
TOWARDS A DISTRIBUTED STORAGE FRAMEWORK FOR EDGE COMPUTING INFRASTRUCTURES

Antonios Makris¹, Evangelos Psomakelis¹, Theodoros Theodoropoulos¹, and Konstantinos Tserpes¹

¹Department of Informatics and Telematics, Harokopio University of Athens, Greece

ABSTRACT

Due to the continuous development of Internet of Things (IoT), the volume of the data these devices generate are expected to grow dramatically in the future. As a result, managing and processing such massive data amounts at the edge becomes a vital issue. Edge computing moves data and computation closer to the client enabling latency- and bandwidth-sensitive applications, that would not be feasible using cloud and remote processing alone. Nevertheless, implementing an efficient edge-enabled storage system is challenging due to the distributed and heterogeneous nature of the edge and its limited resource capabilities. To this end, we propose a lightweight hybrid distributed edge/cloud storage framework which aims to improve the Quality of Experience (QoE) of the end-users by migrating data “close” to them, thus reducing data transfers delays and network utilization. The proposed edge storage component (*ESC*) exploits the Dynamic Lifecycle Framework, in order to enable transparent and automated access for containerized applications to remote workloads. The effectiveness of the ESC is evaluated by employing a number of resource utilization and Quality of Service (QoS) metrics.

Keywords edge computing, edge storage, container-based virtualization, cloud computing, internet of things, kubernetes, minio

1 Introduction

Edge computing has attracted a lots of attention both from industry and academia in recent years [1, 2, 3, 4, 5, 6, 7] and considered as a key enabler for addressing the increasingly strict requirements of next-generation applications [8]. Contrary to (traditionally centralized) cloud computing, in edge computing, the computational resources are placed closer to the end-users into the so-called edge [9]. Amongst many others, this has the benefit of reducing the latency times. Moreover, by placing resource-rich nodes in close proximity to mobile or IoT devices, edge computing offers more responsive services, along with higher scalability and availability than traditional cloud platforms. In addition, edge computing significantly reduces the amount of data in transit towards remote clouds and enable data processing near the data-sources. Ultimately, expanding the possibilities for more delay-sensitive and high-bandwidth applications that would not be feasible using cloud and far remote processing alone [10]. To accelerate adoption and reap the benefits of edge computing, technologies from various domains need to be exploited to eventually realize the cloud/edge integration. These include computing, network and application-oriented technology fragments that need to be ingeniously put together to form and manage a network and computing continuum [11].

One of the key challenges in the development of applications at the edge, is the efficient data sharing between the multiple edge clients. Data sharing can be realized within individual application frameworks or through an external storage service. Edge storage can greatly improve data access which in turn enables latency-sensitive applications [12]. However, the distributed, dynamic and heterogeneous environment in the edge along with the diverse application’s requirements poses several challenges related to i) the coordination of unreliable devices and network, ii) hardware and software incompatibilities that arise due to the plethora of different devices, iii) mobility of the devices and the users, iv) integration of different data storage formats and data types, v) limited resources of the edge devices, vi) security and privacy concerns and vii) QoE & QoS insurance.

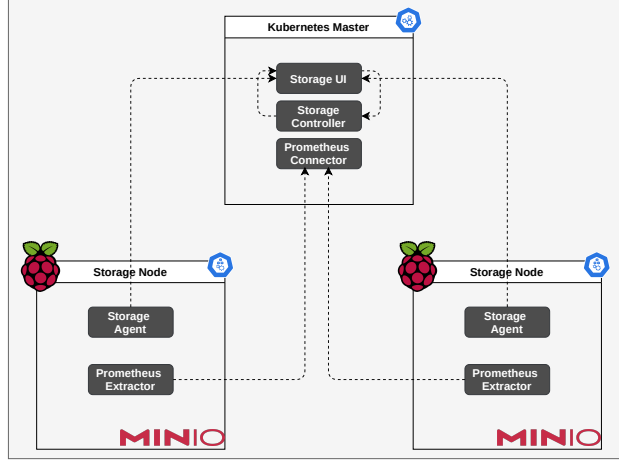


Figure 1: Edge Storage architecture

Despite the recent advancements in providing an edge storage solution, there are still issues left to be dealt with. Some issues related to the non-functional requirements of cloud-based applications. OpenStack Swift, Cinder, CEPH and MinIO are the most prominent technical solutions. However, it takes a more elaborate solution so as to deal with the inherent unreliability of the edge devices. To tackle these issues, one must leverage the core infrastructure and extend or integrate the aforementioned tools with cloud-based storage services. Our research work revolves around the decisions that must be taken into consideration, in order to design an edge storage system. These decisions are about the storage type, the storage system and the system architecture i.e. both the physical and the software architecture.

