Functionalities, Challenges and Enablers for a Generalized FaaS based Architecture as the Realizer of Cloud/Edge Continuum Interplay

George Kousiouris[1] and Dimosthenis Kyriazis[2]
[1]*Department of Informatics & Telematics, Harokopio University of Athens, 9, Omirou Str. 177 78, Tavros, Greece*
[2]*Department of Digital Systems, University of Piraeus, Karaoli & A. Dimitriou 80, 18534 Piraeus, Greece*

Keywords: Function as a Service, Cloud Computing, Cloud Architecture, Service Platforms, Middleware.

Abstract: The availability of decentralized edge computing locations as well as their combination with more centralized Cloud solutions enables the investigation of various trade-offs for application component placement in order to optimize application behaviour and resource usage. In this paper, the goal is to investigate key functionalities and operations needed by a middleware layer so that it can serve as a generalized architectural and computing framework in the implementation of a Cloud/Edge computing continuum. As a primary candidate, FaaS frameworks are taken under consideration, given their significant benefits such as flexibility in execution, maturity of the underlying tools, event driven nature and enablement of incorporation of arbitrary and legacy application components triggered by diverse actions and rules. Related work, gaps and enablers for three different layers (application design and implementation, semantically enriched runtime adaptation/configuration and deployment optimization) are highlighted. These aid in detecting necessary building blocks of a proposed generalized architecture in order to enclose the needed functionalities, covering aspects such as diverse service environments and links with the underlying platforms for orchestration, dynamic configuration, deployment and operation.

1 INTRODUCTION

The current Cloud computing landscape is characterized by an extreme diversity of offerings and services, incorporating multiple solutions. These include centralized Cloud providers (such as typical VM offerings, dedicated nodes, hardware enhanced resources such as GPUs and FPGAs etc), edge and fog environments, HPC facilities, mobile computing applications etc. implementing the Everything as a Service approach. On the other hand, applications are typically consisted of a multitude of components, others in need of locality and others in need of significant computational resources. These applications portray varying abilities to exploit the underlying services, varying requirements for operation, control and technologies/programming structures on which they rely (Ferrer et al, 2017).

For achieving a true smart cloud computing continuum, i.e. a unified approach on available resources, one should examine the domain from three main perspectives:

- **Perspective 1:** Continuum in terms of application design and definition, leading to a generalized adaptive approach for building the software or service stack. Adaptation to dynamic conditions (unexpected loads and failures), to different computing paradigms (microservices and functions) and functionalities of the new execution environment is paramount. Enabling the combination of edge/cloud resources can significantly enhance aspects such as latency (Bittencourt et al, 2018).
- **Perspective 2:** Continuum in terms of functionalities, deployment and management approaches. The scale and differentiation in the new cloud/edge interplay has become quite complex, thus managing dependencies and mechanisms across continuum resources has become a daunting task (Lynn et al, 2020). Once an application has been defined, can it be seamlessly managed by an underlying framework and distributed based on a given set of goals and constraints? Can the application exploit specific

Kousiouris, G. and Kyriazis, D.
Functionalities, Challenges and Enablers for a Generalized FaaS based Architecture as the Realizer of Cloud/Edge Continuum Interplay.
DOI: 10.5220/0010412101990206
In *Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER 2021)*, pages 199-206
ISBN: 978-989-758-510-4

features of the used service, such as GPU processing, multiple cores in the node etc.? If yes, which executable version of it and with which parameter set should be used during deployment, without user intervention? In order to achieve that, a relevant set of semantic descriptions should exist.

- **Perspective 3:** Continuum in terms of the relativity inserted between space (of deployment) and time (of execution) (Foster, 2019). Utilizing a centralized service far away from the observer may introduce latency or other factors that render this selection as more time consuming than a localized deployment. The analysis of this trade-off should result in a unified space-time combinatorial approach for service selection while considering space-time continuum distorting factors like multitenancy and resulting performance interference (Kousiouris et al, 2011). Therefore, dynamic incorporation of this factor's weight on the final optimization should be measured and taken under consideration.

