Information Resources Management Journal
Volume 34 • Issue 1 • January-March 2021

PrEstoCloud:
A Novel Framework for Data-Intensive Multi-Cloud, Fog, and Edge Function-as-a-Service Applications

Yiannis Verginadis, Athens University of Economics and Business, Greece
Dimitris Apostolou, University of Piraeus, Greece
Salman Taherizadeh, Joseph Stefan Institute, Slovenia
Ioannis Ledakis, Ubitech, Greece
Gregoris Mentzas, National Technical University of Athens, Greece
Andreas Tsagkaropoulos, National Technical University of Athens, Greece
Nikos Papageorgiou, National Technical University of Athens, Greece
Fotis Paraskevopoulos, National Technical University of Athens, Greece

ABSTRACT

Fog computing extends multi-cloud computing by enabling services or application functions to be hosted close to their data sources. To take advantage of the capabilities of fog computing, serverless and the function-as-a-service (FaaS) software engineering paradigms allow for the flexible deployment of applications on multi-cloud, fog, and edge resources. This article reviews prominent fog computing frameworks and discusses some of the challenges and requirements of FaaS-enabled applications. Moreover, it proposes a novel framework able to dynamically manage multi-cloud, fog, and edge resources and to deploy data-intensive applications developed using the FaaS paradigm. The proposed framework leverages the FaaS paradigm in a way that improves the average service response time of data-intensive applications by a factor of three regardless of the underlying multi-cloud, fog, and edge resource infrastructure.

KEYWORDS

Edge, Fog Computing, IoT Applications, Multi-Clouds

1. INTRODUCTION

Fog computing extends multi-cloud computing by enabling services or application functions to be hosted close to their data sources, which are typically Internet of Things (IoT) sensors. Hosting functions close to data sources can reduce the latency and cost of delivering sensor-generated data to a remote cloud and can improve the Quality of Service (QoS; Hao et al. 2017). A key challenge for fog computing is auto-scaling, i.e. the autonomous capacity for continuous adaptation and control of the computing infrastructure through the recognition of insights and knowledge in the data. Insights from

DOI: 10.4018/IRMJ.2021010104

the analysis of sensor-generated data can adapt the computing resources to meet current or predicted computing needs, save cost, increase performance and reliability, and meet environmental concerns.

To realize the deployment of applications and services and take advantage of the adaptive capabilities of fog computing, two new software engineering paradigms have emerged: Serverless and Function-as-a-Service (FaaS), which are seen as two enabling technologies for next-generation fog computing (Van Eyk et al. 2018). The serverless paradigm exploits functions or microservices as the unit of deployment and is hence more efficient than using a virtual machine (VM) or a container since their inherent complexity becomes transparent to the application owner (Castro et al., 2019; Trihinas et al., 2018). FaaS is a facet of serverless computing where applications can run server-side logic in stateless compute containers that can be event-triggered, ephemeral (may only last for one invocation), and fully manageable by a third party. Such a serverless paradigm is desirable for many event-based IoT applications, especially mission-intensive applications, as well as applications requiring energy efficiency and data delivery reliability (Gusev et al., 2019). The increased complexity of heterogeneous fog computing infrastructures poses management challenges to the DevOps of IoT applications, however (Chiang et al., 2016). In response, advanced cloud management tools and methods have started to emerge in an effort to automate infrastructure performance tuning and anomaly detection (Di Martino et al., 2019; Mahesh et al., 2011).

This paper reviews prominent fog frameworks that deploy and monitor applications that span over multiple clouds, fog and edge resources. It discusses associated challenges and proposes a novel fog architecture and framework for managing dynamically multi-cloud and edge resources in order to cope with the requirements of FaaS-enabled applications. Our research objective focuses on the development and evaluation of a framework to support the seamless deployment of fog computing applications on heterogeneous cloud, fog, and edge resources independently of underlying infrastructures while supporting the FaaS paradigm and providing auto-scaling capabilities.

2. FOG COMPUTING FRAMEWORKS

2.1. State of Play

Cloud, edge and fog computing is a vibrant and continuously evolving area of distributed computing (Liu et al., 2017; Carroll 2015). The US National Institute of Standards and Technology (NIST) predicts that the fog computing market is likely to emerge as a viable vertical niche, as fog computing deployment needs to adapt to particular physical devices, networks and market needs (Iorga et al. 2018). We analyzed the current snapshot of available offerings and tested prominent frameworks from leading vendors to identify the current state of the art in this domain. We evaluated if existing frameworks support the eight characteristics defined by NIST, six of them defined as essential for distinguishing fog computing from other computing paradigms while two of them are considered optional. The essential ones are a) the contextual location awareness, with the goal of low latency, b) the geographical distribution in contrast to the centralized cloud, and (c) the heterogeneity of the deployment along with (d) the interoperability and federation e) the support for real-time interactions rather than batch processing and finally, (f) the microservice scalability and agility of the federated, fog-node clusters that make fog computing adaptive. The two optional characteristics are (g) the usage of wireless networking and (h) the mobility of devices.

Since our research focuses on the enabling FaaS on heterogeneous cloud, fog, and edge resources, we augmented the eight NIST characteristics with seven additional ones that we consider relevant for supporting FaaS-enabled applications. Specifically, the vendor-specific cloud characteristic refers to whether the framework in question is bound to the use of only certain providers for virtualized computing resources. Support for FaaS characteristic indicates the ability of a framework to cope with functions deployed following the serverless paradigm; the deployment optimization characteristic refers to the capability to consider contradicting business goals and solving constraint

programming problems for detecting the most optimal use of available resources. The offline device communication characteristic highlights the capability of a framework to cope with run-time situations in fog deployments in which the communication fails unexpectedly for a short period of time. The support deployment and management of application on fog and cloud resources characteristic refers to the ability to orchestrate the deployment of applications on both cloud and edge resources. The fog resources asset management characteristic refers to fog resource registration and supervision capabilities and the event-based alerting corresponds to seamless capabilities for aggregating monitoring information from the dispersed topology and detecting situations that call for adaptation.

Our analysis presented in Table 1 indicates that AWS Greengrass (2020), AWS Wavelength (2020), Azure IoT Edge (2020) and Google Cloud IoT Core (2020) are better suited to support fog computing than OpenWhiskLean (2020) because the latter does not support context awareness with respect to latency reduction, cannot cope with intermittent edge connectivity and the support for wireless communication is limited. AWS Greengrass introduces an architecture comprised of three essential components: the Greengrass Core, the Greengrass Group, and the IoT Device SDK. AWS

Table 1. Overview of Fog Computing Framework Characteristics

	AWS Greengrass	AWS Wavelength	Azure IoT Edge	Google Cloud IoT Core	OpenWhisk Lean	Nebbiolo	Cloudify
Context awareness	Yes, location only	Yes, predefined	Yes, location only	Yes, location only	No	Yes, location only	Yes, predefined
Geographical distribution	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Heterogeneity	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Interoperability and federation	Yes	Yes	Yes	Yes	Federation is not supported	Yes	Yes
Real-time interactions	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Microservice scalability and agility of federated, fog-node clusters	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Predominance of wireless access	Yes	Yes	Yes	Yes	No	Yes	Yes
Support for mobility	No	Yes within same Wavelength Zone	No	No	No	Yes	No
Cloud Vendor Specific	Yes	Yes	Yes	Yes	No	No, AWS/ Azure	No
Support for FaaS	Yes	No	Yes	Yes	Yes	No	Yes, AWS Lambda
Deployment optimization of microservices on the edge	No	No	No	No	No	No	No
Intermittent edge connectivity support	Yes, Device Shadows	n/a	Yes	No	No	No	No
Support on Deployment and Management of application on fog/cloud resources	Partially, predefined deployment at the edge	Partially, predefined deployment at the edge of 5G networks	Yes	No	either Cloud or Edge	Yes	Yes
Fog resources Asset Management	Yes	n/a	Yes	No	Yes	Yes	Yes
Event-based alerting	Yes	Yes	Yes	Yes	No	No	Yes

Greengrass Core is a compute node hosted on a local device that bears the appropriate software to enable the use of local device resources like cameras, serial ports, or GPUs. IoT devices, along with Greengrass Cores and Lambda functions, are organized into Greengrass Groups that correspond to collections of inter-communicating entities. AWS Greengrass supports serverless code (i.e., AWS Lambda functions) deployable to both cloud and edge. AWS Wavelength was recently announced and a roll out of the service is expected in 2020. It enables developers to build applications that deliver single-digit millisecond latencies to mobile devices and end-users. While it doesn't support FaaS, it allows containerized applications to be deployed on Wavelength Zones. These zones embed AWS compute and storage services within the telecommunications providers' datacenters at the edge of the 5G networks and seamlessly access other AWS services in the region. Microsoft Azure IoT Edge allows the execution of Azure Functions, which refer to C# precompiled versions of application functions. The framework can be deployed on Linux and Microsoft Windows operating systems and can run on a resource-constrained device such as a Raspberry Pi Zero. The Azure IoT Edge Agent bootstraps itself each time an edge device is activated in order to connect it to the Azure IoT Edge Runtime.