This research work aims in providing a hybrid distributed edge / cloud storage framework spread across heterogeneous edge and cloud nodes with considerations on performance (QoE), emphasizing on the resolution of the problem of data distribution and offloading based on application's requirements. The general requirements can be summarized as follows: i) low data retrieval latency and high bandwidth, ii) high availability by employing fault tolerance and migration methods, iii) high integrity, iv) dealing with the possible lack of storage resources at an edge node, v) support rapid application component deployment, vi) automatically restart/replace unresponsive components, vii) high heterogeneity and viii) edge cloud server pool isolation. These requirements can be achieved by optimizing resource usage, allocation and data management plans on edge devices per minicloud. A minicloud is a set of resources that are available at the edge and can host instances of application components. In detail, the proposed edge storage component provides a reliable, fast, stable and secure shared storage engine, accessible by all devices and users of a minicloud. This engine needs to be extremely lightweight since it is created for edge devices with extremely limited resources, like Raspberry Pis or other micro-computer devices.

The evaluation of ESC is conducted by employing a number of resource utilization and QoS (performance) metrics into two different deployments, one in a local and one in a remote minicloud. In detail, it can be observed that the performance metrics for the local minicloud are of a totally different scale compared to the ones in the remote cluster. The local minicloud deployment can perform data tasks between 3 and 17 ms while the remote host is performing the same tasks in 84 to 450 ms. This great difference is due to the intricate systems and protocols that come in-between the sender and receiver of each data packet, if the sender is on a remote minicloud. On the other hand, in a local minicloud deployment the sender can communicate directly with the receiver, bypassing all these overheads. Moreover, the resource utilization data prove that ESC is extremely lightweight, using only a small portion of the CPU of the host machine and minimal RAM.

The rest of this paper is structured as follows. Section 2 serves as a literature review in the field of storage solutions for edge computing infrastructures. Section 3 presents the system architecture and the edge storage component while Section 4 presents the experimental evaluation of the proposed approach. Finally, Section 5 concludes the merits of this work.

2 Related Work

With the explosion of edge computing, massive amounts of data are generated by IoT devices at the edge. To alleviate the pressure on the network and to achieve low-latency and high-throughput data processing, edge computing emerged

as a prominent paradigm as it makes computing and storage resources available at the edge of the network. In this context, storing and effectively managing the massive amounts of generated data at the edge is of utmost importance [6]. Psaras et al. [13] conducted an initial investigation of the benefits and challenges of data storage at the edge, in order to alleviate the stress that the massive amounts of data generated by IoT devices put on the core network for the transmission of these data to cloud storage. Confais et al. [14] suggest that each edge storage system should employ the following properties: low access time, network containment between miniclouds, availability through partitioning and mobility. Traditional file systems such as the distributed HDFS or Lustre seems to be inappropriate solutions in edge environments due to their centralized nature. On the other hand, object storage software systems which leverage Peer-To-Peer (P2P) mechanisms, can cope better with the proposed edge storage requirements. Clarke et al. [15] propose an epidemic approach for decentralized storage systems which offer data publication, replication and retrieval. However, the proposed approach presents limitations regarding resource discovery. Additionally, Sonbol et al. [16] proposed a decentralized storage system called EdgeKV. It consists of a local layer with several groups formed by geographically “close” nodes and a global layer where different groups are connected through gateway nodes. These layers are connected through a ring overlay which offers a fast and reliable system, utilizes data replication and provides strong consistency. Nicolaescu et al. [17] proposed an in-network storage and management framework at the network edge called SEND, which aims to improve the performance of the network edge and QoS. Raw and processed data are stored in persistent storage close to users for periods of time longer than cache storage. Through a logically centralized management plane, SEND utilizes data attributes generated at the edge to make intelligent decisions about the placement of functions and data across different edge locations, in order to maximize the offered QoS.