In order to achieve these objectives, suitable middleware frameworks need to be designed, deployed and operated across the continuum. The aim of this paper is to investigate what is the status in some of the key areas mentioned above, in order to highlight existing capabilities as well as according gaps and enablers. Section 2 includes the suitability of the FaaS model for the specific purpose as well as investigation of the current status in application design and adaptation (Perspective 1), Section 3 investigates the usage of middleware and semantic technologies (Perspective 2), while Section 4 studies the space-time continuum capabilities (Perspective 3) Finally the paper proposes a high level architectural approach that would enhance the ability of FaaS frameworks to meet this diverse role in Section 5 and Section 6 concludes the paper.

2 FaaS CHARACTERISTICS AND APPLICATION DESIGN

2.1 Why FaaS? Key FaaS Characteristics of Interest

One of the main benefits of the FaaS model is the fact that it is built around the most sophisticated variation of the pay-as-you-go concept, the **pay-as-you-execute model**, thus only charging when the application code is actually executed (including billing factors such as function runtime and memory). What is more it alleviates from server environment maintenance. Break down into functions enables easier scalability and elasticity of the applications, thus better ability to exploit elastic resources and services, as well as software modularity and maintenance. Therefore it strengthens the benefits of a migration towards a cloud/edge service environment. This function-based break-down enables the easier distribution of tasks between centralized and edge resources available.

This is further strengthened by the fact that application structure is fed in the underlying FaaS platform (such as Openwhisk) which handles many of the deployment, configuration and orchestration needs. FaaS has been among the highest growth public cloud service types while the need to optimize cost savings from cloud services is the top priority for cloud users in 2020 (Flexera, 2020).

One of the main abilities of FaaS platforms is to include diverse components and behaviours and adapt them to event driven sequences and workflows, a feature known as **polyglot ability**. Its operators include notions such as Actions (application code that may include pure function code enforced on input message data, legacy non-FaaS components, arbitrary and diverse executables of any programming language in docker containers) and Rules (used to associate one event trigger with one or multiple actions, therefore enabling the definition of complex workflows). One key aspect is that the event sources can be anything i.e messages arriving on Message Queues, changes in databases, web interactions, service APIs etc. This enables bridging the FaaS model with other popular existing approaches such as microservices, REST services, legacy web applications or any arbitrary legacy component.

Functions can typically range from small and lightweight to larger and more computationally demanding, therefore very suitable for the cloud/edge interplay scope and offloading trade-offs investigation, depending on available hardware on the edge, latency considerations etc. However careful consideration should be given to aspects such as the distribution of the load, auto-scaling mechanisms, operational tasks and function limitations (Kuhlenkamp et al, 2020). A thorough analysis of numerous open and closed source FaaS frameworks (Van Eyk et al, 2019) indicates the fact that in many cases there are misconceptions around characteristics, while costs may also differ from the initial considerations.

Other benefits of a FaaS platform middleware include the abstraction of the underlying resources and infrastructure services used. Each platform may span across different providers, resource types etc in a manner that is agnostic to the end user. Locally in each

service resource used, the FaaS platform (through its distributed managers and agents) handles aspects such as function deployment, enabling seamless federation and distribution of the components through underlying container orchestrators.

The need to offload computation exploiting various trade-offs and capabilities as well as transform monolithic applications in order to be decomposed in smaller components, executed separately and on the most suitable resources, is proposed by the FuncX framework (Chard et al, 2020). Furthermore, approaches at an architectural level have started to emerge in order to scale and distribute FaaS platforms across different providers in multi-cloud or hybrid cloud scenarios (Vieira et al, 2020) as close as possible to the client.

Other approaches such as A3-E (Baresi et al, 2019) enable applications to execute parts of their logic on different infrastructures, with the goal of minimizing latency and battery consumption and maximizing availability. Cross-site orchestration has been investigated in the context of the AWS Lambda service in GlobalFlow (Zheng et al, 2019). Other frameworks based on e.g. WebAssembly (Hall et al, 2019) have emerged, in an effort to reduce container performance overheads in environments with need for low-latency response or hardware platforms with limited resources, however their tool support is not near the maturity of platforms such as Openwhisk.