Google Cloud IoT Core runs on Google's serverless infrastructure, which scales automatically in response to real-time changes and its main focus is on digesting data from edge devices in a secure way. Google advertises this technology as anti-vendor lock-in as the supported environments include Cloud Functions, local development environment, on-premises, Cloud Run for Anthos, and other Knative-based serverless environments. Nevertheless, the main functionalities are built around vendor specific products that are cannot easily accommodate applications deployed over Google Cloud IoT to be executed in an open environment like OpenWhisk Lean. Moreover, Nebiolo (2020) is hybrid platform for supporting industrial IoT applications. It introduces its own rich operating system stack (FogOS) that enables the distributed real-time processing. Although it offers an advanced management system doesn't support FaaS functionalities and optimized placement. Finally, Cloudify (2020) also known as multi-cloud everything-as-a-service (EaaS) supports both cloud and edge deployments without optimized placement. The cloud support is vendor independent while the serverless capabilities are considered only through AWS Lambda functions.

Summarizing this analysis, we found that none of the existing prominent fog computing frameworks offers an integrated solution capable of addressing the full list of the fundamental challenges of serverless data-intensive applications. Therefore, we propose a novel framework called PrEstoCloud, which can fully support these aforementioned essential characteristics of such services.

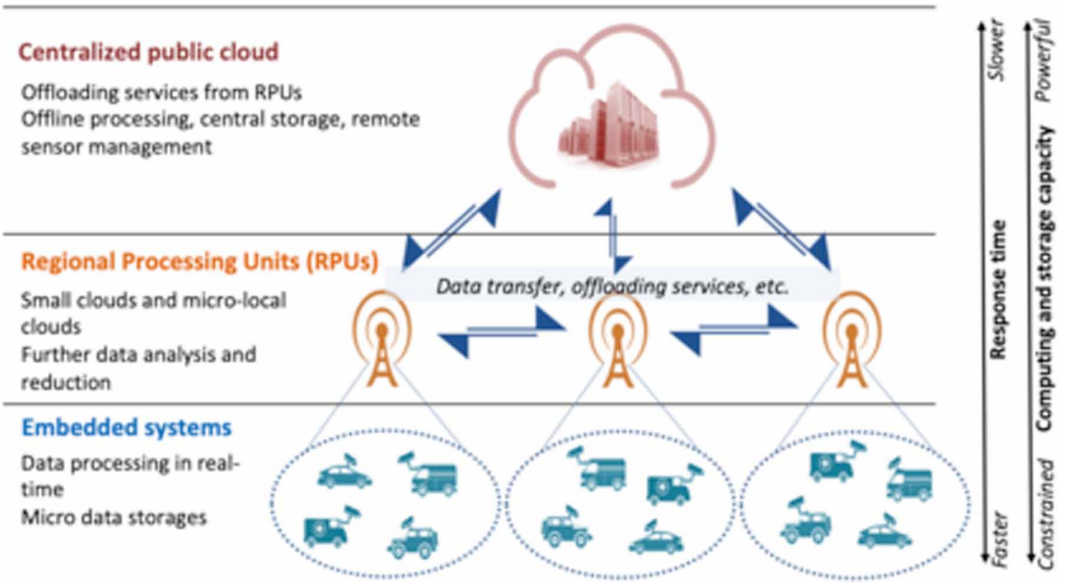
2.2. Challenges and Requirements

To investigate the challenges and requirements associated with the deployment of FaaS applications on fog infrastructures, we focus on an intelligent logistics IoT application that collects and analyzes video streams from cameras installed on and inside vehicles. The application performs real-time anomaly detections, triggering runtime alerts to the fleet manager, and observes driving dynamics for situations such as sudden acceleration and aggressive turning, where possible accidents may occur. Decomposing the application into loosely coupled functions packaged into microservices distributes computational workloads among three processing layers (Figure 1):

- *Embedded systems*: In this layer, microservices deployed on embedded systems collect video streams, process a large volume of data and store them temporarily.
- *Regional processing units (RPUs):* This layer resides in the proximity of vehicles and is aimed at data reduction operations such as data aggregation and filtering.
- *Centralized public cloud:* This layer provides central storage and powerful processing capabilities. It also supports management functions to control IoT devices installed in vehicles remotely.

Dynamic IoT applications such as the intelligent logistics one puts forward the following challenges and requirements for the fog computing framework with respect to its auto-scaling capabilities: multi-

Figure 1. Intelligent logistics IoT application



resource allocation including edge, fog and cloud; optimal microservice deployment; and context-aware microservice offloading.

2.2.1. Multi-Resource Allocation

One of the main benefits of the cloud computing model is the elasticity of the infrastructure that allows the user to manage the size and configuration of their computing fleet finely. This adaptation can be driven by indications, coming either from the user, or from an automated system. One great challenge of cloud computing has been to build such an automated resource allocator/de-allocator, that will make it possible to accurately match the actual usage in order to avoid under-provisioning or over-provisioning. Under-provisioning leads to performance issues and additional latency, and over-provisioning uselessly increases a user's bill and poorly exploits the cloud resources. As an alternative to allocating and de-allocating, an option to optimize resource usage is to reconfigure the resources allocated to the application (Kokkinos et al., 2013). It also requires being able to accurately model variations in the workload, which is not an easy task, particularly for public clouds, where trace data is scarce. Reiss et al. (2012) paved the way to model a workload dynamicity that is experienced in (public and private) heterogeneous cloud computing platforms.

Duplyakin et al. (2013) propose a multi-cloud environment to process user requests. In this system, users specify the percentage of the resources to be used in each cloud computing environment. If user specifications are not satisfied due to a lack of resources, the system will balance the load progressively on already-deployed instances until the user requirements are met. This approach allows the best possible use of hybrid clouds, but requires an intrusive solution installed inside a VM. Kailasam et al. (2013) consider the optimization of the execution time in a hybrid cloud context. They propose three heuristic-based scheduling methods that adapt themselves to the evolution of the resources of the workload and the availability of the clouds. This approach allows the use of hybrid clouds in the context of HPC but requires modifications to the application, or, more specifically, to the application's task scheduler. Leitner et al. (2013) propose a model that enables running applications to burst into a different cloud infrastructure. The authors propose a framework for the creation of elastic cloud applications. This framework enables the monitoring of the performance and decides when to burst

to a public cloud and, in the other direction, where to consolidate into the private cloud, but this approach needs the re-writing of the applications.

Other research works have been achieved on the hybrid cloud, but focusing on economic aspects, e.g., Guo et al., (2012) and Tordsson et al., (2012). In a fog environment, multi-resource allocation should not only facilitate the selection of the best resource on the grounds of cost minimization but also ensure application reliability and QoS and avoid vendor lock-in. This ability allows the selection of the best resource to run microservice-based on end-user requirements, which is considered an important challenge in fog computing.