Research for the efficient data placement takes a prominent role in developing a reliable edge storage solution when heterogeneous storage systems in edge and cloud nodes need to exchange data. In addition, regarding resource management, several challenges concerning the adaptation to the dynamic environments and the large-scale optimization for the collaboration of multiple edge servers must be addressed. Near real-time decisions can be efficiently improved by moving the analytics “close” to the data. As a result, edge architectures can reduce the amount of data traversing the network, thus minimizing latency and overall costs. Lujic et al. [18] propose a three-layer architecture model for data storage management on the edge. The edge storage component is responsible for storing and sending data to an adaptive algorithm which dynamically finds the trade-off between the quality and the amount of data stored, receiving data for the cloud layer and storing the results from the adaptive algorithm. As edge nodes do not have the same capacity as specialized storage devices, it is critical to determine which parts of data will be stored to improve real-time performance, while other less used data may be uploaded to the clouds. In order to address the problem of limited storage space in edge computing and to reduce data loss caused by unstable networks, Xing et al. [19] proposed a distributed multi-level storage system model which is based on a multiple-factors Least Frequently Used (mLFU) replacement algorithm. While direct algorithms like LFU are effective on the edge with restricted computational capabilities, they only consider access frequency of data. On the other hand, mLFU considers also the importance of data which is defined as the correlation degree between the file and the program.

3 System Architecture

The proposed module employs the Kubernetes¹ ecosystem which is an open-source system for automating deployment, scaling, and management of containerized applications. More specifically, a lightweight Kubernetes distribution built for IoT & edge computing is used, called K3s². K3s is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT appliances. The smallest deployable unit of execution in Kubernetes is a pod, which consists of one or more containers. Pods access storage back-ends through volumes provisioned by a storage provider such as the local filesystem on the worker node, Networked File System (NFS) and so on. However, since pods are assigned on demand, the actual access occurs through a resource object called Persistent Volume Claim (PVC). A PVC is bound to a Persistent Volume (PV) object that is provisioned by the chosen storage provider. This interaction is standardised through the Container Storage Interface (CSI)³.

As a storage solution, an open-source framework created by IBM is utilized, called MinIO⁴. MinIO is an inherently decentralized, P2P solution which is designed to be cloud native and can run as lightweight containers managed by external orchestration services such as Kubernetes. It supports a hierarchical structure in order to form federations of clusters and it has been proven as a valid candidate for an edge data storage system [20]. MinIO writes data and metadata together as objects, eliminating the need for a metadata database. In addition MinIO performs all functions

¹<https://kubernetes.io/>

²<https://k3s.io/>

³<https://kubernetes-csi.github.io/docs/>

⁴<https://min.io/>

(erasure code, bitrot check, encryption) as inline, strictly consistent operations. The result is that MinIO is exceptionally resilient. In addition, it uses object storage over block storage so it is in fact a combination of the two systems, preserving the lightweight distributed nature of block storage while providing the plethora of metadata and easy usage of the object storage. Unlike other object storage solutions that are built for archival use cases only, the MinIO platform is designed to deliver the high-performance object storage that is required by modern big data applications. In addition, Prometheus⁵ is responsible for the collection of monitoring data about the real time performance of the nodes and the component as a whole, in order to analyse the behaviour of different applications, and optimize the cluster architecture, the options, and the data distribution. Prometheus is an open-source systems monitoring and alerting toolkit originally built by SoundCloud.

A high-level architecture of the storage component with the interconnections between the sub-modules is illustrated in Figure 1, which consist of an edge server (Kubernetes master) and two Raspberry Pis acting as storage workers. Kubernetes master acts as the storage controller, storage UI access point and Prometheus master for the specific cluster while each node connected to the Kubernetes cluster has the potential of becoming a storage worker for this cluster. As a node we define a Kubernetes node, which can be a PC, a laptop, an IoT device or any other compatible device. In addition, MinIO provides both a web-based GUI and an AWS S3 compatible API library. Finally, Prometheus comes bundled with Grafana⁶, which provides an excellent web-based GUI for displaying the highly customized datasets that Prometheus gathers when monitoring the edge storage component and the machines involved in it.