Identified Gap. While FaaS frameworks portray a number of promising characteristics in terms of execution on demand, improved cost, ease of placement and inherent/direct parallelization achieved, along with a long list of open source tooling and approaches maturing, they come also with a number of shortcomings that should be addressed. Tooling availability related to **deployment and function reuse**, remains a major difficulty, in current FaaS systems (Leitner et al, 2019). Furthermore, abstractions and programming models for building non-trivial FaaS applications are limited. Currently frameworks imply the need for full porting of the application to the FaaS model, thus the redesign of their execution model around short-lived functions, leading to potential need for extensive application rebuilding. Another challenge is the handling of state in the FaaS model. The latter primarily targets at stateless functions that do minimal I/O and communication. Frameworks such as CloudBurst (Shreekanti et al, 2020) have tried to extend the scope to a broader range of applications and algorithms, while incorporating key-value stores for state sharing between functions. However new challenges emerge when **functions operate at a distributed and federated cloud-edge environment**, including data consistency, locality and performance.

2.2 Application Design and Adaptation to the FaaS Model

Visual environments have emerged in recent years as a user friendly and abstract mean of development that can speed up application development. Typically these environments are based on flow programming and offer ***palettes of readymade nodes or operators*** that incorporate the major functionalities needed. Function code is applied on the input message, transforming its contents based on the function logic and passing it to the next node in line. Furthermore, they encompass means of extension for these nodes as well as external repositories. Environments such as open-source Node-RED for IoT event driven applications and KNIME (mixture of open and proprietary models) for data science flows have emerged, indicating that the need for easier development is very relevant. Therefore they can be extended and adapted to eventually deploy the developed flows in a FaaS environment.

In terms of major open source FaaS platforms, these typically do not come with a UI for workflow definition (Van Eyk et al, 2019), with the exception of Apache Airflow that also includes the incorporation of operators to include typical cloud services or processes. One drawback of Airflow is that these operators are typically provider specific and thus ***cannot be reused,*** while amplifying the vendor lock-in. Also, they do not include advanced and abstracted cloud design patterns. Fission workflows are mainly programmatically defined. Proprietary solutions also exist with an extensive list of accompanying services such as the IBM Cloud (formerly Bluemix) environment (and Blueworks) as well as Google Composer.

In the cloud design patterns domain, Big Vendors have promoted Pattern-Based development through new programming and deployment paradigms in order to build value added services. This development methodology has the goal of providing complex services and resources by interaction of simpler ones and can be used to define proper orchestration actions (Amato et al, 2017). Typical cloud design patterns may include template structures and workflows such as AI training and optimization (Giampa et al, 2020), map/reduce types of structures, MPI based, data ingestion, preprocessing, encryption, privacy and transformation flows, load balancer structures, preconfigured messaging structures (e.g. publish/subscribe), data caching mechanisms, auto-scaling and throttling functionalities, continuous

deployment patterns etc. Some of these patterns may exist in implementation, even directly in the FaaS model or enabling transformations to this through suitable converter frameworks (Carvalho et al, 2019), however they are in need of parameterization and/or wrapping around the core design framework for achieving maximum abstraction.

Identified Gap. While helpful in the sense of each domain's usability, current design environments lack the ability to aid an application in exploiting cloud benefits through ready-made supporting structures that enhance functional and non-functional aspects. Furthermore, they lack a unified and vertical approach to enable application definition, enhancement with features and creation of cloud deployment specification in an integrated manner. Either deployment specification or workflow specification are supported but not combined and do not include design patterns and functionality automatically incorporated and configured in the application graph. Many of the proposed operators/environments are provider/platform specific and increase vendor lock-in.

Proposed Enablers for FaaS Application Design Linked to FaaS Frameworks. Cloud design and programming patterns offered as FaaS reusable components may significantly aid application adaptation or extension in order to embed this functionality alongside its current implementation. Furthermore, visual environments for workflow creation can significantly aid developers in their transition and application adaptation. Thus ***incorporation of such patterns in arbitrary flows*** and instantiation of them with the specific software artefacts/functions needed may be performed. Furthermore, a vertical approach in these tools is needed in the sense of the ability to produce directly the application deployment specification to a FaaS framework from the application design.