2.2.2. Optimal Deployment

Microservices orchestration should balance various criteria, such as making the best utilization of computing resources, increasing response time while decreasing network traffic load over the Internet. There are also QoS-specific trade-offs that must be considered, particularly in situations when vehicles are dynamically changing their geographic positions. For example, some processing may need to be moved from one RPU to another in the proximity of vehicles. Reliable execution of the application as a whole requires coordination of execution flows. Fog computing requires orchestration capabilities to manage both the application and the data flow between available resources. Lorido-Botran et al. (2014) identify five categories of approaches for realizing auto-scaling: Threshold-based rules, Reinforcement learning (RL), Queuing theory (QT), Control theory (CT), and Time series analysis (TS). Gandhi et al. (2014) identify five similar auto-scaling approach categories: prediction models, control theoretic techniques, queuing-based models, and black-box and grey-box approaches. Black-box models use machine learning or statistical methods for decision making in order to overcome the problem of modeling the cloud application using expert knowledge. Grey-box models are hybrid approaches that use models in combination with machine learning. Qu et al. (2018) reason that resource estimation in horizontal or vertical auto-scaling can be performed using rules, fuzzy-inference, application-profiling, analytical modeling, machine learning or hybrid methods. Analytical modeling includes queuing theory and markov chains. Machine learning includes reinforcement learning and regression. Regression is applied in auto-scaling techniques that use time-series analysis or control theory.

Due to its highly networked nature, serverless computing challenges the ways in which auto-scaling approaches can be applied. There is a need for well-defined event protocols and methods, for example, for message queues (Abowd et al. 1999) to continuously monitor the state of serverless applications and the infrastructure to ensure that the latter always has enough capacity to handle the current workload. Resilience to failure is another essential requirement of a FaaS application because each application request is divided and translated to different service calls. A bottleneck in a specific service operation should not bring the entire system down.

2.2.3. Context-Aware Offloading

Typically, microservices offload persistence to the host or use highly available cloud data stores to provide a persistence layer. Offloading to the host makes it difficult to port containers from one host to another. Technologies like the Flocker and Docker volume plugins address this problem by creating a separate persistence layer that is not host-dependent. Other technologies such as JPPF (Java Parallel Processing Framework, JPPF, https://www.jppf.org/.), CloneCloud (Chun et al., 2011), MACS (Kovachev et al., 2012), and JADE (Qian & Andresen, 2014) enable the partitioning of cloud application processing tasks to multiple processing nodes at source-code level. JPPF, for instance, supports the allocation of processing tasks on any operating system capable of running a JVM (also on Android despite running its own JVM).

In a fog environment, offloading should span all three processing layers: embedded systems, RPUs, and centralized public clouds. For example, when instructed by the data processing performed in the embedded systems layer, video streams and other measured data should be sent to fragments deployed on the RPUs of the second layer for content analysis. Other fragments with higher processing

requirements, such as driver behavior profiling can only run on the cloud. Deployment constraints should be clearly defined and interpreted by the fog framework so that fragments are deployed on capable resources. Furthermore, there is a need to manage offloading and onloading of microservices intelligently, e.g., if the processing requires to exceed the capabilities of RPUs, new processing modes should be spawned in the cloud. According to RPU mobility, location, and network condition, the system may need to dynamically select a new RPU, re-deploy the service, and hence support the application through context-aware offloading capabilities.

3. PRESTOCLOUD FRAMEWORK

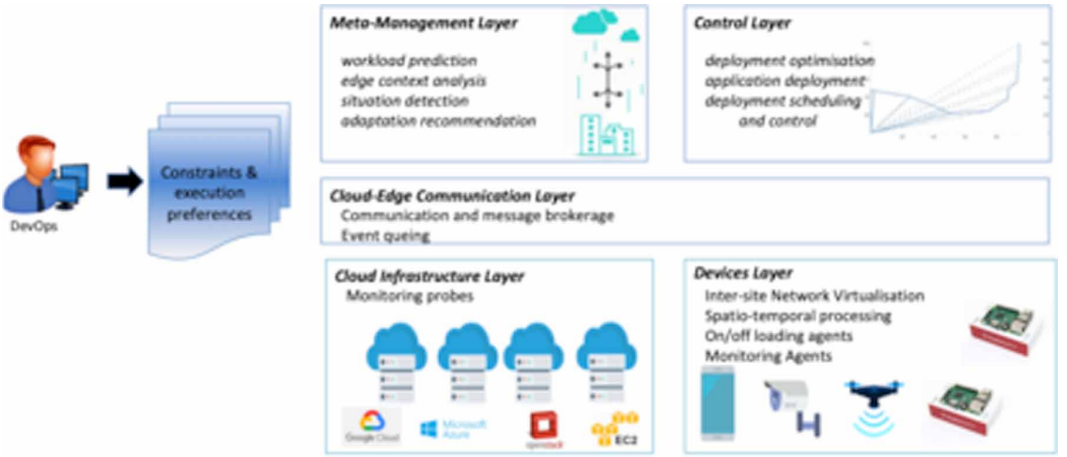
In PrEstoCloud (2020), we focused our research efforts on developing capabilities that address the aforementioned challenges and requirements. The PrEstoCloud framework allows not only the execution of microservices with cloud, fog, and edge devices but also decides, at run-time, the shifting of processing tasks from edge devices to multi-cloud or fog resources and vice-versa. To facilitate this, PrEstoCloud allows DevOps to define constraints and execution preferences about microservices that can be executed either on cloud, fog, or edge resources. PrEstoCloud decides during run-time to offload/onload processing tasks on specific types or instances of edge resources to/from multi-cloud resources. Such decisions are based on the current context of the resources used, the microservices status with respect to QoS and the present and estimated workload.

3.1. Five-Layered Architecture

PrEstoCloud follows a five-layer architecture (Figure 2):

- The *meta-management layer* provides decision logic capabilities required for enhancing the control layer. Components of this layer use as input the situation details, the variation of the data streams and the context of the mobile devices at the extreme edge of the network. This layer recognizes the situations when it is required to recommend, at the appropriate time, the necessary adaptations, such as scaling of functions and on/offloading of microservices.
- The *control layer* contains components responsible for the optimized scheduling of function execution over available resources. The control layer detects available edge resources and selects target resources for deployment and plans fragment scheduling according to the recommendation of the meta-management layer. Optimization involves the examination of a big variability space to

Figure 2. PrEstoCloud architecture



find those resources that satisfy certain business goals (e.g., reduce cloud costs while maintaining an adequate response time).

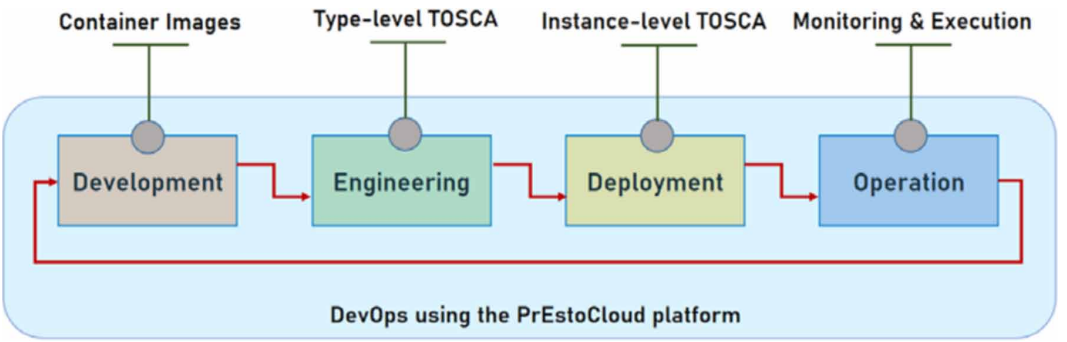
- The *cloud infrastructure layer* realizes the dynamic deployment and scheduling capabilities, according to the instructions of the control layer. The deployment of microservices is handled based on deployment constraints related to different properties like response time, security constraints or any other preferences of the DevOps.
- The *cloud–edge communication layer* contains the inter-site network virtualization technology for coping with the need for connecting resources situated in multi-cloud environments and managing their orchestration and provisioning across different and heterogeneous providers. This layer is also responsible for relaying data streams securely on and off the PrEstoCloud platform and for providing publish and subscribe event brokering capabilities.
- The *devices layer* contains edge devices that can be used as processing nodes. The appropriate agents are installed in each device for (a) network virtualization, (b) spatio-temporal processing in poor connectivity cases in which the communication to the cloud layer is maintained even if only one edge device maintains adequate connection, (c) shifting processing jobs in our out of the edge device, and (d) monitoring agents for aggregating infrastructure and application-level metrics that reveal the status of each edge device.