3.1 Dynamic Lifecycle Framework

Hybrid edge/cloud environment is rapidly becoming the new trend for organizations seeking the perfect mix of scalability, performance and security. As a result, it is now common for an organization to rely on a mix of on-premises datacenters (private cloud), and cloud/edge solutions from different providers to store and manage their data. Nevertheless, many obstacles arise when applications have to access the data. On the one hand, developers need to know the exact location of the data and on the other hand to manage the correct credentials to access the specified data-sources holding their data. In addition, access to cloud/edge storage is often completely transparent from the cloud management standpoint and it is difficult for infrastructure administrators to monitor which containers are accessing which cloud storage solution. Even if containerized components and microservices are largely promoted as the appropriate solution to efficiently deploy and manage storage on top of a hybrid edge/cloud infrastructure, containerization makes it more difficult for the workloads to access the shared file systems. Currently, there are no established resource types to represent the concept of data-source on Kubernetes. As more and more applications are running on Kubernetes for batch processing, end users are burdened with configuring and optimising the data access [21].

To tackle the aforementioned issues, Dataset Lifecycle Framework (DLF)⁷ is employed which is an open-source project that enables transparent and automated access for containerized applications to data-sources. DLF enables users to access remote data-sources via a mount-point within their containerized workloads and it is aimed to improve usability, security and performance, providing a higher level of abstraction for dynamic provisioning of storage for the users' applications. With the integration of DLF on Kubernetes pipelines, it is able to mount object stores as PVCs and present them to pipelines as a POSIX-like file system. In addition, DLF makes use of Kubernetes access control and secret so that pipelines do not need to be run with escalated privilege or to handle secret keys, thus making the platform more secure. In more technical detail, DLF orchestrates the provisioning of PVCs required for each data-source which users can reference in their pods, allowing them to focus on the actual workload development rather than configuring/mounting/tuning the data access. DLF is designed to be cloud-agnostic and due to CSI, it is highly extensible to support various data sources. CSI is a standard for exposing arbitrary block and file storage storage systems to containerized workloads on Container Orchestration Systems (COS) like Kubernetes. With the adoption of COS, the Kubernetes volume layer becomes truly extensible. On the infrastructure side, DLF also enables cluster administrators to easily monitor, control, and audit data access.

DLF introduces the Dataset as a Custom Resource Definition (CRD)⁸, which is a pointer to existing S3 or NFS data-sources. A Dataset object is a reference to a storage provided by a cloud-based storage solution, potentially populated with pre-existing data. In other words, each Dataset is a pointer to an existing remote data source and is materialized as a PVC. The Dataset is a declarative construct that abstracts access information and provides a single reference for data in Kubernetes. Users only need to include this reference in their deployments to make the data available in pods, either through the file system or through environment variables [22]. DLF is completely agnostic to where/how a

⁵<https://prometheus.io/>

⁶<https://grafana.com/>

⁷<https://datashim.io/>

⁸<https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/>

specific Dataset is stored, as long as the endpoint is accessible by the nodes within the Kubernetes cluster, in which the framework is deployed.

Creating a CRD is just the first step to add custom logic in the Kubernetes cluster. The next step is to create a component that has embedded the domain-specific application logic for the CRD. Essentially, a service provider needs to develop and install a component which reacts to the various events which are part of the lifecycle of a CRD and implements the desired functionality. DLF utilizes the Operator-SDK, an open-source component of the Operator Framework⁹, which provides the necessary tooling and automation in the development of these components in an effective, automated, and scalable way. Operator-SDK is utilized to create the Dataset Operator in DLF. Its main functionality is to react to the creation (or the deletion) of a new Dataset and materialize the specific object. Specifically, when a Dataset gets created, the software stack invokes the necessary Kubernetes CSI plugin and creates a PVC that provides a filesystem view of the bucket in the COS.

