DRAFT Position Paper: Edge Clouds Multiple Control Planes Data Replication Challenges

hidden for double-blind review purposes

***Abstract*—Fog computing is an emerging paradigm aiming at bringing cloud functions closer to the end users and data sources. Traditional DC-centric DevOps paradigms established for cloud computing should also span to Edge clouds, which expects them being always manageable and available for monitoring/alerting, scaling, upgrading and applying security fixes. That is a challenge as classic methods do not fit edge cases. Data synchronization is also a common issue for IaaS/PaaS platforms and Mobile Edge Computing environments hosted there. In this paper, we aim at initiating discussions around that challenge. We define operational invariants for Edge clouds based on the Always Available autonomy requirement and related state-of-art work. We evaluate global/replicated causal consistent data stores and middleware as possible match for meeting those invariants. We point out a great opportunity for designing causal consistent systems for edge cases as a common solution for cloud infrastructure owners, DevOps/SRE and MEC applications. Finally, we bring vision of Replication-as-a-Service, unified design approach to benefit applications, like NFV/StatelessNF, and cloud middleware/APIs. Having RaaS implemented as vendors-agnostic commodity software/storage systems that provide interoperability over hybrid clouds and multiple service providers is the ultimate mission for future work.**

***Index Terms*—Open source software, Edge computing, Network function virtualization, Distributed computing, System availability, Design**

I. Introduction

Hybrid and multi-cloud environments is inevitable future of cloud computing. Interconnected private and public clouds, optionally hosting Platform-as-a-Service (PaaS) solutions on top, like OpenShift/Kubernetes, with its workloads pushed closer to end users, unlock great potential for Mobile Edge Computing (MEC) and massively distributed scenarios. In fog environments, expectations for management, control and operational capabilities are pretty same as to the traditional cloud environments, while DC-centric approaches do not work over wide area networks (WANs) and multiple autonomous control planes. Whereby we can have state of applications or control/management planes synchronized eventually and partially, which requires no strong consistency and no global view for the most of the cases and most of the time. That imposes requirements for specific distributed/replicated data stores and/or middleware to maintain constant availability for centralized control/management planes without violating the requirement of local manageability of systems/applications. These must be available, while being offline or executing a handover between base radio network stations, and recover its state fast upon a crash. All that requires smart, WAN-optimized state transfers and resolving of possible conflicts caused by concurrent updates or applying locally cached operations after a network drop out ends. We aim at initiating debates on these challenges through numerous communities, foundations and project groups dedicated to building solutions for MEC, Network Function Virtualization (NFV) and other edge computing cases that involve multiple control planes and multi-site or multi-cloud operations.

II. Background Concepts

Aside of the established terms [3], we define a few more for the data processing and operational aspects:

Deployment Data: data that represents the configuration of *cloudlets* [3], like API endpoints URI, or numbers of deployed *edge nodes* [3] in *edge clouds* [3]. That data may represent either the most recent state of a deployment, or arbitrary data chunks/operations required to alter that state. When there is unresolved data merging conflicts or queued operations pending for future execution, the most recent state becomes the *best known* one.

Cloud Data: similarly to deployment data, represents the most recent or the best known internal and publicly visible state of cloudlets, like cloud users or virtual routers. Cloud data also includes logs, performance and usage statistics, state of message queues and the contents of databases. It may as well represent either data chunks or operations targeted for some state

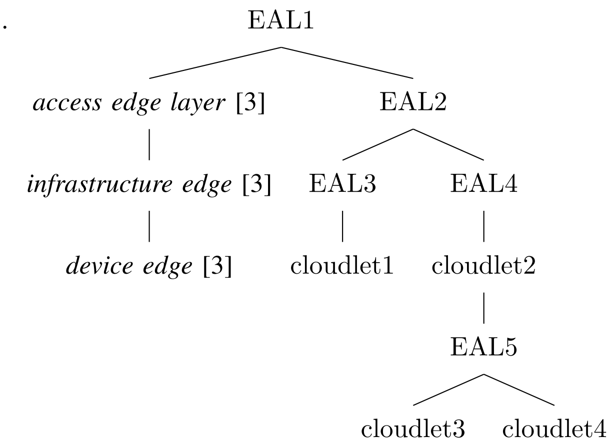
S transitioning into a new state S′.

Control Plane: corresponds to any operations performed via cloudlets API endpoints or command line tooling. E.g. starting a virtual machine instance or a Kubernetes pod, or creating an OpenStack Keystone user. Such operations are typically initiated by cloud applications, tenants or operators.

