Fog Native Architecture: Intent-Based Workflows to Take Cloud Native Towards the Edge

Merlijn Sebrechts*, Bruno Volckaert*, Filip De Turck*, Kun Yang†, Mays AL-Naday†
*IDLab, Department of Information Technology (intec), Ghent University - imec, Belgium
†School of Computer Science and Electronic Engineering, University of Essex, UK

***Abstract*—The cloud native approach is rapidly transforming how applications are developed and operated, turning monolithic applications into microservice applications, allowing teams to release faster, increase reliability, and expedite operations by taking full advantage of cloud resources and their elasticity. At the same time, "fog computing" is emerging, bringing the cloud towards the edge, near the end user, in order to increase privacy, improve resource efficiency, and reduce latency. Combining these two trends, however, proves difficult because of four fundamental disconnects between the cloud native paradigm and fog computing. This article identifies these disconnects and proposes a fog native architecture along with a set of design patterns to take full advantage of the fog. Central to this approach is turning microservice *applications* into microservice *workflows*, constructed dynamically by the system using an intent-based approach taking into account a number of factors such as user requirements, request location and available infrastructure and microservices. The architecture introduces a novel softwarized fog mesh facilitating both inter-microservice connectivity, external communication, and end-user aggregation. Our evaluation analyses the impact of distributing microservice-based applications over a fog ecosystem, illustrating the impact of CPU and network latency and application metrics on perceived Quality of Service of fog native workflows compared to the cloud. The results show the fog can offer superior application performance given the right conditions.**

***Keywords*—fog native, cloud native, fog computing, edge computing, microservices, intent-based management, orchestration, compute-network softwarization, service management**

I. Introduction

The cloud native paradigm advocates for developing applications and network services to run intrinsically in the cloud, rather than merely transitioning to it [1], [2]. The objective is to realize applications at scale and provide capabilities including dynamic scaling, automatic recovery and seamless roll-out. This requires turning monolithic applications into *microservice applications* [3] by decomposing them into self-contained components interconnected by Application Programming Interfaces (APIs). The added complexity of managing microservices has been widely studied [4], [5], [6]. The considerable increase in message exchange between microservices, however, has been largely overlooked because it has not posed a major challenge given cloud providers' tight control over internal network bandwidth and latency.

At the same time, more and more companies are combining cloud applications with edge computing [7]. The Netflix Open Connect program, for example, invites ISPs to place Netflix caching servers in the edge, in order to increase user experience and decrease strain on the network. These capabilities are opened up to a much broader industry by fog computing, which extends the cloud towards the edge. This creates a new economic market where even small players can run applications over a mixture of cloud and edge resources. This enables application developers to increase privacy [8] and reducing latency [9], and helps ISPs to ensure more efficient resource usage. On a technical level, this is enabled by providing both cloud and edge resources in a unified platform such as the OpenFog reference architecture [10] shown in Figure 1.

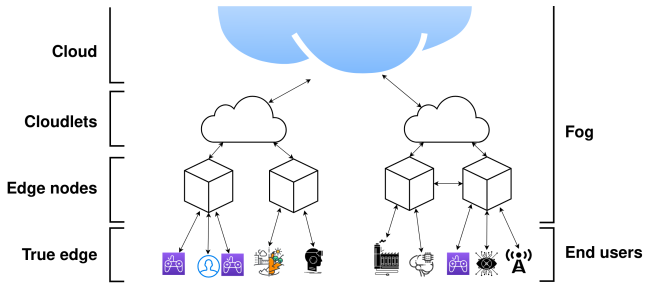

Fig. 1. The OpenFog reference architecture showing a three-tier fog connected by a softwarized network controllable by management functions.

Taking full advantage of the fog requires an approach similar to cloud native paradigm. Naively applying this paradigm and tools to the fog, however, results in inefficient resource utilization and sub-optimal compute distribution; potentially negating the proximity benefits of the fog. These possibly counter-productive results are caused by a number of cloud native assumptions which do not always hold true in the fog.

- The cloud is relatively homogeneous and seamlessly hides the specifics of the underlying infrastructure from the end user, while the fog is inherently heterogeneous, accompanied with **operational complexity**.
- While clouds have an overabundance of relatively cheap resources, **resource constraints** become more apparent closer towards the edge of the fog.
- While clouds offer reliable low-latency and high-bandwidth communication between internal nodes, fog nodes are fully distributed and connected by a network with **highly variant latency and bandwidth**.
- The cloud is central with a relatively limited number of geographical locations, while the fog is **dispersed** with a much higher number of geographical compute locations offered by edge tiers.

