the processing and memory computational requirements of a microservice m, while nr_{CPU} and nr_{MEM} are the processing and memory computational resources available in the node nr in the region r. DF is the data flow shown in Figure 1, i.e., $DF = \{$ (FE-A, PR-A), (PR-A, MM), (PR-A, DM), (DM, PR-B), (PR-B, MM), (MM, PR-B), (PR-B, FE-B) $\}$, while $LAT(r_1; r_2)$ is the latency between r_1 and r_2 (see Figure 3).

III. Architecture and Demo Setup

Figure 2 shows the architecture of FogAtlas: (1) a Cloud-native application, encoded as a graph of microservices and modelled as a K8s Custom Resource Definition with a FADepl resource (see [8] for more details), is sent to the FADepl Controller component; this (2) invokes a Placement Algorithm (i.e., MOMO) with, as input, the application (i.e., $\mathcal{M}$), the infrastructure (i.e., $\mathcal{R}$), and further information to compute objective functions (i.e., computational resources available in each node, network resources, computational resources required by each microservice, E2E latency constraint). The Placement Algorithm (3) computes Placement Scores encoding the best placement identified; the Scheduler Plugin (a custom plugin implemented in the K8s Scheduling Framework acting during the *score phase* of the K8s scheduling cycle) uses the scores to assign a node to each microservice. Finally, K8s (5) deploys the microservices on the infrastructure accordingly.

We run the demo in an emulated environment with a K8s cluster deployed in our data center. As in Figure 3, the cluster has the 5 regions presented in Section I. We refer to experimental settings validated in 5G-CARMEN [1, §D4.3] and the work in [9] for *distributing* the cluster with realistic network latency and computational resources for each node. For the demo, we reduce the computational resources by a factor of 8 to fit the cluster in our data center and tune the latency with the `tc` utility. As for microservices, we choose Redis as MM, the Mosquitto MQTT broker as DM, and the `CryptoAC` tool [10] implementing CAC for MQTT as PR-A/B. Since 5G-CARMEN lacks the implementation for the FE (and driving decisions are not relevant in our demo), we implement FE-A/B using `Locust` and `ACME` [11] to measure the E2E latency of a CAM going from vehicle-A to vehicle-B (see Figure 1). Finally, we derive computational resource requirements of the MM, DM and PR-A/B from their documentation; for FE-A/B — there being no implementation — we can only make reasonable assumptions. For consistency with the cluster, we scale requirements on computational resources by 8 and report them in Table II (the CPU is in milli-units, the MEM in MB).[2]

IV. Demonstration and Results

Using the setup in Section III, we demonstrate the effectiveness of MOMO in identifying the placement that satisfies

[2]The replication package is available at gitlab.fbk.eu/fogatlas-k8s/uc-ccam.

TABLE II: Requirements of CLC Microservices

Microservice	CPU	MEM	Microservice	CPU	MEM
FE-A	50	50	FE-B	50	50
PR-A	130	64	PR-B	130	64
MM	50	50	DM	50	50

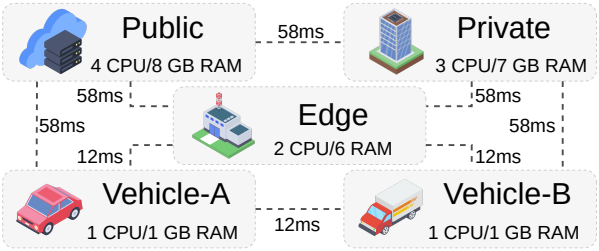

Fig. 3: K8s Cluster Demo Setup

the performance requirements while maximizing the security of CAMs in CLC. To this end, we compare MOMO with 3 other placement strategies, for a total of 4 experiments:

1) *K8s* - as a baseline strategy, we use the default placement of K8s, which mainly considers resourceful nodes;
2) *performance* - we use MOMO only for the satisfaction of performance constraints ($0 < g_{op}(pl) \leq 1 \ \forall op \in \mathcal{O}_P$);
3) *security*: we use MOMO only for the maximization of the security of CAMs ($\max_{pl2PL}(g_C(pl), g_I(pl), g_A(pl))$);
4) *MOMO*: we use MOMO as described in Section II.

A. Demo Workflow

In all 4 experiments, we adopt the following workflow:

1) force the assignment of FE-A/B to vehicle-A/B, respectively (as intuitively expected by CLC);
2) deploy the rest of the CLC microservices following the placement strategy of the corresponding experiment;
3) take note of the placement of microservices on the nodes;
4) configure and run `Locust` to start transmitting CAMs;
5) collect (through `Locust`) the E2E latency of the transmission of 360 CAMs — each experiment lasts 6 minutes so to reduce measurement errors — and compute the E2E latency mean and median.