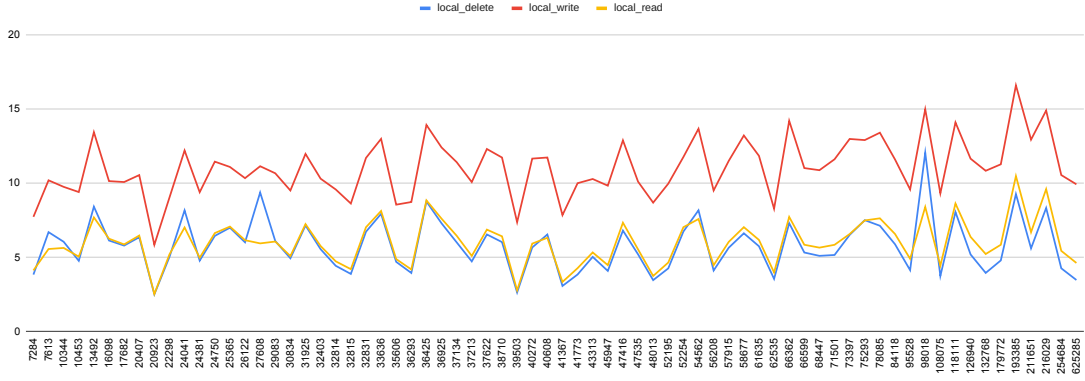


Figure 2: Read, Write and Delete operation response times in milliseconds for the local ESC deployment

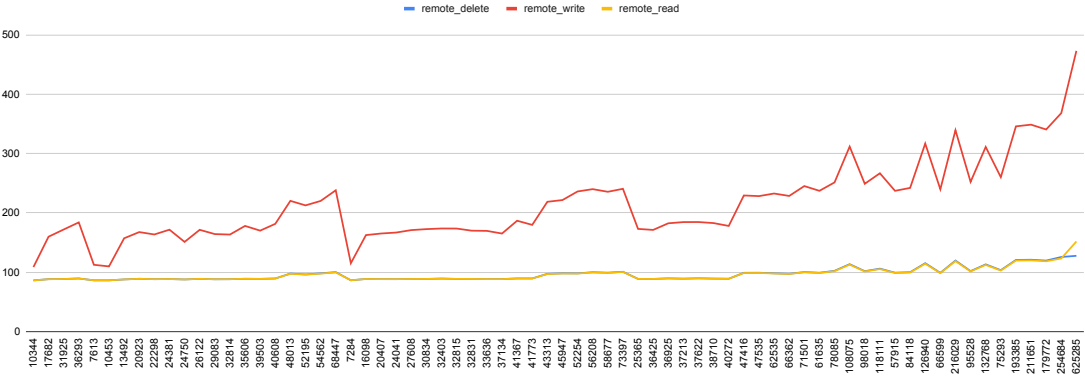


Figure 3: Read, Write and Delete operation response times in milliseconds for the remote ESC deployment.

4 Experimental Evaluation

ESC aims at improving the QoE of the end users by migrating the data "close" to them, thus reducing data transfer delays and network utilization. In order to evaluate the effectiveness of the ESC, a number of resource utilization and QoS metrics are collected using the Prometheus system. The data are collected on the edge, by Prometheus agents running on the edge nodes that handle the data storage. These data are stored in the Prometheus database of each minicloud. More specifically, the data are collected at regular intervals of 5 minutes throughout the functional period of the component, i.e. for the whole duration that the edge storage component is active and waiting for serving data requests.

⁹<https://operatorframework.io/>

The evaluation metrics employed are divided into two categories: i) Resource consumption - CPU available (total, used), RAM available (total, used), HDD available (total, used), Network available (total, used) and ii) Performance - Throughput, Data request response time and Network time.

The resource consumption metrics of the first category are all being passively collected by the Prometheus agents placed on storage nodes. The performance metrics of the second category on the other hand, require a client-side approach so they are actively collected only during benchmarks and tests. The evaluation is conducted using two ESC deployments, one in a local and one in a remote minicloud. The behavior of ESC is evaluated using a collection of small to medium binary files ranging from $15KB$ to $10MB$. All these files are forming the evaluation dataset that is stored in various MinIO buckets, created and managed by ESC in the local and remote miniclouds. These buckets are then mounted onto new pods, using the DLF, and these new pods are taking the role of clients, sending data requests to the ESC and recording performance metrics for these requests.