Current efforts applying cloud native technologies to the edge do not fully address these issues. KubeFed, for example, allows creating federations of Kubernetes clusters, but is focused on the cloud instead of the fog. KubeEdge aims to shrink Kubernetes to fit on edge nodes, but does not address the fog's operational complexity and dispersion.

This paper identifies four fog native challenges and proposes an architecture that instigates a paradigm shift by defining applications as *microservice workflows*, constructed dynamically using *intent-based matching* of user requirements. This microservice workflow is an evolution from Service Function Chaining of Virtual Network Functions (VNFs), addressing its limitation of hard dependencies. This approach also provides better alignment with cloud-native norms and offers uniform representation of 'logic-execution' elements in the fog. Moreover, the architecture introduces a novel *fog mesh* enabling seamless internal/external communication and flexible grouping of end-users. The architecture is evaluated in terms of distributing workflows over the fog to provide a baseline assessment on the impact of heterogeneity as well as the characteristics of microservices in a workflow on QoS and resource utilization.

This paper starts off by identifying four disconnects between cloud native and the fog in Section II. Section III introduces the fog native architecture addressing these challenges. Section IV outlines a pathway towards implementation through a sample use case. Section V provides a baseline assessment of workflow performance in the fog. Finally, Section VI draws conclusions and outlines future work.

II. Disconnect between cloud native and the fog

This section identifies four key incompatibilities between cloud native and the fog. They stem from four assumptions about the underlying infrastructure and context that do not hold true in the fog.

A. Operational Complexity

Clouds present themselves as relatively homogeneous offerings, allowing developers to reason about what products and infrastructure to use for building an application. As a result, cloud applications are designed as rigidly connected microservices described using desired-state models [2]. Although these descriptions often use over-simplified assumptions of underlying operational complexity, this is not an issue in the relatively homogeneous-looking cloud.

The fog, however, is inherently heterogeneous from a developer standpoint. Providers cannot abstract away the underlying complexity because nodes have varying capabilities and availability of functionality is highly dependent on location, fog tier, and current resource usage [7]. Designing applications for a common denominator risks losing out on an untapped wealth of useful-but-not-ubiquitous features of fog nodes. Moreover, since resources in the fog are not infinitely scalable, design-time assumptions about infrastructure availability might not hold true anymore when the application is deployed.

B. Resource constraints

Clouds offer the illusion of infinite capacity at a relatively cheap price. Consequently, cloud native technologies do not necessarily optimize resource usage. Service meshes, for example, typically duplicate the number of containers needed to run a microservice application [4]. These meshes are dedicated infrastructure layers that manage communication between microservices to improve observability, control and security of inter-microservice communication. They commonly use a "sidecar" approach in which each containerized microservice is accompanied by a containerized proxy which acts as an intermediate in all communications for that microservice. This approach works without code changes to the microservices themselves, but introduces significant resource overhead.

These inefficiencies are generally tolerable in a cloud environment with abundant resources. The fog, however, has strict constraints on resource consumption [7] that may not tolerate such waste.

C. Latency

Clouds offer reliable internal networks which enable low-latency and high-bandwidth communication between nodes. As a result, many cloud schedulers do not consider inter-service dependencies and network latency when placing services because their impact on performance is often negligible. This is not the case in the fog, however, due to its heterogeneous and turbulent internal network latency. As the evaluation in Section V shows, not taking into account inter-service dependencies and network parameters during scheduling of microservices in the fog results can negate much of the locality benefits of the fog.

A second issue arises in common cloud native patterns such as API gateways [5], which sit between a client and a microservice application, acting as the ingress endpoint for all external connections [6]. This pattern enables microservice applications to use asynchronous communication internally, and centralizes concerns such as compression, response aggregation and authorization [5]. However, given the aim of the fog is to bring compute closer to the edge, centralized gateways are antithetical to it, negating the latency [7] and privacy [8] benefits of keeping data and compute at the edge.

D. Dispersion

The cloud provides a relatively limited number of geographical compute locations, often called "regions". As a result, developers normally manually plan the geographical distribution of their microservices based on (predicted) user demand.

The fog, however, has a very high number of geographical compute locations [7], making it difficult to manually select where an application needs to run. Moreover, automatically distributing fog native applications based on individual end user requests can be challenging because of the sheer volume and diversity of them. Although this might be feasible for relatively static requirements such as a home automation system, it falls short at the scale and variability of applications such as video streaming and social networking.