The framework that was developed based on this architecture provides the capability to run functions by using microservices, specifically, with the deployment of Docker images. Therefore, PrEstoCloud supports the deployment of applications built using the microservice and FaaS paradigms through the use of lambda functions, as well as the deployment of any applications capable of exploiting the distributed deployment paradigm. Deployment blueprints are expressed in an extended topology and orchestration specification for cloud applications specification (TOSCA 2020) that considers edge and FaaS-related constructs. These blueprints are exploited in processing workflows that enable the deployment of microservices.

The microservices added by the DevOps through a UI are deployed on FaaS processing nodes. Processing nodes can be further specified with the use of affinity and anti-affinity constraints, as well as the precedence of deployment of some nodes before others. Affinity constraints indicate that some nodes should be placed in the same geographical region, and if possible, in the same datacenter or rack. Similarly, anti-affinity constraints indicate that nodes should be placed in different geographic areas. Precedence-of-deployment constraints force a particular order in the deployment, enabling nodes that provide programmatic interfaces to be instantiated before the nodes that require them. New processing nodes may be added or removed from the current processing topology without affecting the overall processing structure. This enables PrEstoCloud to timely use of horizontal scaling to optimize the performance of the processing topology according to the processing specifications of the DevOps.

In addition to processing nodes, PrEstoCloud makes use of coordinator nodes, in the form of a Lambda proxy component (i.e., a Traefik instance https://traefik.io/). A coordination node acts as a gateway to access the functionality of microservices and can transmit metrics related to the running processing tasks. The state of processing nodes is continuously monitored by coordinator nodes and control layer components. If a change in the topology is detected, e.g., a node does not respond within a timeout, the platform will still be operational using short-term and medium-term measures. In the short term, a coordinator node will redirect incoming processing tasks to other nodes processing the same task—a necessary prerequisite for the effectiveness of short-term measures is the availability of such nodes. In turn, this implies that the DevOps should require a minimum number of instances for this function, balancing the need for short-term fault-tolerance with the increased cost. In the meantime, the control layer will initiate remedy actions, e.g., the instantiation of a new processing node, aiming to reinstate the balance in the processing topology in the medium term.

Figure 3. PrEstoCloud deployment lifecycle



3.2. Deployment Lifecycle

Figure 3 illustrates the deployment cycle for the intelligent logistics application scenario. These four steps shown are fully supported by the PrEstoCloud framework to allow the DevOps to respond to business requirements.

The cycle begins with the input of the developer and the DevOps and resulting in the actual deployment of the application. We focus and describe how the reconfiguration cycle is realized. Reconfiguration is based on the available VM flavors and edge devices as well as the qualitative and quantitative preferences of the DevOp. Based on this information, the system generates a type-level TOSCA specification of the fragmentation along with a recommended deployment without specific VM and edge instances. The specification takes into consideration constraints such as security constraints or other quantitative or qualitative constraints, e.g., cost, response time, and data sanitization support. The recommendation is forwarded to the control layer for optimization, instantiation of the TOSCA specification and deployment.

3.2.1. Development step

In the development step, system requirements are refined into a complete product design including the hardware and software architecture and detailed descriptions of the system, data and interfaces. Software developers know that the application includes small, self-contained deployable containers, each one acting as an individual function application, working together through APIs, which are not dependent on a particular language, library or framework. In this way, decomposing the whole logistics application into small containers enables us to distribute the computational workload of services among various resources. Besides this, each of the services can be easily developed and operated by different software engineering teams, and hence, this container-based architecture affects both organizational forms of cooperation as well as technological decisions which can be made locally by each team. Resilience to failure is another characteristic of the microservice approach because each application request is divided and translated to different service calls in this software architecture. Therefore, a bottleneck in a specific service operation will not bring the entire system down and it only affects that service. In such a situation, other services are able to carry on processing their requests as usual.

3.2.2. Engineering step

The main output in the engineering step is a type-level TOSCA file by parsing code-level annotations, as well as DevOps preferences and requirements (e.g. cloud provider requirements) expressed in a policy file (not shown for brevity). These requirements are then grouped, and a type-level TOSCA file is produced. The type-level TOSCA includes generic specifications, for example the application topology, relations among all application components, deployment order of application components, constraints such as required resource features for each application component, etc. Afterwards, the

type-level TOSCA file of the logistics application is pushed to the PrEstoCloud repository, whence it is retrieved by the Control Layer in order to calculate the optimal configuration for the application deployment.

3.2.3. Deployment Step

The deployment step of the application is performed by the PrEstoCloud cloud and edge layers. Specifically, an orchestrator automatically generates the instance-level TOSCA converted based on the type-level TOSCA received from the previous step. The instance-level TOSCA includes various characteristics such as the public IP addresses of all host machines automatically provisioned, all required network configurations, inbound and outbound rules for VMs, security policies, components start-up sequence, and similar. When the deployment step is accomplished, VMs are automatically allocated to instantiate all containers running the application. The orchestrator automatically provisions public clouds and edge resources, which provides a wide selection of instance types optimized to fit the logistics use case. Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give the orchestrator the flexibility to choose the appropriate mix of resources for the application. Each of microservice is packaged in Docker containers is deployed on an individual host machine.

3.2.4. Operation Step

This step manages cloud and edge resources. To this end, the operation step exploits available capabilities to monitor and control cloud-based resources, which can be also extended to the edge of the network. This step is responsible for the optimal scheduling of services deployed for the application over all available resources according to the recommendation of Meta-Management Layer. The selection of cloud-based infrastructures is followed by an optimization step, which is necessary for finding the most appropriate alternatives which can satisfy certain business goals such as cloud infrastructure cost reduction while maintaining a satisfactory response time.

The monitoring system provided by the PrEstoCloud framework plays a key role to track the execution environment. The Monitoring Agent is able to continuously measure a set of necessary metrics related to both infrastructure and application performance. Infrastructure-specific metrics are CPU, memory, disk, network, etc. Furthermore, application-related metrics represent the information about the status of the application such as service response time.

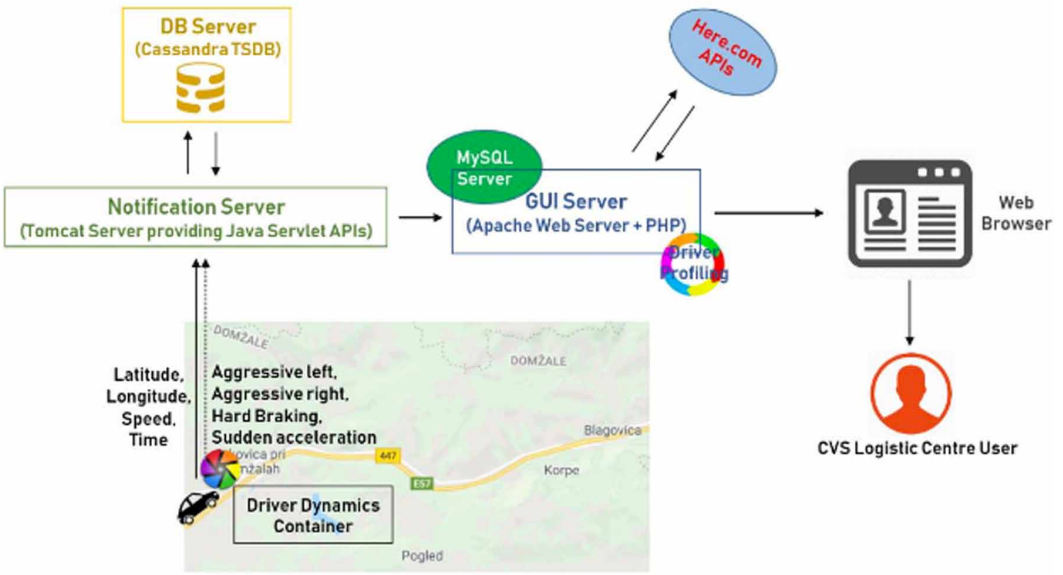
4. EVALUATION

4.1. Use Case

To illustrate PrEstoCloud's capability to support auto-scaling in deployments of microservices, we use as a case study the microservice-based logistics application, which is divided into four fragments: driver dynamics service, notification server, DB server, and GUI server (Figure 4). The driver dynamics service running on the unit installed in the vehicle (edge) receives data from sensors and recognizes unexpected driving dynamics, e.g., sudden acceleration, hard braking, and aggressive turns. If there is such an occurrence, the service will instantly send a run-time message to the notification server, which is running on the RPU. Moreover, the driver dynamics service transmits some sensor data periodically to the notification server, such as the vehicle's speed and GPS information, which is helpful in knowing where the vehicle is located. All information is available for the logistic center end user via web-based GUI. The GUI server is a web server that processes all incoming requests over HTTP sent from end users (e.g., logistic center end users using web browsers) and delivers web page contents to them.