Figure 4 illustrates the percentage change of various resource utilization metrics -CPU Usage, Memory Usage, Available Memory, Disk Write Latency, Disk IO time- during intense data transactions and during normal functionality of the node. As the results suggest, ESC is not overusing the RAM of the node, although it is slightly increasing the

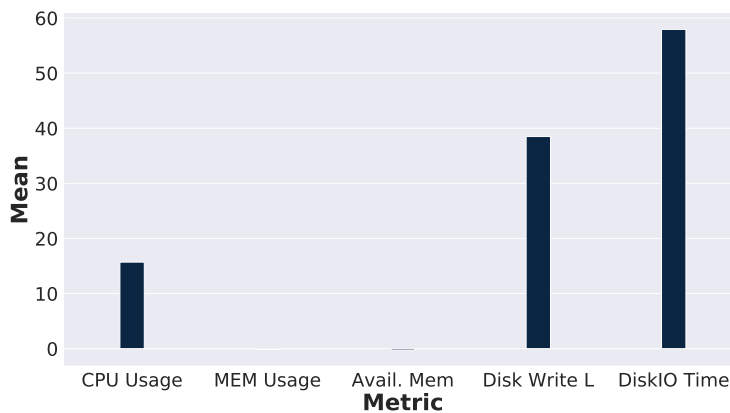


Figure 4: Percentage change of various resource utilization metrics

usage of the CPU and the disk operations, as expected. This proves that ESC is lightweight enough to be deployed on most edge devices, including the Raspberry Pis that are utilized for the evaluation. More specifically, the RAM related metrics are near to zero, meaning almost no change, the CPU metric is slightly increased while the disk metrics are increased by a larger degree, proving intense I/O activity.

Client-side metrics collected to assess the impact of ESC on QoE, are presenting a clearer picture of how ESC improves the response times of various data requests. Figure 2 and Figure 3 demonstrate the comparison between read, write and delete operations for the local and the remote ESC respectively. Due to the object store nature of MinIO, it can be observed that write operations are more time consuming compared to read and delete operations. On the other hand, read and write operations do not differ much compared to each other, the only difference is the network delay for the final file transfer, which is pretty small taking into account that present evaluation tests were conducted using file transfers of multiple small to medium files. The comparison between the different operations are similar but at a different scale; for the local ESC, response times vary between 3 to 17 ms while for the remote ESC, response times vary between 84 to 450 ms. This is becoming more obvious when putting the response times into direct comparison, as illustrated in Figure 5. The request response time for the local ESC is under 20 ms for all file operations which is significantly lower than the remote ESC.

5 Conclusion

The distributed and heterogeneous nature of the edge and its limited resource capabilities pose challenges in implementing an efficient edge storage system. In this paper, we presented a hybrid distributed edge storage component which aims at improving the QoE of end-users by migrating data "close" to them and ultimately reducing data transfer time and network utilization. A minicloud of three nodes is realized in the Kubernetes ecosystem, which consist of a master node/edge server and two Raspberry Pis acting as storage workers. The Dynamic Lifecycle Framework is utilized in order to enable transparent and automated access for containerized applications via mount-points to remote

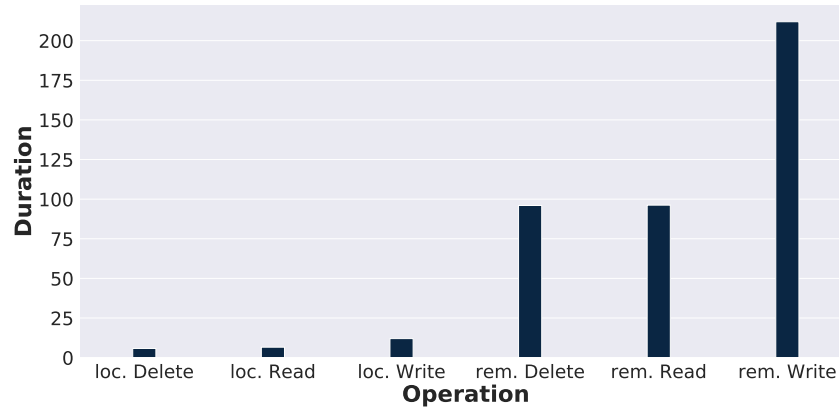


Figure 5: Comparison of response times for various operations for the remote and local ESC deployments

workloads, stored in S3 buckets. The effectiveness of the proposed edge storage component is evaluated by employing a number of QoS and resource utilization metrics, proving two things; a) the lightweight nature of ESC, making it a perfect fit for edge device deployments and b) the great reduction in data request response times, which on some edge use cases is a necessity for their basic functionality.

Acknowledgment

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 101016509 (CHARITY) and No 871793 (ACCORDION).