III. Designing a fog native architecture

This section introduces a number of fog native design patterns which fit together into an architecture that tackles the aforementioned challenges of bringing the cloud native paradigm to the fog.

A. Overview

The proposed fog native architecture instigates a paradigm shift where developers no longer define *what* should be deployed. Instead, they define *the desired behavior* of the application using *intents*. The system then dynamically composes and deploys *workflows of microservices*, taking into account a number of factors such as user location, network topology, available infrastructure, and existing services. This *intent-based workflow construction* permits much larger flexibility than the traditional desired-state approach because the orchestrator can change application topology and interchange application components depending on where user demand is for certain functionality and what infrastructure is available at that location.

Figure 2 shows an overview of the proposed fog native architecture, consisting of three conceptual components.

- The ***discovery service*** is a registry of individual microservice templates enriched with metadata about their functionality, characteristics and dependencies. This registry also tracks the offerings of the fog provider and the functionality of already deployed microservices and workflows.
- The ***workflow manager*** responds to user requirements and either identifies a running workflow or uses intent-based workflow construction, explained further in Section III-B, to create a new workflow from scratch to satisfy the user request.
- The ***fog mesh*** enables communication both between microservices, and to end-users. It consists of a set of interlinked service proxies, each of which provides softwarized network services to a regional cluster of microservices, saving up valuable resources by removing the one-to-one relationship between proxy and microservice.

The remainder of this section explains in detail the four key innovations of this fog native architecture compared to a traditional cloud native approach.

B. Intent-based workflow construction

As Section II-A explains, the fog's heterogeneity escalates the complexity for developers to describe their application's desired state. To tackle this challenge, we propose to dynamically create workflows based on user *intents*. This gives the system flexibility to dynamically update the desired state based on user demand, location and available infrastructure. Following this approach, user requests provide a description of the desired functionality, constraints, and tolerances. Developers advertise microservice templates annotated with rich metadata describing the functionality of each component, its infrastructure requirements and its dependencies. End users request certain functionality, for example using using semantic-based addressing similar to that proposed by Al-Naday et al. [11]. The system parses this request and matches the intents with microservices, active workflows, fog offerings and VNFs. The system then either directs the request towards an existing workflow or dynamically constructs one that meets the demand. The resulting workflow can consist of microservices, VNFs, and XaaS offerings.

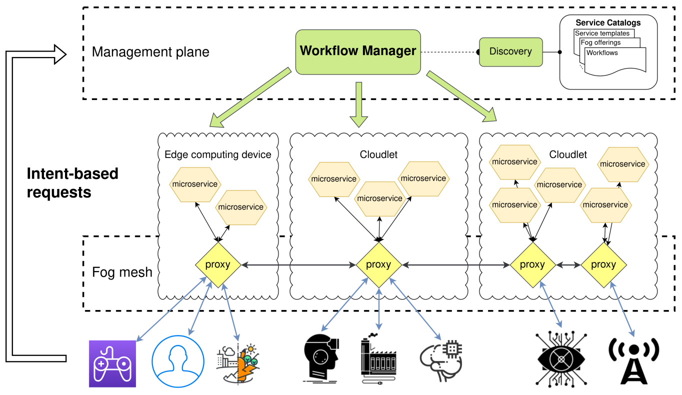

Fig. 2. A fog native architecture showing a fog mesh supporting both internal and external communication, and a workflow manager using a discovery service to design and deploy microservice workflows based on user requests.

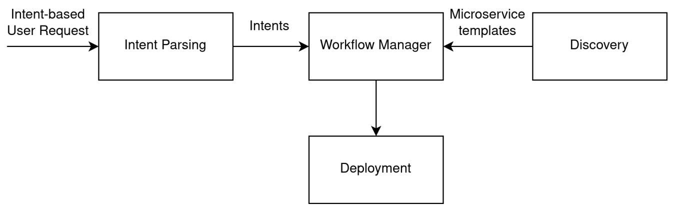

Fig. 3. Intent-based construction of a new workflow.

As shown in Figure 3, when constructing a new workflow, the workflow manager selects a number of microservices and offerings using a matching logic which takes into account the locality of the end-user and workflow constraints in order to ensure the required QoS. The resulting workflow takes the form of a desired state model that contains multiple connected components such as microservices and offerings. This model is delegated to lower-level orchestrators such as Kubernetes and/or deployed as VNFs. As a result, the practical implementation of requested functionality is different depending on the geographic location of the end-user, available infrastructure, and available microservices. Note that this work is focused on laying the foundations of intent-based workflow construction; leaving optimization solutions for future work.