The notification server running on the RPU receives all telematics messages sent by driver dynamics services. Messages are stored in the DB server, a database used to store the time-series

Figure 4. Logistics application microservices



data, e.g., the routes where vehicles are moving, the places where driving dynamics happened. The GUI server, which is deployed on a different cloud (Google Cloud), shows end users all information stored in the DB server. Facilitating multi-cloud resource allocation, PrEstoCloud allows the DB server to be deployed on one cloud, for example, Amazon EC2, which guarantees favorable service-level agreement (SLA) terms for the DB server and web-based GUI server on another cloud, e.g., Google Cloud Platform, which offers optimal pricing.

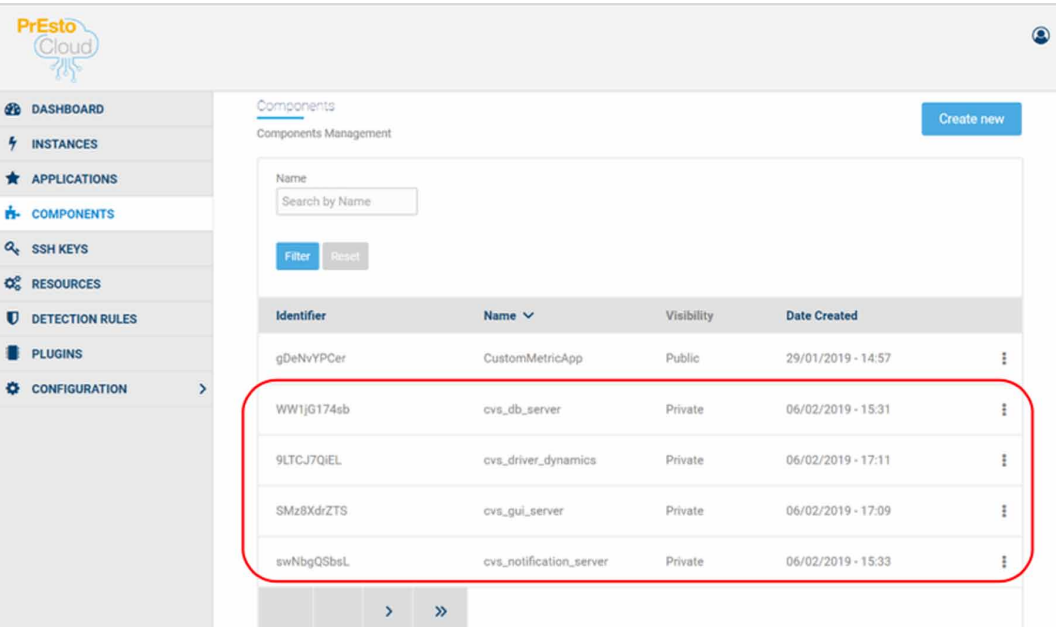
Each RPU receives messages from vehicles within its transmission range. According to the number of vehicles in every geographic region, heterogeneous RPUs with different computing capacity may be allocated. The optimized microservice deployment of Notification functions for each area, avoids resource under-provisioning and over-provisioning while adhering to the processing and functional requirements of the fragment, e.g., that the DriverDynamics fragment needs to be deployed before the Notification Server. Then, the cheapest VM having at least one CPU, 4 GB of RAM, and 4 GB of disk space is requested. According to a DevOps-configured rule, e.g., whenever the CPU load is increased by more than 70% or decreased to less than 30% over two minutes, the framework tries to add or remove notification server instances, respectively.

In some cases, a new RPU should be allocated because of a movement of the vehicle from a geographic location to a new place. In this case, the notification server running on the current RPU, which provides the service is not able to offer favorable QoS required for the intelligent logistics application. Therefore, the Notification Server needs to be migrated from the current RPU to another RPU near the vehicle. A different reason for violations of QoS constraints may also be the situation where the RPU is overloaded due to an increasing workload. In such a case, offloading the notification server from the current RPU to the cloud or vice versa is also necessary.

4.2. Deployment Using PrEstoCloud

Figure 5 shows the PrEstoCloud UI dashboard through which, the DevOps specifies the application microservices. Once specified, the DevOps can start an instance of the application which will be automatically deployed on resources already defined on the dashboard. For the specific application,

Figure 5. Logistics application specification on PrEstoCloud GUI



four types of containerised components as specified: cvs_db_server, cvs_notification_server, cvs_gui_server, cvs_driver_dynamics_arm / cvs_driver_dynamics_x86.

In order to specify each microservice, various parameters need to be completed, such as name of component, architecture (X86 or ARM), name of Docker registry, name of Docker image, minimum execution of hardware requirements (e.g. vCPUs, RAM, storage, etc.), health check API, environmental variables, exposed interfaces (port numbers, etc.) and required interfaces from other components (Figure 6).

To complete the application deployment specification, the architecture should be defined. To this end, the connections among all components are required to be precisely described through exposed interfaces and required interfaces. As mentioned before, such interfaces are already defined when we created each component. Figure 7 depicts a graph which is automatically generated via the PrEstoCloud UI based on the connections defined for each component.

Figure 6. Microservice parameters

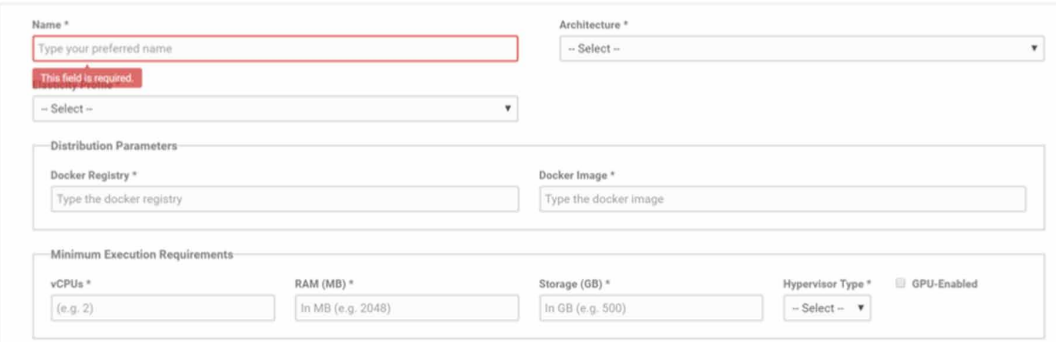
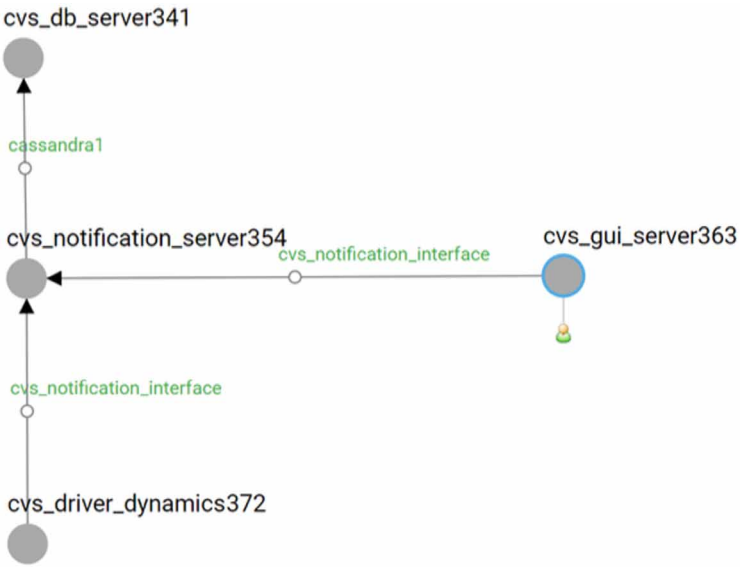