3 RESOURCE SEMANTIC ENRICHMENT AND USAGE IN RUNTIME DEPLOYMENT

Currently, a significantly high number of available cloud services exist in the market. This makes the task of **matchmaking** between user demands and service capabilities difficult. Approaches have been developed in order to cater for differences in semantics for the service composition process (Di Martino et al, 2017). However what is needed is a tighter link between semantics and platform **self-configuration processes** in order to fully exploit semantic descriptions usage during runtime. In some specialized cases, such transformations are performed for exploiting special purpose cloud services e.g. cloud-based FPGA's (Chen et al, 2019). In terms of fully integrated runtime usage and management through semantics, the AffectUs framework (Kousiouris et al, 2019) has used a combination of ontologies, semantic inference, REST services and flow based programming adapters in order to integrate the use of ontologies in a functional manner in the life-cycle of a supply chain management application.

Extensive cloud service description frameworks have been proposed in recent years (Ghazouani et al, 2020), covering a wide variety of common cloud service characteristics (type, deployment model, evaluation, functionality and operations, accessibility and authorization features, QoS capabilities, legal issues, pricing, resource control).

Identified Gap. Very interesting works exist in the field of cloud service descriptions, thus exploitable in the context of the Linked Data paradigm. What is yet to be accomplished is a fully integrated use of ontologies and semantics, not only for a preselection or interface composition process but also to enable functionally more automation of configuration.

Proposed Enablers for Usage of Semantics in Deployment Configuration and Automation. First of all, relevant ontologies need to be enriched or integrated in order to capture specific aspects of an application configuration, linked to the way this application is configured, deployed and managed. Afterwards, instantiated tuples (examples in Table 1) would enable inference on the way the application should be configured or whether it can exploit specific resource characteristics. This can also be extended at the function or cloud template pattern level and can be easily supported during runtime (e.g. multiple image versions with different characteristics for a given software artefact).

However, this needs to be coupled by a middleware layer that will bridge service/resource descriptions with software artefact descriptions, inferring whether capabilities of the former can be exploited through the needs/characteristics of the latter. Purposes for such inference can include usage ***during deployment and/or application adaptation***, in order to enhance both functionally and non functionally the automated and seamless application adaptation to ***diverse environments***. Moreover, a semantically enriched controlling logic can improve its agility. One could retrieve the specific resource type in which currently the application component is

running and infer whether changing a specific parameter would be expected to make a difference. Other aspects such as locality constraints, e.g. imposed by legal (Barnitzke et al, 2011), ethical or other requirements, can be expressed. Finally, through modelled information like endpoints and their relation to QoS features (monitoring or controlling parameter endpoints), automated incorporation of controller logic can be inserted in the designed workflows and parameterized on the fly.

Table 1: Example Ontology Relations Usable for Deployment Optimization.

Subject (example instance)	**Predicate**	**Object Class (example instance)**
cliOption (-threads)	setsParameter	Parameter (numberOfThreads)
Resource (aws.medium)	isA	ResourceType (VM)
ResourceType (VM)	hasParameter	Parameter (numberOfCores)
Parameter (numberOfCores)	isUsedBy	Parameter (numberOfThreads)
Endpoint (/setThreads)	managesParameter	Parameter (e.g. numberOfThreads)
Endpoint (/setThreads)	isA	HTTPEndpoint
Parameter (numberOfThreads)	Affects	QoSMetric (e.g. ResponseTime)
applicationComponent	hasSpecificLocalityConstraints	ProviderDC (if specific location and provider is needed)
applicationVersion (myImage)	isA	ContainerImage
ContainerImage(myImage)	isOptimizedFor	ResourceType (GPUEnabledVM)
CloudPattern (MessageMQTT)	isOptimizedFor	ResourceType (EdgeResource)
ResourceType (EdgeResourceA)	hasMeasuredPerformance	QoSMetric (Bench results)

One aspect that needs to be stressed is that this mechanism needs to be linked with respective annotation capabilities of typical orchestrator systems (e.g. Docker Compose, Kubernetes). Annotating the abilities of nodes (feasible in current orchestrators through e.g. node naming) will bridge the gap between selection and enforcement of a given deployment scheme.