C. Fog mesh providing inter-microservice connectivity

Internal connectivity in a cloud native environment is facilitated by a service mesh. As stated in Section II-B, the current generation of service meshes are not well adapted to the resource constraints of the fog. Addressing this challenge requires removing the one-to-one relationship between sidecar proxies and microservices, and adding regional awareness from a network standpoint. The resulting fog native service mesh, "fog mesh", automatically groups microservices into a number of regional clusters based on network constraints.

For example, microservices which are close to each other from a latency and network bandwidth perspective could share a single sidecar proxy. This vastly reduces the overhead required for the sidecar approach. Moreover, using this fog mesh, inter-microservice communication can use performant, not necessarily user-friendly, protocols. This mesh can be implemented as a flexible network of decentralized network functions.

D. *Fog mesh providing external connectivity*

The API gateway pattern is difficult to implement in the fog due to the heterogeneous network latency and possible spread of workflows over multiple regions, as explained in Section II-C. Therefore, this architecture automatically distributes the API gateway functionality by merging it into the fog mesh. Each fog mesh proxy acts as an ingest point for the microservices connected by the proxy, providing service-based handling of internal-external communication, automatically configured based on local needs. This effectively merges the concepts of "service mesh" and "API gateway" into a single "fog mesh", which provides this functionality in a distributed manner.

This has two advantages: firstly, it regionally distributes API gateway functionality automatically based on microservices and network constraints as explained in Section III-C. As a result, user requests and responses can be handled by the gateway in the region closest to the user, without the need for redirection to a central API gateway. Secondly, it removes the need for an additional service: fog mesh proxies handle both internal and external communication. This results in lower resource usage overhead.

Additionally, to fully utilize the potential for optimized communication, responses from microservices to end-users do not leave the fog mesh from the proxy closest to the last microservice, but from the proxy closest to the client. This fog mesh then translates the communication into a protocol tailored to the end user device, such as HTTPS in case of a browser. Notably, narrowing down the requests admitted by a proxy to those targeting the proxy's microservices combined with having a generally smaller number of users by virtue of locality is foreseen to incur a manageable state in the proxy.

E. *Fog mesh providing end-user aggregation*

In a cloud environment, the geographical distribution of an application is often manually designed by the application developer based on historic records of use and availability of cloud regions. This is infeasible in the fog, however, due to its high number of geographical compute locations as outlined in Section II-D. Thus, a fog native scheduler is needed which distributes a microservice workflow based on end-user demand. Automating this distribution, however, is non-trivial due to the higher dispersion of users, causing higher demand variation and thus increased pressure on the fog scheduler.

To address this challenge, this architecture includes the novel design pattern of aggregating end-users and their workflow requests by the fog mesh. This allows the scheduler to make decisions on aggregations of end-user requests instead of individual requests, lowering the demand for scheduling decisions. Since, as explained in Section III-D, fog mesh proxies act as the ingress point for local end-user requests, they have the required information to aggregate user requests for similar functionality into regional groups.

This, however, means that the end-user from a scheduling perspective is an aggregate and not the *actual* end-user. Since the fog mesh uses the scheduling information in order to manage communication, it will only be able to manage the connection up to the component acting as the aggregator. To ensure the aggregate component knows how to manage the connection to the end-user, this pattern introduces a *response-path token* uniquely identifying the end-user. This way, a workflow which, from a scheduling perspective, has a single end-user, can fan-out to an aggregate of nearby users.

IV. Example: decision-support in the fog

This section outlines a pathway towards system-level implementation of the architecture, through an example use case of UAV-based disaster management; enabling a dynamic decision-support system based on live drone feeds for aiding first responders. Autonomous drones observe the incident area, interpret the data and provide an action list prioritized on urgency or danger to responders as shown by Moeyersons et al. [12].

Due to the unexpected nature of most incidents, proactively designing and deploying decision support pipelines on location is not possible. Due to the high bandwidth and low latency requirements of the decision support pipeline, running them on a centralized data center will negatively affect both the end user experience and strain the network resources. Therefore, this requires a system that dynamically designs and schedules local workflows based on responders' needs (i.e. *intents*)

Figure 4 illustrates the example using the proposed fog-native architecture. A subset of fog mesh proxies may already be active at the responders' site while others have yet to be instantiated at the incident site. The proxy's implementation can build on experience gained from sidecar and API gateway technologies. The workflow manager and discovery service are placed in the network, possibly at the responders' site, having latency-bound communication with both end-user proxies. Notably, this example assumes a single instance of each management service controls the fog resources of both sites of interest; otherwise, distributed instances of each service might be needed.