Figure 7. Microservice deployment graph



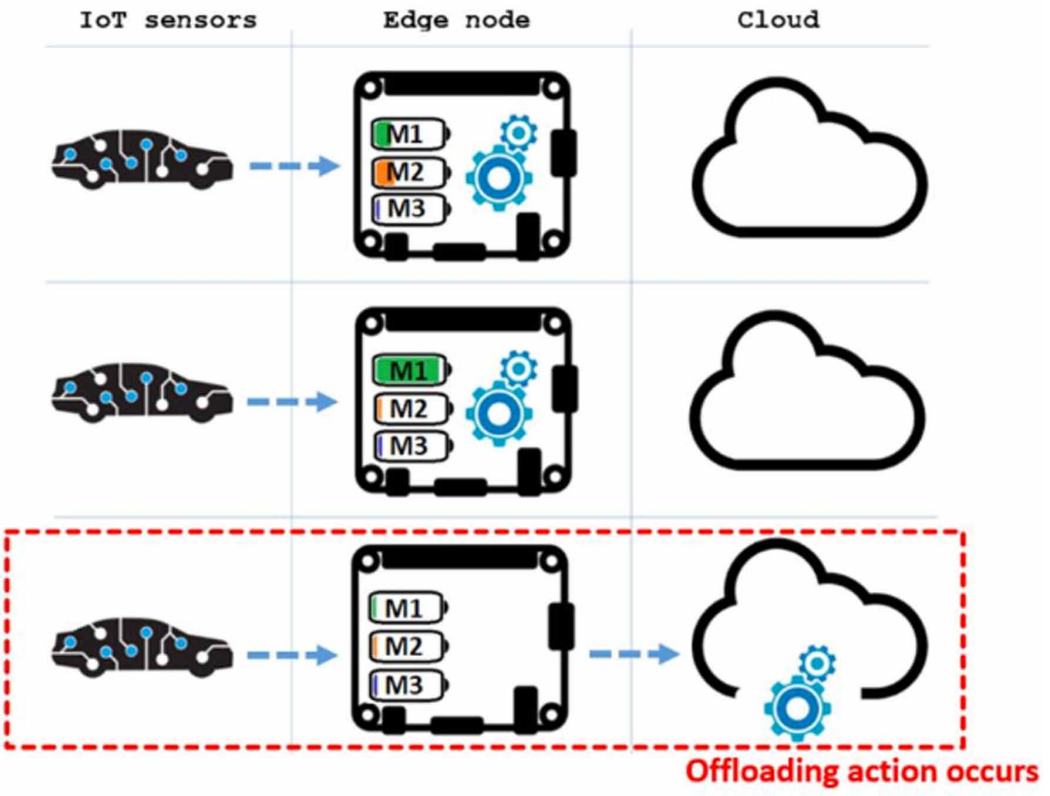
Upon the successful deployment, the Cloud–edge communication layer of PrEstoCloud makes the DB Server accessible only from the Notification Server in the network, not from other hosts on the Internet. These security concerns, also for other components, are automatically addressed by the packet filtering service. For the deployment of the logistics application, three cloud providers have been used: Amazon, Azure and ARNES (Academic and Research Network of Slovenia). The DB Server is deployed on the Amazon cloud infrastructure, the Notification Server is deployed on the Azure cloud infrastructure, and the GUI Server is deployed on the ARNES cloud infrastructure.

Figure 8 shows the scenario how the Devices layer of PrEstoCloud performs at runtime a typical example of auto-scaling: off-loading. Because the Driver Dynamics Container is running on the edge, it is useful for analyzing the telematics data very fast. In this case, the container image named cvs_driver_dynamics_arm is exploited since the edge nodes use ARM processor architecture. If the limited-resource edge node is not able to provide the service (e.g., any of the monitoring metrics may show that an abnormal situation is going to happen, for example the free disk capacity or computation power on the edge node is not available anymore), the PrEstoCloud solution offloads the service from the edge node to the cloud infrastructure.

Monitoring data are communication through the Cloud-edge communication layer of PreStoCloud, which uses the MQTT protocol. MQTT is an IoT connectivity protocol, designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required or network bandwidth needs to be considered so it has been used in sensors communicating to the Broker.

In order to have a zero-failure rate, the start time of the Driver Dynamics Container needs to be taken into consideration when it should be offloaded from one resource to another one. In essence, any container termination on the source node before the time when the new container instance on the destination node would be ready to offer its own service means the death of this specific service for a while. Therefore, either success or failure status of start and stop requests should be observed to reach a fortunate on/offloading operation. In this regard, a numerical value will be returned by the developed On/Offloading Server that implies if the request (whether start or stop) has been successfully executed or not.

Figure 8. Microservice on/offloading



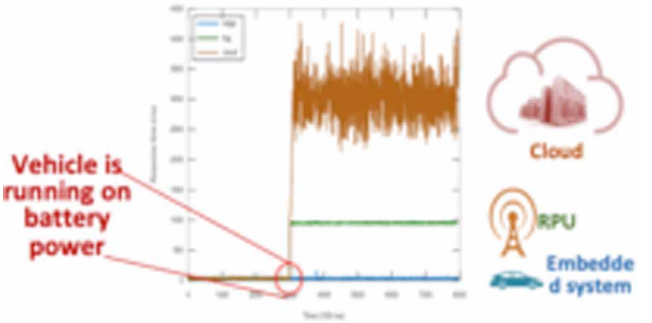
4.3. Results

To evaluate PrEstoCould's auto-scaling capabilities, we measured the average, 99th percentile, and standard deviation of application response time during an offloading action. Here, the response time is the time period from sensor data acquisition to the time when driver dynamics microservice, a function of the intelligent logistics app, detects if an unexpected event happens or not. Three different types of infrastructures were exploited to host the driver dynamics microservice to evaluate the importance of its placement, including embedded system, RPU, and centralized cloud. The custom embedded system called MACH (Motorhome AI Communication Hardware; Taherizadeh et al., 2018) developed by the Jožef Stefan Institute (JSI) is installed in the vehicle at the edge of the network. The MACH has a 4-core 1.4 GHz CPU, VideoCore-IV GPU, 1 GB of RAM, and 20 GB of storage. The RPU as a fog computing resource belonged to the Academic and Research Network of Slovenia (ARNES) with a 1-core 2397 MHz CPU, 4 GB of RAM, 10 GB of storage, and 1000 MBps bandwidth. The cloud server also had a 4-core 2659 MHz CPU, 16 GB of RAM, 80 GB of storage, and 1000 MBps bandwidth. Each experiment was repeated five times to obtain verified results.

A 25 km trip, which took 13 minutes and 16 seconds, was selected to gather sensor data from an ordinary vehicle. The vehicle's sensor measurements were sampled at the rate of 10 Hz, so each sample was recorded every 100 ms. Therefore, there were nearly 8,000 samples from the whole trip.

We evaluated the extent to which fog and cloud computing can help the logistics application improve the response time when PrEstoCloud handles an auto-scaling action (Figure 9). This action was performed to re-deploy the driver dynamics microservice from the edge node to either the fog or the cloud at time = 3,000 ms when the electric vehicle is running out of battery power, and hence the

Figure 9. Intelligent logistics application



embedded system is not able to provide computing operations on the edge. Re-deploying the driver dynamics microservice from the edge resource to the fog or cloud helps save the battery as much as possible. Table 2 shows the response time provided in three different experiments, including edge, fog, and cloud. In the second and third experiments, the driver dynamics microservice was running respectively on the fog and cloud after the time when it was offloaded from the edge node.