4 SPACE-TIME CONTINUUM EVALUATION/ OPTIMIZATION

Even though the FaaS model promises reductions of cost compared to IaaS and PaaS offerings, its billing mechanisms typically include function invocation numbers as well as execution time. However, in this context ***costs are less predictable***, especially because they are tied to function performance (as well as the provider's environment). Reports (Bortolini et al, 2019) have observed significant differences (up to 8.5× in performance and 67 × in cost between providers, 16.8× in performance and 67.2× in cost between programming languages). The problem of the assessment of black box Cloud services performance is especially intense in FaaS environments (Pellegrini et al, 2019). Cloud Service Providers usually restrict the maximum size of code, memory and runtime of Cloud Functions. The aforementioned work introduces a baseline FaaS benchmarking tool, which allows users to evaluate the performance of Cloud Functions. Challenges as well as requirements for a future FaaS measurement framework include taking into account notions of cost, realistic workloads, more (open-source) platforms, and cloud integration.

Baseline approaches such as FunctionBench (Kim et al, 2019), including workloads such as ML, network and micro-level benchmarks are a step forward. Comparisons have been performed to better understand the cost vs performance trade-off between FaaS and IaaS such that cloud users can decide which approach is suitable for them. Reports (Malla et al, 2020) on comparisons between Google cloud's FaaS (Cloud Functions) and its IaaS (Compute Engine) in terms of cost and performance have indicated that FaaS can be 14% to 40% less expensive than IaaS for the same level of performance. However, performance of FaaS exhibits higher variation.

In terms of deployment optimization and placement, numerous approaches are available (Cao et al, 2019), even from the initial Cloud era, focusing on multi-cloud service placement (Ferrer et al, 2012) or more recently for cloud-to-edge specific issues (Meixner et al, 2019). In many cases the size of the problem due to its NP hard nature is impossible to calculate optimally without breaking another constraint (e.g. time to reach the deployment decision), given the range of possibilities to take under consideration.

Identified Gap. There needs to be a closer link and tight collaboration between multiple functionalities and primarily resource capacity measurement/evaluation (through e.g. benchmarks that take under consideration other parameters such as function based execution, time to completion, memory used etc.) and deployment optimization.

Proposed Enablers for Enhanced Deployment Optimization Across the Continuum. Through the

inclusion of periodic benchmarking/ evaluation details for each resource type (integrated also with the semantic descriptions of Section III), as well as relevant factors included in the optimization model, the space-time continuum approach (Foster, 2019) may be implemented. Furthermore, the integration of the semantic querying as an initial filter of resources (and based on the discussion in Section 3), can significantly reduce the search space of an optimization algorithm and therefore potentially enable the application of globally optimal algorithms such as branch-and-bound. The overall process of the previous sections can be summarized in the sequence presented in Figure 1.

5 GENERALIZED ARCHITECTURAL BLOCKS

The functionalities presented in the previous sections are included in the form of generalized architectural building blocks in Figure 2, responsible for implementing the sequence of Figure 1. This architecture does not assume that there is any special connection or federation between the entities (e.g. Cloud or Edge providers), only a middle managing entity like a broker/platform manager that creates and operates resources on the cloud/edge.

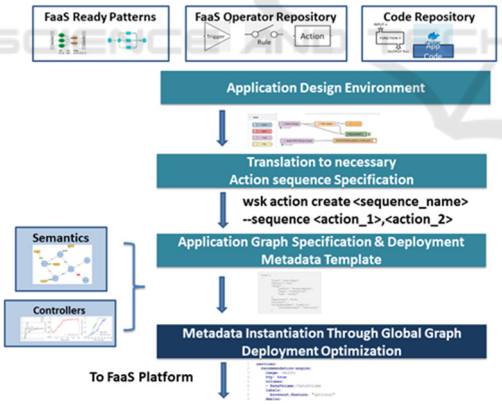

Figure 1: From application design to deployable FaaS.