Replication Topology: represents a graph for allowed state replication flows and targets for operations on interconnected cloudlets, including *edge aggregation layers* [3]. For hierarchical topologies, there would be a limitation for horizontally interconnected cloudlets cannot be replicating its state nor targeting operations to each other. This corresponds to the acyclic graph (tree) topology. There may be also P2P topologies [9] or mixed cases.

For the example tree (Fig. 1), the $\mathrm{infrastructure\ edge}$ can replicate data, like contributing its local stats into the global

Fig. 1. Example Replication Topology



view of hardware and performance metrics, to the the edge aggregation layer EAL1 omitting the access edge layer. Meanwhile the latter can be pushing something, like deployment data changes, to the device edge and infrastructure edge. On the right side of the graph, cloudlet3 and cloudlet4 cannot replicate to each other, but can do to the edge aggregation layer EAL1 either directly or consequently via EAL5, cloudlet2, EAL4 and EAL2, which is totally the replication implementation specific. We also say that EAL2 and EAL1 are hierarchically upper associated with EAL4 and EAL5, or just EAL1 is an *upper layer* for EAL2, when we mean that the latter is a potential subordinate of the former. Finally, we can say that EAL4 controls/manages cloudlet2 and anything sitting down of it.

That is, a control/manage-subordinate hierarchical association only defines influence domains and does not imply state replication as a mandatory thing nor imposes bidirectional data synchronization, but rather provides an opportunity of one-way or bidirectional state replications and regulates a possibility of issuing control/management operations. State replication may be performed from subordinates to its upper associates, think of sending reports. Or vice versa, think of sending directives to subordinates. While control/management operations may only flow up-down, think of planning future work for subordinates.

Management Plane: corresponds to administrative actions performed via configuration and lifecycle management systems. Such operations are typically targeted for cloudlets, like edge nodes, *edge datacenters* [3], or edge clouds. E.g. upgrading or reconfiguring cloudlets in a *virtual datacenter* [3], or scaling up edge nodes. And typically initiated by cloud infrastructure owners. For some cases, like Baremetal-as-a-Service, tenants may as well initiate actions executed via the management plane. Collecting logs, performance and usage statistics for monitoring and alerting systems also represents the management plane operations, although it operates with the cloud data.

When we refer to an edge aggregation layer and cloudlets under its control/management, we mean exactly any operations executed via the control/management planes of that edge aggregation layer and targeted for cloudlets sitting down the nested connections graph. And a replication topology regulates allowed targets for operations and state replication.

Data Replication Conflict: according to [1], two operations on the same target are in conflict if they are not related by causality.

Always Available: the operational mode of the control and management planes that corresponds to *sticky available causal consistency* [4] data replication models, i.e. RTC (*Real-Time Causal* [2]), or *causal*+ [1]. Depending on the consistency model choices, there may be additional constraints:

- the stickiness property is a mandatory constraint for sticky available causal consistent systems. That is: “on every non-faulty node, so long as clients only talk to the same servers, instead of switching to new ones” [4].
- the real-time constraint is keeping the system time synchronized for all cloudlets. That is a mandatory constraint for RTC.
- *one-way convergence* [2] is a mandatory for RTC.

Causal+ and RTC consistency ensure ordering of relative operations, i.e. all causal related writes can be seen in the same order by all processes (connected to the same server). It is also known that “the causal consistency supports non-blocking operations; i.e. processes may complete read or write operations without waiting for global computation. Therefore, the causal consistency overcomes the primary limit of stronger criteria: communication latency” [6]. All that provides the best causal consistency guarantees we can get for today for always available systems.

III. Analysis and Discussion

A. Autonomy Requirements

We define always available autonomy for cloudlets as the following strict requirements:

1) Maintain multiple control/management planes: Autonomous cloudlets may only operate as always available when having multiple control/management planes. For example, when an offline cloudlet cannot start virtual machines or containers, that violates the autonomy requirements.

2) Provide no read-only or blocking access for inter-cloudlet operations: Any operations performed on cloudlets state, despite its aliveness/failure conditions as it’s shown in the global view of upper edge aggregation layers, fit consistency models that allow the involved control/management planes operating as always available, and there is no limitations, like read-only or blocking access. Operations may be queued for future processing in order to meet this autonomy requirement though. Internal state of cloudlets is allowed to keep its failure modes unchanged. E.g. its DC-centric quorum requirements still apply for internal data transactions. That is a transitioning requirement until the internal cloudlets state can be migrated to causal consistent data storages as well.

3) Provide a local control/management plane, whenever possible: Operations on cloudlets can be scheduled at any given moment of time. For offline/partitioned autonomous cloudlets, that can be done via local control/management plane, if it exists and not failed. The same transitioning considerations for internal cloudlets state apply.