In this example behavior, an emerging incident may trigger an intent-based workflow request by the responders' proxy (A), describing required tasks and data, including video feeds. The workflow manager interacts with the discovery service (B), to identify existing components and data. A subset of microservices may already be available, while others - such as the drone feed - are yet to be established. Upon arrival to the scene, the drone uses a fog proxy to advertises an intent-based description of its video stream to the discovery service (C). The latter informs workflow manager to complete the construction of the workflow. The deployment of the workflow is delegated to lower-level orchestrators. In this

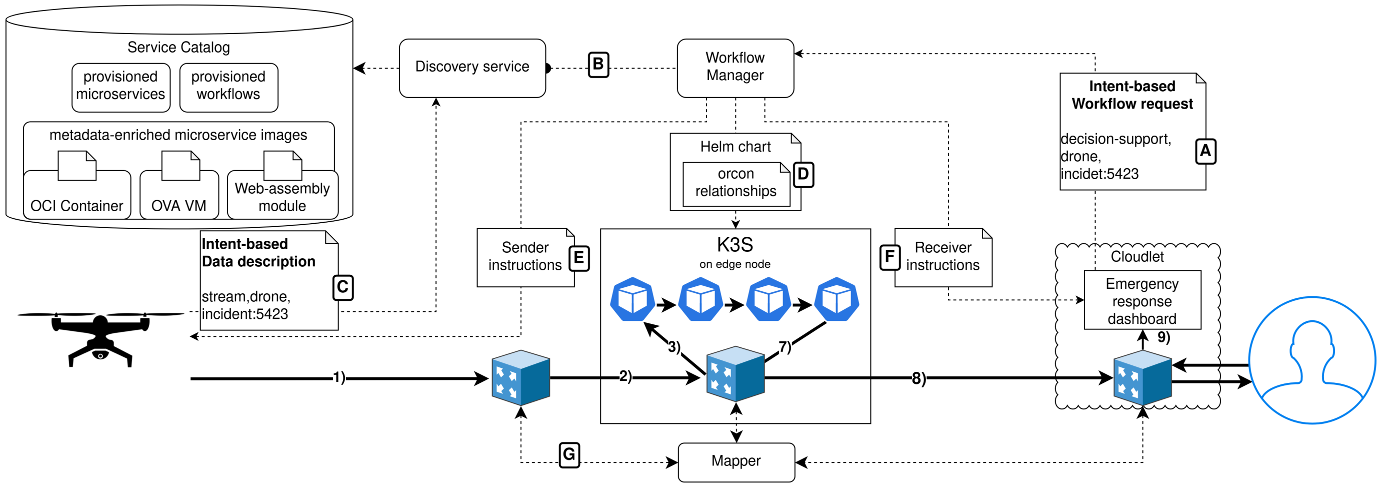

Fig. 4. Overview of the proposed fog native architecture enabling drone-based decision support for crisis response teams through matching of responders intents with offered microservices and data.

example, the workflow manager contacts a K3S edge node and uses a Kubernetes relationship orchestrator [13] to deploy an undistributed composition of microservices (D). After a successful deployment, the workflow manager instructs the drone to start transmitting the video stream (E) to be processed by the workflow, i.e. by the microservices following their dependency map in a hop-by-hop fashion. The last microservice delivers the decision support outcome to the responders' dashboard after it completes.

V. Evaluation

This section analyzes the impact of distributing workflows over a fog ecosystem using the discrete event simulation framework simmer [14]. The analysis illustrates the impact of internal network latency and application metrics on the perceived Quality of Service (QoS) of workflows distributed over the fog compared to *undistributed workflow allocation* in cloud systems. We present our results in terms of the *latency residual budget*, corresponding to QoS by measuring the difference between the latency threshold of a workflow, specified by the end-user, and the observed response time. A negative value indicates a violation of the latency agreement.