The entry point is the Application Design Layer, that includes aspects such as management of the application code basis, visual flow programming environments enriched with ready-made software pattern flows, FaaS operators, followed by suitable parameterization and code insertion where needed. A necessary step in this case is also the semantic description of the application parts, whether these are existing legacy components and executables or newly created functions. This step aids in the creation of semantic triples in an according knowledge base, following the respective Ontological definitions. The same step of semantic description population needs to be undertaken also at the resource side, so that the inference engine can exploit and combine the two sources of semantics.

Next, the main management layer (Continuum Deployment Layer) is responsible for receiving the application graphs extracted from the design environments and convert them to the action sequence specification needed by the main FaaS platform (e.g. Openwhisk). The optimization process is only performed after a relevant query towards the Inference Engine, that can apply functional or non functional constraints and return the subset of resources that address the requirements. These should then be mapped on the application graph and evaluated in terms of performance, cost or other goal for which information is available (e.g. energy efficiency). Benchmarking or historical monitoring information for resources can be acquired from the Performance Measurement block, that aims to capture this either through monitoring of executions or from periodic performance evaluation tests.

From this optimization process (applied in the Global Continuum Placement Optimizer), the final resulting deployment graph may be acquired and forwarded for deployment to the mainstream FaaS platform, properly annotated via the mechanisms described in Section 3 in order to dictate the deployment scheme. However this mainstream FaaS platform needs to be managed by the Global FaaS layer, in the sense that the latter will have the responsibility to create according resource instances (e.g. Openwhisk nodes) in each of the Cloud/Edge federation resources. It is necessary to stress that this resource creation **does not assume any special agreement or link between the different entities**. The Continuum Deployment layer can act like any typical customer of the latter. A final step is the incorporation of supporting structures to functionally link these distributed resources (virtual networking layers and/or in-memory data services for data locality and state preservation for functions).

At the provider-local level, once the application is deployed, the elasticity controllers (embedded in the application graph) can check the detailed function execution logs or application endpoints (obtained from the semantic descriptions) for performance information and upon detection of an under or over performing application part they can toggle local resources given by the local CSP (e.g. through API calls). If despite these efforts this application part does not suitably adapt to the desired QoS levels,

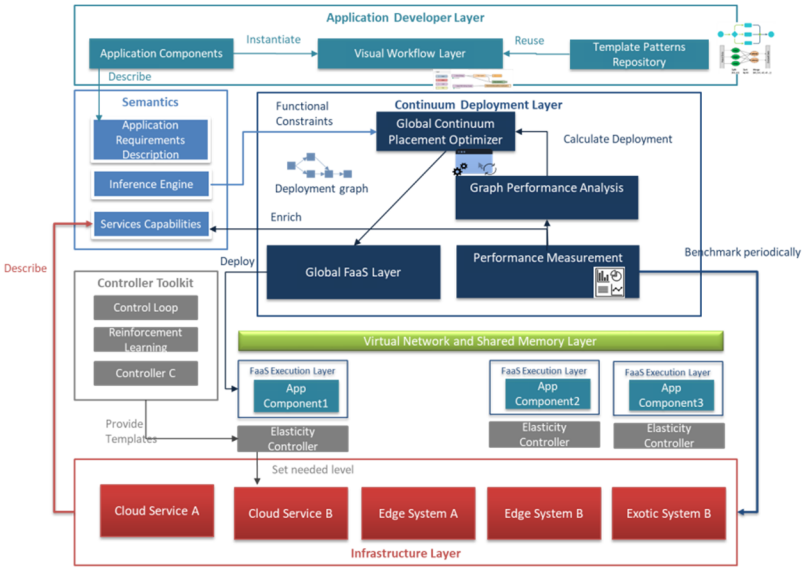

Figure 2: Generalized Architectural Building Blocks.

then the problem can be redirected to the global optimization plane for a new deployment optimization.