When the driver dynamics microservice was offloaded from the edge node, the average and 99th percentile of response times for the fog test 948.33 ms and 954.60 ms, respectively, whereas these parameters were 3,013.23 ms and 3,901.52 ms in the cloud test. This means that the difference in average response times between fog and cloud is almost two seconds, while the difference in the 99th percentile of response times between fog and cloud is nearly three seconds. Moreover, the response time offered by the fog infrastructure was appropriately steady during the experiment since the standard deviation of response time was less than 3 ms, whereas the standard deviation of response time with the cloud was more than 364ms. Therefore, the fog node provided a more stable quality of computing resource in comparison to the cloud.

5. DISCUSSION

To avoid vendor lock-in and increase the choices available to DevOps, some of the available fog computing frameworks allow cross-cloud deployments as well as orchestration of deployments on fog resources using open standards such as TOSCA. Although some work has been done to extend TOSCA to support deployment on edge resources, the standard is mostly still restricted to deployments in conventional cloud resources. To facilitate the adoption of fog computing implementations, the availability of up-to-date standards supporting widely adopted deployment technologies, such as Docker containers, is essential. Further efforts should be put in extending these standards to address fog computing requirements and concerns.

Fog computing frameworks are increasingly more capable of dealing with the dynamicity of fog-deployed applications and services. Specifically, adjusting and adapting the computing infrastructure to

Table 2. Response time offered by the Driver Dynamics Service on edge, fog and cloud resources

Time (ms)	Edge	Fog	Cloud
Average response time	35.64	948.33	3013.23
99th percentile of the response time	42.5	954.60	3901.52
Standard deviation of response time	3.52	2.62	364.50

changing data flows and producing automatically optimal orchestrations of cloud and fog-resources is a prominent trend, which ameliorates the need for the DevOps to monitor the deployment continuously and make the necessary changes manually. Still, further work is needed to fully support multi-cloud and cross-resource optimization of deployments of data-intensive applications. In this direction, PrEstoCloud allows extending the deployment and networking capabilities to the extreme edge of the network, enabling efficient processing of the data produced at the edge. Depending on the current situation, PrEstoCloud foresees to make an optimized placement of the application as well as of parts of an application, to find the right balance between edge and cloud usage. Moreover, PrEstoCloud allows for scaling and adapting the deployed applications in real-time, based on the existing and also on the anticipated processing load. Still, further research is needed to support the self-adaptivity of fog infrastructures fully. To this end, new methods and tools are needed that can detect ahead of time needed for changes by analyzing all available contextual information, generating the most efficient reconfiguration and defining the optimal redeployment of the running processing tasks. Such methods and tools will become more and more useful as their placement is feasible closer to edge nodes.

The edge-processing capability of fog computing frameworks is challenged by the serverless computing trend. PrEstoCloud combines edge computing and serverless paradigms to provide an innovative solution for FaaS-enabled applications. For example, it enables IoT applications that perform analytics to host machine learning models within an edge network, so that applications can exploit the model close or at the edge. Every edge location deploys the machine learning model as a serverless function. The edge layer simplifies the deployment experience, while serverless streamline the developer experience. Functions can be provided faster, leading to increased flexibility and greater availability of the deployed application components.

Continuous monitoring and analysis of the status of fog resources and deployed applications or application parts is a challenging task. Monitoring probes need to be capable of distributed deployment. Moreover, analysis of probed data needs to be done at or close to the edge due to network latency and throughput limitation issues. For example, instead of having a security camera stream its video and audio feed up to the cloud to be analyzed for certain situations, that analysis can be done within the camera itself. Clearly, a more advanced, cloud-based analysis may still be needed, but it is likely to be on a much smaller segment of the camera data. To analyze monitoring data on edge resources, efficient machine learning methods are needed, with minimal computing and memory requirements. Such a distributed monitoring and analysis capability can have a minimal footprint on the network traffic while leveraging effective decision making for the overall adaptation and optimization of the fog infrastructure as well as help in maintaining high QoS.

The traditional centralized cloud computing continues to remain a significant part of fog computing frameworks. Function and data portability, infrastructure monitoring as well as orchestration methods employed in the centralized cloud environment cannot be useful in the context of interconnected, heterogeneous fog computing resources, in which decision spaces become larger, increasingly complex and more dynamic. Next-generation fog computing frameworks will enable the smart deployment of applications on a combination of cloud, edge, and extreme edge resources and are expected to employ advanced methods for dealing with the dynamicity of FaaS applications and fog infrastructures.

6. CONCLUSION

In this paper, we described the details of the PrEstoCloud framework that introduces advanced methods that deal with the dynamicity of FaaS applications and fog infrastructures. The main three distinctions among the surveyed fog frameworks and PrEstocloud are the following. First, PrEstoCloud enables the use of multi-private or public clouds by considering which is the most efficient cloud resource to use in each case. Second is computation at the extreme edge on a per-device level; further, benefiting from the computational resources of cameras and mobile devices (wherever possible) is something that PrEstoCloud aspires to deliver which is not available in similar solutions like AWS

Greengrass. Finally, PrEstoCloud allows the application developer and the DevOps to define their constraints properly but also provide hints about application fragments that can be executed either on cloud resources, edge resources, or both. From that point on, PrEstoCloud can decide during run-time to offload/onload processing tasks on edge resources to/from cloud resources based on the current state of the resources used, the application fragments status, and the current and predicted workload.

Funding: This research was funded by the European Commission, grant number XXX.

ACKNOWLEDGMENT

We would like to thank all partners of the XXX project who has contributed with ideas towards the development of the PrEstoCloud framework.

REFERENCES

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., & Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing, ser. HUC '99*. Springer. doi:10.1007/3-540-48157-5_29

AWS Greengrass. (n.d.). https://aws.amazon.com/greengrass/

AWS Wavelength. (n.d.). https://aws.amazon.com/wavelength

Azure IoT Edge. (n.d.). https://azure.microsoft.com/en-us/services/iot-edge/

Carroll, N. (2015). Modelling the dynamics of trust across a cloud brokerage environment. *Information Resources Management Journal*, *28*(1), 17–37. doi:10.4018/irmj.2015010102

Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. (2019). The Rise of Serverless Computing. *Communications of the ACM*, *62*(12), 44–54. doi:10.1145/3368454

Chiang, M., & Zhang, T. (2016). Fog and IoT: An overview of research opportunities. *IEEE Internet of Things Journal*, *3*(6), 854–864. doi:10.1109/JIOT.2016.2584538

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems, ser. EuroSys '11*. Salzburg, Austria: ACM. doi:10.1145/1966445.1966473

Cloudify. (n.d.). https://cloudify.co/

Di Martino, B., Esposito, A., & Damiani, E. (2019). Towards AI-Powered Multiple Cloud Management. *IEEE Internet Computing*, *23*(1), 64–71. doi:10.1109/MIC.2018.2883839

Duplyakin, D., Marshall, P., Keahey, K., Tufo, H., & Alzabarah, A. (2013). Rebalancing in a multi-cloud environment. In *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*. ACM. doi:10.1145/2465848.2465854

Gandhi, A., Dube, P., Karve, A., Kochut, A., & Zhang, L. (2014). Adaptive, model-driven autoscaling for cloud applications. *11th International Conference on Autonomic Computing (ICAC 14), USENIX, 2014*, 57– 64.

Google Cloud IoT Core. (n.d.). https://cloud.google.com/iot-core

Guo, T., Sharma, U., Wood, T., Sahu, S., & Shenoy, P. (2012). Seagull: Intelligent cloud bursting for enterprise applications. *Proceedings of the 2012 USENIX Annual Technical Conference, ser. ATC'12*.

Gusev, M., Koteska, B., Kostoska, M., Jakimovski, B., Dustdar, S., Scekic, O., Rausch, R., Nastic, S., Ristov, S., & Fahringer, T. (2019). A Deviceless Edge Computing Approach for Streaming IoT Applications. *IEEE Internet Computing*, *23*(1), 37–45. doi:10.1109/MIC.2019.2892219

Hao, Z., Novak, E., Yi, S., & Li, Q. (2017). Challenges and software architecture for fog computing. *IEEE Internet Computing*, *21*(2), 44–53. doi:10.1109/MIC.2017.26

Iorga, M., Feldman, L., Barton, R., Martin, M. J., Goren, N., & Mahmoudi, C. (2018). *The NIST Definition of Fog Computing*. NIST Special Publication 800-191.