A fog-native network is modeled as a set of *fog nodes* overlaid on top of a softwarized routing network. Traffic per workflow is modeled as a set of simmer *trajectories* starting from the user to the fog node of the first microservice; and ending from the fog node of the last microservice back to the user. The CPU capacity of each node depends on the node's tier, with cloud nodes having the highest CPU capacity and edge nodes having the lowest. Similarly, network links are characterized by their bandwidth capacity (Mbps) and length (Km) with core links having the largest bandwidth and longest distance and edge links having the smallest and shortest counterparts. Propagation latency and the queuing counterpart at each routing node are calculated using link attributes, current state and data size. The processing latency of the deployed microservices are derived from the CPU capacity, current workload of a fog node, and task size. The total response time is then calculated as the additive accumulation of all latencies, between 'user-to-first-microservice' and 'last-microservice-to-user'.

For application workflows, the evaluation considers two forms of dependency maps: *Chain* and *Hub and Spoke* (H&S). In a *Chain* map, microservices are serially related to each other; whereas in a H&S map, the first and last microservices are *hubs* and intermediary ones are *spokes*. A workflow may either be distributed (i.e. microservices assigned) over multiple fog nodes, hence classified as *Distributed* or all corresponding microservices are assigned to one fog node and so deemed *Undistributed*. Moreover, each workflow is characterized by a *latency budget* indicating the maximum tolerable response time. Each microservice has a task size measured in number of CPU cycles, and input and output data measured in megabytes.

The simulation assumes 100 workflows offered in the network, each of which consists of 5 microservices selected from a catalog of 1000. Each workflow has either *Chain* or *H&S* dependency map. The CPU and data specification per microservice is defined per scenario. For the network, we consider the topology of the AT&T MPLS network of 25 nodes and 114 links [15]. It assumes a $3-$Tier fog network with: tier-0 central cloud (2 nodes), tier-1 the smaller *cloudLets* (4 nodes) and tier-2 the highly constrained edge (8 nodes). The fog nodes in each tier are placed randomly in the network. In all the results, the CPU and bandwidth capacities of any tier are approximately 10% equivalent of the upper tier. Finally, the simulation assumes each switching node to connect between 1000 and 4000 end-devices, generating requests for workflows at a rate of approximately 1500 requests per second.

A. Latency vs. distributability

This evaluation analyzes the interplay between workflow dependency and infrastructure distribution, and the impact on latency perceived by end-user.

Figure 5 shows the latency residual budget when varying the distribution of the fog infrastructure, extending from the typical central cloud to a hierarchically distributed fog. The results

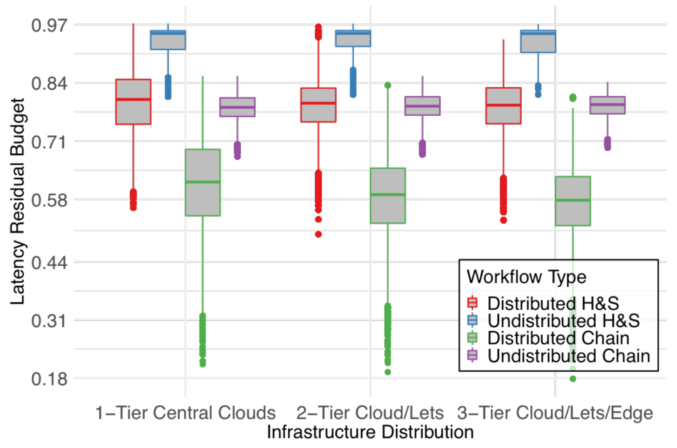

Fig. 5. The latency residual budget when varying the fog infrastructure from central cloud to hierarchical fog.

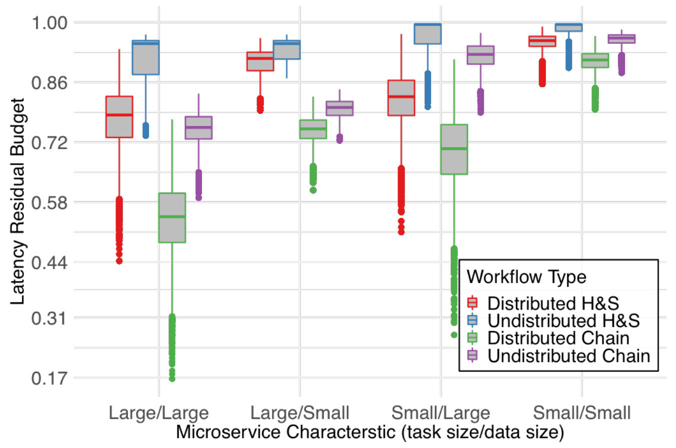

Fig. 6. The latency residual budget for variant task and data size in a 3-tier fog.