6 CONCLUSIONS

As a conclusion, FaaS presents a set of characteristics (e.g. granular execution, flexible deployment structures, supporting platform tools) that enable its usage across diverse devices as well as distributed environments. However in order for this upcoming computing model to reach its full potential in the Cloud/Edge interplay scenario, a number of additions need to be performed to enable easier application creation, adaptation, and seamless management across diverse locations.

The work in this paper proposed additions from application design to deployment and operation. Initially the application design and implementation process, supported by implementation templates that cover typical cloud oriented functionalities directly in the FaaS model, as well as visual flow programming environments, will aid in abstracting FaaS migration processes or application functionality extensions. To this end, the polyglot ability of FaaS frameworks is a major strength.

Following, extensions in terms of semantic descriptions and their incorporation during the runtime selection and configuration is another key enabler that aids in automatically adapting to the diverse environments and optimizing application setup. Optimization of deployment selection can significantly benefit from applying such semantic descriptions as well as runtime evaluation of Cloud/Edge resources. The overall analysis has led to a generalized architectural approach that can aid in addressing functional and non-functional requirements of complex applications deployed over dynamic and volatile environments in an abstract manner. The proposed architecture does not imply any specific relation or exposure between providers and is regulated by a brokering entity that has the goal of managing the middleware layer.

ACKNOWLEDGEMENTS

The research leading to the results presented in this paper has received funding from the European Union's Project H2020 PHYSICS (GA 101017047).

REFERENCES

Amato, F. and Moscato, F., 2017. Exploiting cloud and workflow patterns for the analysis of composite cloud services. Elsevier FGCS, 67, pp.255-265

Baresi, L., Mendonça, D.F., Garriga, M., Guinea, S. and Quattrocchi, G., 2019. A unified model for the mobile-edge-cloud continuum. ACM Transactions on Internet Technology (TOIT), 19(2), pp.1-21

Barnitzke, B., Ziegler, W., Vafiadis, G., Nair, S., Kousiouris, G., Corrales, M., Wäldrich, O., Forgó, N. and Varvarigou, T., 2011. Legal restraints and security requirements on personal data and their technical

implementation in clouds. In Workshop for E-contracting for Clouds, eChallenges (pp. 51-55).

Bittencourt, L., Immich, R., Sakellariou, R., Fonseca, N., Madeira, E., Curado, M., Villas, L., DaSilva, L., Lee, C. and Rana, O., 2018. The internet of things, fog and cloud continuum: Integration and challenges. Internet of Things, 3, pp.134-155.

Bortolini, D. and Obelheiro, R.R., 2019, November. Investigating Performance and Cost in Function-as-a-Service Platforms. In International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (pp. 174-185). Springer, Cham.

Cao, B., Zhang, L., Li, Y., Feng, D. and Cao, W., 2019. Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework. IEEE Communications Magazine, 57(3), pp.56-62.

Carvalho, L. and de Araújo, A.P.F., 2019. Framework Node2FaaS: Automatic NodeJS Application Converter for Function as a Service. In Proceedings of the 9th International Conference on Cloud Computing and Services Science (Vol. 1, pp. 271-278).

Chard, R., Babuji, Y., Li, Z., Skluzacek, T., Woodard, A., Blaiszik, B., Foster, I. and Chard, K., 2020. funcX: A Federated Function Serving Fabric for Science. arXiv preprint arXiv:2005.04215.

Chen, Y., He, J., Zhang, X., Hao, C. and Chen, D., 2019, February. Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs. In Proceedings of the 2019 ACM/SIGDA International Symposium on FPGA (pp. 73-82).

Di Martino, B., Cretella, G. and Esposito, A., 2017. Cloud services composition through cloud patterns: a semantic-based approach. Soft Computing, 21(16), pp.4557-4570

Ferrer A.J., Woitsch R., Kritikos K., Kousiouris G., Aisopos F., Garcia D., Plebani P., and Masip X. 2017. Future Cloud Research Roadmap, FP9 Cluster Inputs. Technical Report. Retrieved from https://drive.google.com/file/d/0B4hHTKjZDMXGSGxoYnh4eXhURzA/view.