B. What the Audience Will See

To speed up the demo execution, we plan to set up 4 identical K8s clusters and run the aforementioned experiments in parallel. Through a dedicated interface, the audience will observe the placement of microservices, comparing the results obtained in each experiment on the performance and security requirements imposed.

C. Results

We report the placements, security objective function values, and median/mean E2E latency results in Table III (we omit the other requirements as all placements respect them). Note that E2E latency includes also the microservices processing time (around 15ms) which is relatively small (4%—37%) but not irrelevant. The *K8s* and *security* experiments do not respect the E2E latency constraint ($\leq$ 100ms), as the former just assigns microservices to the most resourceful nodes ignoring latency among regions, while the latter achieves the highest *possible* security, assigning PRs to vehicles-A/B — for CAMs E2E protection — and the MM and DM to the Private Cloud, the most secure region (but also distant from vehicles). The *performance* experiment fulfills all constraints by relying on the Edge. However, g_C, g_I, and g_A have lower values: one of the reasons is that CAMs sent from vehicle-A are protected by PRs only after reaching the Edge, and are thus exposed to the threats present in the (public) network in-between. MOMO, on the one hand, improves E2E latency while losing little security with respect to the *security* experiment and, on the other hand, improves security while entailing slightly higher (but still acceptable) E2E latency with respect to the *performance* experiment. In other words, the placement proposed by MOMO is not the best either performance- or security-wise, but it allows to balance a heterogeneous set of relevant — and seemingly conflicting — objectives.

V. Conclusions and Future Work

In this demo paper, we demonstrated MOMO, a methodology for orchestrating microservices that consists of solving a MOCOP over a set of relevant (performance and security) objectives, in the context of the CLC service. Compared with other 3 placement strategies, MOMO proved its effectiveness in satisfying both security and performance requirements.

As future work, we plan to test MOMO after scaling on the number of vehicles and to verify the dynamic re-placements of microservices by monitoring the evolution of objective functions, e.g., considering changes in latency, in microservices processing time, in real-time usage of computational and network resources and in security threats as they occur (e.g., DDoS).

References

[1] 5G-CARMEN. https://5g-ppp.eu/5g-carmen/.
[2] R. Morabito, R. Petrolo, V. Loscri, N. Mitton, G. Ruggeri, and A. Molinaro. Lightweight virtualization as enabling technology for future smart cars. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1238–1245. IEEE, 2017.
[3] RedHat. https://www.redhat.com/en/blog/running-containers-cars.
[4] M. Centenaro, S. Berlato, R. Carbone, G. Burzio, G. F. Cordella, R. Riggio, and S. Ranise. Safety-related cooperative, connected, and automated mobility services: Interplay between functional and security requirements. *IEEE Vehicular Technology Magazine*, 16(4):78–88, 2021.
[5] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.
[6] FogAtlas. http://fogatlas.fbk.eu/.
[7] S. Berlato, R. Carbone, A. J. Lee, and S. Ranise. Formal modelling and automated trade-off analysis of enforcement architectures for cryptographic access control in the cloud. *ACM Transactions on Privacy and Security*, 25(1):1–37, 2022.
[8] FogAtlas CRD. https://gitlab.fbk.eu/fogatlas-k8s/crd-client-go.
[9] F. Palumbo, G. Aceto, A. Botta, D. Ciuonzo, V. Persico, and A. Pescapé. Characterization and analysis of cloud-to-user latency: The case of azure and AWS. *Computer Networks*, 184:107693, 2021.
[10] CryptoAC. https://cryptoac.readthedocs.io/.
[11] Access Control Mechanisms Evaluator. https://github.com/stfbk/ACME.

TABLE III: Experiments Placements, Security, E2E Latency

Experiment	Microservices and Regions				Security			E2E Latency	
	PR-A	MM	DM	PR-B	g_C	g_I	g_A	Median	Mean
K8s	Public	Private	Edge	Private	○	◐	◐	373ms	374ms
performance	Edge	Edge	Edge	Edge	○	◐	◐	**40ms**	**40ms**
security	V-A	Private	Private	V-B	●	◐	●	373ms	375ms
MOMO	V-A	Edge	Edge	V-B	●	◐	◐	**94ms**	**95ms**

V-A and V-B are abbreviations for vehicle-A and vehicle-B, respectively
○, ◐ and ● correspond to 0, 1, and 2, respectively (the fuller, the better)