are shown for both undistributed workflows and workflows distributed randomly over multiple fog nodes. The observed latency residual budget for distributed workflows is lower than for undistributed variants, illustrating the impact of network latency on overall response time. Notably, the average residual budget of H&S workflows is approximately 20% higher than that of chain workflows, showing improvement as a result of parallelizing microservice execution. Moreover, the residual budget for distributed workflows decreases as the infrastructure changes from central to distributed. This increased response time is caused by

- additional communication latency from distribution of the workflow over a larger number of fog nodes, and
- increased computation latency from the lower CPU capacity of edge infrastructure.

Interestingly, undistributed workflows in a hierarchical fog perform no different from their counterparts in central clouds, showing the reduction in *communication* latency is countered by increased *computation* latency.

B. *Latency vs. application metrics*

Figure 6 shows the latency residual budget when having variant task and data size. The results show the response time for workflows of large sized data, (approximately 2-4 megabyte) is on average 10-25% higher than for small data (approximately 0.5-2 megabyte), irrespective of the task size. Nonetheless, workflows with large task size have, on average, a higher response time by approximately 5-20% compared with workflows of small task size, for the same data size. This reveals the significance of communication latency when having to transmit large volumes of data. The dependency map and distribution also impact the perceived response time. H&S workflows incur the quickest response time, even when comparing distributed ones to undistributed chains. This shows the effect of parallel microservice execution and the interplay with CPU and link bandwidth capacities. Although the last microservice in a H&S workflow waits for all the intermediaries to complete, the combined execution and waiting time remains smaller than that in undistributed chains. Although distributing a workflow reduces residual latency, the budget is not exceeded, which means it can be a valid option to reduce workload congestion by spreading compute load.

VI. Conclusion

Operational complexity, stringent resource constraints, varying internal network latency, and high granularity of geographic regions make the fog inherently incompatible with the cloud native paradigm. To address this challenge, this work proposes a fog native architecture along a set of design patterns to facilitate flexible and dynamic provisioning of microservice-based applications over the heterogeneous fog. Using intent-based workflow construction, applications are composed of loosely-dependent microservices selected to best match user requirements. A novel fog mesh enables microservice grouping under one proxy, seamless user-microservice and inter-microservice communications, and request aggregation. To illustrate a pathway towards implementation of the architecture, this article describes an example use case of drone-based decision support for first responders in the fog. Evaluation shows the impact of running microservice-based applications in a fog ecosystem, confirming, for example, network latency plays a bigger part in distributed workflow response time in the fog compared to the cloud.

Future work is foreseen to provide a prototype of the fog mesh and further investigate algorithms, optimizations and implementations for translating intents into desired state models. It will also further investigate management of data at rest in the fog.

References

[1] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Migrating to cloud-native architectures using microservices: An experience report,” in *Advances in Service-Oriented and Cloud Computing*, A. Celesti and P. Leitner, Eds. Springer International Publishing, 2016, pp. 201–215.

[2] T. Laszewski, K. Arora, E. Farr, and P. Zonooz, *Cloud Native Architectures: Design High-Availability and Cost-Effective Applications for the Cloud.* Packt Publishing, 2018.

[3] D. Gannon, R. Barga, and N. Sundaresan, “Cloud-native applications,” *IEEE Cloud Computing*, vol. 4, no. 5, pp. 16–21, 2017.

[4] A. El Malki and U. Zdun, "Guiding Architectural Decision Making on Service Mesh Based Microservice Architectures," in *Software Architecture*, ser. Lecture Notes in Computer Science, T. Bures, L. Duchien, and P. Inverardi, Eds. Cham: Springer International Publishing, 2019, pp. 3–19.

[5] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices Anti-patterns: A Taxonomy," in *Microservices: Science and Engineering*, A. Bucchiarone, N. Dragoni, S. Dustdar, P. Lago, M. Mazzara, V. Rivera, and A. Sadovykh, Eds. Cham: Springer International Publishing, 2020, pp. 111–128. [Online]. Available: https://doi.org/10.1007/978-3-030-31646-4_5

[6] A. Akbulut and H. G. Perros, "Performance Analysis of Microservice Design Patterns," *IEEE Internet Computing*, vol. 23, no. 6, pp. 19–27, Nov. 2019, conference Name: IEEE Internet Computing.