References

- [1] M. Chiang and T. Zhang, “Fog and iot: An overview of research opportunities,” *IEEE Internet of things journal*, vol. 3, no. 6, pp. 854–864, 2016.
- [2] Z. Hao, E. Novak, S. Yi, and Q. Li, “Challenges and software architecture for fog computing,” *IEEE Internet Computing*, vol. 21, no. 2, pp. 44–53, 2017.
- [3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing—a key technology towards 5g,” *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
- [4] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, “Mobile-edge computing introductory technical white paper,” *White paper, mobile-edge computing (MEC) industry initiative*, vol. 29, pp. 854–864, 2014.
- [5] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
- [6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [7] I. Korontanis, K. Tserpes, M. Pateraki, L. Blasi, J. Violos, F. Diego, E. Marin, N. Kourtellis, M. Coppola, E. Carlini *et al.*, “Inter-operability and orchestration in heterogeneous cloud/edge resources: The accordion vision,” in *Proceedings of the 1st Workshop on Flexible Resource and Application Management on the Edge*, 2020, pp. 9–14.
- [8] D. Sabella, A. Alleman, E. Liao, M. Filippou, Z. Ding, L. G. Baltar, S. Srikanteswara, K. Bhuyan, O. Oyman, G. Schatzberg *et al.*, “Edge computing: from standard to actual infrastructure deployment and software development,” *ETSI White paper*, pp. 1–41, 2019.
- [9] L. Ferrucci, M. Mordacchini, M. Coppola, E. Carlini, H. Kavalionak, and P. Dazzi, “Latency preserving self-optimizing placement at the edge,” in *Proceedings of the 1st Workshop on Flexible Resource and Application Management on the Edge*, 2020, pp. 3–8.

- [10] A. Makris, A. Boudi, M. Coppola, L. Cordeiro, M. Corsini, P. Dazzi, F. D. Andilla, Y. G. Rozas, M. Kamarianakis, M. Pateraki *et al.*, “Cloud for holography and augmented reality,” in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*. IEEE, 2021, pp. 118–126.
- [11] A. Makris, K. Tserpes, and T. Varvarigou, “Transition from monolithic to microservice-based applications. challenges from the developer perspective,” *Open Research Europe*, vol. 2, no. 24, p. 24, 2022.
- [12] T. Theodoropoulos, A. Makris, A. Boudi, T. Taleb, U. Herzog, L. Rosa, L. Cordeiro, K. Tserpes, E. Spatafora, A. Romussi *et al.*, “Cloud-based xr services: A survey on relevant challenges and enabling technologies,” *Journal of Networking and Network Applications*, vol. 2, 2022.
- [13] I. Psaras, O. Ascigil, S. Rene, G. Pavlou, A. Afanasyev, and L. Zhang, “Mobile data repositories at the edge,” in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.
- [14] B. Confais, A. Lebre, and B. Parrein, “Performance analysis of object store systems in a fog and edge computing infrastructure,” in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXIII*. Springer, 2017, pp. 40–79.
- [15] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system,” in *Designing privacy enhancing technologies*. Springer, 2001, pp. 46–66.
- [16] K. Sonbol, Ö. Özkasap, I. Al-Oqily, and M. Alogaily, “Edgekv: Decentralized, scalable, and consistent storage for the edge,” *Journal of Parallel and Distributed Computing*, vol. 144, pp. 28–40, 2020.
- [17] A.-C. Nicolaescu, S. Mastorakis, and I. Psaras, “Store edge networked data (send): A data and performance driven edge storage framework,” in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.
- [18] I. Lujic, V. De Maio, and I. Brandic, “Efficient edge storage management based on near real-time forecasts,” in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2017, pp. 21–30.
- [19] J. Xing, H. Dai, and Z. Yu, “A distributed multi-level model with dynamic replacement for the storage of smart edge computing,” *Journal of Systems Architecture*, vol. 83, pp. 1–11, 2018.
- [20] L. Baresi and D. F. Mendonça, “Towards a serverless platform for edge computing,” in *2019 IEEE International Conference on Fog Computing (ICFC)*. IEEE, 2019, pp. 1–10.
- [21] Y. Gkoufas and D. Y. Yuan, “Dataset lifecycle framework and its applications in bioinformatics,” *arXiv preprint arXiv:2103.00490*, 2021.
- [22] P. Koutsovasilis, S. Venugopal, Y. Gkoufas, and C. Pinto, “A holistic approach to data access for cloud-native analytics and machine learning,” in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 654–659.