Ferrer, A.J., HernáNdez, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R.M., Djemame, K. et al., 2012. OPTIMIS: A holistic approach to cloud service provisioning. Future Generation Computer Systems, 28(1), pp.66-77.

Flexera State of the Cloud Report, 2020, available at: https://info.flexera.com/SLO-CM-REPORT-State-of-the-Cloud-2020

Foster I., "Coding the Continuum," 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rio de Janeiro, Brazil, 2019, pp. 1-1, doi: 10.1109/IPDPS.2019.00011.

Ghazouani S., H. Mezni, and Y. Slimani, “Bringing semantics to multicloud service compositions,” Softw: Pract Exper, vol. 50, no. 4, pp. 447–469, Apr. 2020.

Giampa, P. and Dibitonto, M., 2020. MIP An AI Distributed Architectural Model to Introduce Cognitive computing capabilities in Cyber Physical Systems (CPS). arXiv preprint arXiv:2003.13174

Hall, A. and Ramachandran, U., 2019, April. An execution model for serverless functions at the edge. In Proceedings of the International Conference on Internet of Things Design and Implementation (pp. 225-236).

Lynn, T., Mooney, J.G., Domaschka, J. and Ellis, K.A., 2020. Managing distributed cloud applications and infrastructure: A self-optimising approach.

Kim, J. and Lee, K., 2019, July. Functionbench: A suite of workloads for serverless cloud function service. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD) (pp. 502-504). IEEE.

Kousiouris, G., Cucinotta, T. and Varvarigou, T., 2011. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. Journal of Systems and Software, 84(8), pp.1270-1291

Kousiouris, G., Tsarsitalidis, S., Psomakelis, E., Koloniaris, S., Bardaki, C., Tserpes, K., Nikolaidou, M. and Anagnostopoulos, D., 2019. A microservice-based framework for integrating IoT management platforms, semantic and AI services for supply chain management. ICT Express, 5(2), pp.141-145.

Kuhlenkamp, J., Werner, S. and Tai, S., 2020, April. The Ifs and Buts of Less is More: A Serverless Computing Reality Check. In 2020 IEEE International Conference on Cloud Engineering (IC2E) (pp. 154-161). IEEE.

Leitner P., Wittern E., Spillner J., and Hummer W ,2019. A mixed-method empirical study of Function-as-a-Service software development in industrial practice. Journal of Systems and Software, 146:340–359

Malla, S. and Christensen, K., 2020. HPC in the cloud: Performance comparison of function as a service (FaaS) vs infrastructure as a service (IaaS). Internet Technology Letters, 3(1), p.e137

Meixner S., D. Schall, F. Li, V. Karagiannis, S. Schulte and K. Plakidas, "Automatic Application Placement and Adaptation in Cloud-Edge Environments," 2019/ 24th IEEE ETFA, Spain, 2019, pp. 1001-1008.

Pellegrini, R., Ivkic, I. and Tauber, M., 2019. Function-as-a-Service Benchmarking Framework. arXiv preprint arXiv:1905.11707.

Sreekanti, V., Lin, C.W.X.C., Faleiro, J.M., Gonzalez, J.E., Hellerstein, J.M. and Tumanov, A., 2020. Cloudburst: Stateful Functions-as-a-Service. arXiv preprint arXiv:2001.04592.

Van Eyk, E., Grohmann, J., Eismann, S., Bauer, A., Versluis, L., Toader, L., Schmitt, N., Herbst, N., Abad, C. and Iosup, A., 2019. The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms. IEEE Internet Computing.

Vieira S., L., Vasconcelos, A., Batista, Í., Silva, R.A. and Brasileiro, F., 2019, September. DisOpenFaaS: A Distributed Function-as-a-Service Platform. In Anais Estendidos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (pp. 33-40).

Zheng, G. and Peng, Y., 2019, July. GlobalFlow: A Cross-Region Orchestration Service for Serverless Computing Services. In 2019 12th IEEE CLOUD (pp. 508-510). IEEE