[7] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues," *Journal of Network and Computer Applications*, vol. 98, pp. 27–42, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804517302953

[8] C. Zhou, A. Fu, S. Yu, W. Yang, H. Wang, and Y. Zhang, "Privacy-Preserving Federated Learning in Fog Computing," *IEEE Internet of Things Journal*, vol. 7, no. 11, pp. 10 782–10 793, Nov. 2020, conference Name: IEEE Internet of Things Journal.

[9] I. Pelle, J. Czentye, J. Dóka, and B. Sonkoly, "Towards latency sensitive cloud native applications: A performance study on aws," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 272–280.

[10] OpenFog Architecture Workgroup, "OpenFog Reference Architecture for Fog Computing," Feb. 2017.

[11] M. Al-Naday and I. Macaluso, "Flexible semantic-based data networking for iot domains," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, 2021, pp. 1–6.

[12] J. Moeyersons, P.-J. Maenhaut, F. De Turck, and B. Volckaert, "Aiding First Incident Responders Using a Decision Support System Based on Live Drone Feeds," in *Knowledge and Systems Sciences*, ser. Communications in Computer and Information Science, J. Chen, Y. Yamada, M. Ryoke, and X. Tang, Eds. Singapore: Springer, 2018, pp. 87–100.

[13] M. Sebrechts, S. Borny, T. Wauters, B. Volckaert, and F. De Turck, "Service Relationship Orchestration: Lessons Learned From Running Large Scale Smart City Platforms on Kubernetes," *IEEE Access*, vol. 9, pp. 133 387–133 401, 2021, conference Name: IEEE Access.

[14] I. Ucar, J. A. Hernandez, P. Serrano, and A. Azcorra, "Design and analysis of 5g scenarios with simmer: An r package for fast des prototyping," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 145–151, 2018.

[15] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, Oct 2011.

Merlijn Sebrechts received the M.Sc. degree in information engineering technology from Ghent University, in July 2015, where he is currently pursuing the Ph.D. degree with imec, IDLab, Department of Information Technology (INTEC). He has worked on a number national and international research projects. His research interests include simplifying the management of complex applications in the cloud and beyond.

Bruno Volckaert (Member, IEEE) received the Ph.D. degree in resource management for grid computing from Ghent University, in 2006. He is currently a professor in advanced distributed systems at Ghent University and senior researcher at imec. He has worked on over 45 national and international research projects and is author or co-author of more than 150 peer-reviewed papers published in international journals and conference proceedings. His current research deals with reliable and high performance distributed software systems for a.o. Smart Cities, scalable cybersecurity detection and mitigation architectures and autonomous optimization of cloud-based applications.

Prof. Filip De Turck (Fellow, IEEE) leads the network and service management research group at Ghent University, Belgium and imec. He has coauthored over 700 peer reviewed papers. His research interests include design of secure and efficient softwarized network and cloud systems. He was elevated as an IEEE Fellow for outstanding technical contributions. He is involved in several research projects with industry and academia, served as chair of the IEEE Technical Committee on Network Operations and Management (CNOM), and steering committee member of the IFIP/IEEE IM, IEEE/IFIP NOMS, IEEE/IFIP CNSM and IEEE NetSoft conferences. He serves as Editor-in-Chief of IEEE Transactions on Network and Service Management (TNSM).

Prof. Kun Yang (Senior Member, IEEE) received his PhD from the Department of Electronic & Electrical Engineering of University College London (UCL), UK. He is currently a Chair Professor in the School of Computer Science & Electronic Engineering, University of Essex, UK, leading the Network Convergence Laboratory (NCL). His main research interests include wireless networks and communications, future Internet and edge computing. He manages research projects funded by UK EPSRC, EU FP7/H2020 and industries. He has published 300+ papers and filed 10 patents. He serves on the editorial boards of a number of IEEE journals (such as IEEE TNSE, WCL, ComMag). He is a Member of Academia Europaea (MAE), a Fellow of IET/BCS and a Senior Member of IEEE.

Mays Al-Naday (Member, IEEE) received her PhD degree from the University of Essex, United Kingdom, in 2015. She is currently an Assistant Professor in the School of Computer Science and Electronic Engineering, University of Essex. She has actively worked on a number of EU H2020 research projects in the area of future networking architectures. Her current research focuses on microservice networking, smart resource management, fog computing networks, networks for federated learning and security and Quality of Service in B5G/6G. She has been the organizer of prestigious workshops in Sigcomm 17-18 and IFIP 17.