Kailasam, S., Gnanasambandam, N., Dharanipragada, J., & Sharma, N. (2013, November). Optimizing ordered throughput using autonomic cloud bursting schedulers. *Transactions on Software Engineering*, *39*(11), 1564–1581. doi:10.1109/TSE.2013.26

Kokkinos, P., Varvarigou, T. A., Kretsis, A., Soumplis, P., & Varvarigos, E. A. (2013). Cost and utilization optimization of amazon ec2 instances. *2013 IEEE Sixth International Conference on Cloud Computing*, 518–525. doi:10.1109/CLOUD.2013.52

Kovachev, D., Yu, T., & Klamma, R. (2012). Adaptive computation offloading from mobile devices into the cloud. *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, 784–791. doi:10.1109/ISPA.2012.115

Leitner, P., Rostyslav, Z., Gambi, A., & Dustdar, S. (2013). A framework and middleware for application-level cloud bursting on top of infrastructure-as-a-service clouds. *Proceedings of the 6th IEEE/ACM International Conference on Utility and Cloud Computing*. doi:10.1109/UCC.2013.39

Liu, Y., Fieldsend, J. E., & Min, G. (2017). A framework of fog computing: Architecture, challenges, and optimization. *IEEE Access: Practical Innovations, Open Solutions*, *5*, 25445–25454. doi:10.1109/ACCESS.2017.2766923

Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014, December). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, *12*(4), 559–592. doi:10.1007/s10723-014-9314-7

Mahesh, S., Landry, B. J., Sridhar, T., & Walsh, K. R. (2011). A decision table for the cloud computing decision in small business. *Information Resources Management Journal*, *24*(3), 9–25. doi:10.4018/irmj.2011070102

Nebiolo. (n.d.). https://www.nebbiolo.tech/

OpenWhiskLean. (n.d.). https://github.com/kpavel/incubator-openwhisk/tree/lean

Papageorgiou, N., Apostolou, D., Verginadis, Y., & Mentzas, G. (2019). Fog Context Analytics. *IEEE Instrumentation & Measurement Magazine*.

PreStoCloud. (n.d.). https://gitlab.com/prestocloud-project

Qian, H., & Andresen, D. (2014). Jade: An efficient energy-aware computation offloading system with heterogeneous network interface bonding for ad-hoc networked mobile devices. *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. doi:10.1109/SNPD.2014.6888703

Qu, C., Calheiros, R. N., & Buyya, R. (2018). Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys*, *51*(4), 73. doi:10.1145/3148149

Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H., & Kozuch, M. A. (2012). Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third Symposium on Cloud Computing, ser. SoCC '12*. ACM. doi:10.1145/2391229.2391236

Taher, C., Mallat, I., Agoulmine, N., & El-Mawass, N. (2019). An IoT-Cloud Based Solution for Real-Time and Batch Processing of Big Data: Application in Healthcare. In *2019 3rd International Conference on Bio-engineering for Smart Technologies (BioSMART)*, (pp. 1-8). IEEE.

Taherizadeh, S., Stankovski, V., & Grobelnik, M. (2018). A Capillary Computing Architecture for Dynamic Internet of Things: Orchestration of Microservices from Edge Devices to Fog and Cloud Providers. *Sensors (Basel)*, *18*(9), 2938. doi:10.3390/s18092938 PMID:30181454

Tamrakar, K., Yazidi, A., & Haugerud, H. (2017). Cost efficient batch processing in Amazon cloud with deadline awareness. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, (pp. 963-971). IEEE. doi:10.1109/AINA.2017.170

Tordsson, J., Montero, R. S., Moreno-Vozmediano, R., & Llorente, I. M. (2012). *Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers*. Academic Press.

TOSCA. (n.d.). https://www.oasis-open.org/committees/tosca/

Trihinas, D., Tryfonos, A., Dikaiakos, M. D., & Pallis, G. (2018). Devops as a service: Pushing the boundaries of microservice adoption. *IEEE Internet Computing*, *22*(3), 65–71. doi:10.1109/MIC.2018.032501519

Van Eyk, E., Toader, L., Talluri, S., Versluis, L., Uță, A., & Iosup, A. (2018). Serverless is more: From paas to present cloud computing. *IEEE Internet Computing*, *22*(5), 8–17. doi:10.1109/MIC.2018.053681358

Wang, N., Varghese, B., Matthaiou, M., & Nikolopoulos, D. (2017). ENORM: A framework for edge node resource management. *IEEE Transactions on Services Computing*.

Wen, Y., Wang, Z., Zhang, Y., Liu, J., Cao, B., & Chen, J. (2019). Energy and cost aware scheduling with batch processing for instance-intensive IoT workflows in clouds. *Future Generation Computer Systems*, *101*, 39–50. doi:10.1016/j.future.2019.05.046

Zhang, F., Tang, X., Li, X., Khan, S., & Li, Z. (2019). Quantifying cloud elasticity with container-based autoscaling. *Future Generation Computer Systems*, *98*, 672–681. doi:10.1016/j.future.2018.09.009

Zhang, W., Xu, L., Duan, P., Gong, W., Liu, X., & Lu, Q. (2014). Towards a high speed video cloud based on batch processing integrated with fast processing. In *2014 International Conference on Identification, Information and Knowledge in the Internet of Things*, (pp. 28-33). IEEE. doi:10.1109/IIKI.2014.13

Yiannis Verginadis is an Assistant Professor at Athens University of Economics and Business and a senior researcher at the Institute of Communication and Computer Systems (ICCS). He has more than ten years of experience in R&D projects in several research areas such as: management of information systems, software engineering, workflow management, electronic government, electronic commerce and cloud computing. He holds Diploma and Doctoral degrees in Electrical and Computer Engineering from the National Technical University of Athens, Greece (2001 and 2006).

Dimitris Apostolou is Associate Professor in the Informatics Department of the University of Piraeus and Senior Researcher at the Institute of Communication and Computer Systems. His research concerns decision support, knowledge management and intelligent information systems. He is a member of the IEEE Computer Society and various national scientific and professional associations.

Salman Taherizadeh obtained his PhD in Computer Science at the University of Ljubljana in 2018.

Giannis Ledakis is Software Engineer at UBITECH.

Gregoris Mentzas is full Professor of Management Information Systems, School of Electrical and Computer Engineering, National Technical University of Athens and Director of the Information Management Unit (IMU). His area of expertise is information technology management and his research concerns knowledge management, semantic web and social computing. His recent research concerns big data management in multi-cloud environments and prescriptive analytics in Industry 4.0 cases especially for predictive maintenance. He has published 4 books and more than 200 papers in international peer-reviewed journals and conferences, has 5 best papers awards, sits on the editorial board of five international journals and has served as (co-)Chair or Program Committee Member in more than 55 international conferences.

Andreas Tsagkaropoulos is a PhD student in the Information Management Unit of the school of Electrical and Computer Engineering. He graduated in 2016 from the School of Electrical and Computer Engineering of the National Technical University of Athens (Greece), and his Diploma thesis was about enhancing the security of a personal cloud storage solution. He is working as a researcher in the Institute of Communications and Computer Systems, a technology research institute. His research interests include cloud application modelling and adaptation, serverless applications and cloud and edge infrastructure.

Nikos Papageorgiou holds a Diploma degree in Electrical and Computer Engineering (1998) and an MSc on "Engineering-Economic Systems" (2007) from the National Technical University of Athens, Greece. He also holds a Degree of Proficiency in English from Cambridge University and he is member of the Technical Chamber of Greece.

Fotis Paraskevopoulos is CEO of Existanze - #connectingdots, and currently he is a PhD candidate in the School of Electrical & Computer Engineering at NTUA. His current research interests include software engineering, knowledge management, semantic web, ontology-based modeling and collaboration services.