4) Support fully autonomous (offline) cloudlets: Aggregation edge layer cloudlets should allow for arbitrary or all of its managed/controlled cloudlets running fully autonomous long-time or indefinitely long, queuing any operations targeted for such cloudlets. That is, to have failure domains size of a 1. For a permanently disconnected cloudlet it may make more sense though to detach it from its adjacent aggregation layer and/or reorganize its place taken in the replication topology.

5) Queue operations to keep it always available at the best effort: Queued operations have to be eventually replayed if/after the control/management plane capabilities restored, or dropped (e.g. expired) otherwise. That poses a delayed replication principle. If there is intermediate aggregation edge layers down the way to the target of the queued operations, the queue may be shared across each of the involved aggregation edge layers or optionally, queued operations may be distributed across not shared queues. That should reduce the associated memory and disc pressure for aggregation layers. Note that we induced no ordering constraints for the operations replayed from queues, that should be defined on case-by-case basis and ideally maintain ordering of causal related operations only, like create a VM and snapshot that VM. Global unique IDs may be a good fit for maintaining such a causal ordering.

6) Provide a global view for cloudlets aliveness state: Global view of cloudlets needs to be periodically presented for at least one of upper aggregation edge layers. For example, with the state marks, like "unknown", or "autonomous"; "synchronizing", "connected"; "failed", or "disconnected", or "fenced", if and only if it is confirmed as failed, or manually disconnected, or fenced automatically. That poses the aliveness of the control/management planes principle.

B. Operational Invariants

To be always available as we defined it, control and management planes of cloudlets should provide the following operational capabilities (*invariants* hereafter):

1) Keep control planes always available at best effort: CRUD (Create, Read, Update and Delete) operations on cloud data can always be requested via API/CLI of local cloudlets or upper layers. The same queuing requirements apply as it is defined for the autonomy requirements.

2) Do not wait for edge aggregation layers control planes for local operations: Local CRUD operations for offline cloudlets, if its control plane exists and not failed, can be processed without waiting for the upper aggregation layers to recover its control over the cloudlets. When a cloudlet is running only compute/storage resources, it cannot meet this requirement.

3) Allow local scaling of infrastructure edge nodes without waiting for management planes of upper aggregation layers: Similarly the to control plane operations, deployed infrastructure edge nodes can always be scheduled for scaling up/down by the cloudlets local management planes, if it is possible (a cloudlet may be relying on the remote configuration management only), or via its upper layers. Same queuing requirements apply for operations.

4) Allow hotfixes and kernel/software updates applied locally for cloudlets: Security patches and minor system updates, including kernel upgrades, can always be scheduled for installation by the same meanings (via operations issued locally or by the associated aggregation layers, including the same queuing requirements).

5) Allow major software versions upgrades applied locally for cloudlets: Similarly, major versions of system components, like OpenStack or OpenShift/Kubernetes platforms, can be always scheduled for upgrades, using the same meanings as above.

6) Provide an extended global view for cloudlets: Additionally to the aforementioned global view for cloudlets control/management plane aliveness state marks, there needs to be a periodically updated global view for each of the edge aggregation layers into its controlled/managed cloudlets, at least the adjacent ones, for the key system administration aspects, like hardware status, power management, systems state logging, monitoring and alerting events, performance and metering statistics.

C. State Replication Consistency Requirements

As it follows from the defined always available autonomy requirements and operational invariants, we define the following data replication requirements[1]:

1) Incorporate convergent conflict handling [1]: Data replication conflicts can be resolved automatically, or by hand and maintained as causal related. The conflicts resolving strategies and rules should be customizable, like "last writer wins" or "return them all". After the conflicts resolved, the data may be considered causal related, that is by definition [1] of the data conflicts in eventually consistent systems.

2) Prefer one-way convergence in replication topologies: As far as the replication topology and queuing capabilities allow that, causal related data can be replicated across cloudlets. Prefer one-way convergence and avoid bidirectional replication whenever possible.

3) Bidirectional replication is only a nice to have requirement: Bidirectional (two-way convergence) data replication is not a strict requirement but is nice to have. Indeed, some state needs to be replicated one-way from aggregation edge layer to cloudlets under its control/management, like virtual machine or hardware provisioning images data. While logs, performance and metering statistics may be collected only from cloudlets to its upper layers.

OpenStack/Kubernetes, have yet support for neither causal consistent storage backends for its cloud/deployment data, nor

[1] when we refer to *data* or *state*, we do not differentiate either that is deployment/cloud data, internal state of an NFV application or cloud API operations. That poses the **unified approach principle**

middleware that could drive replication of casual related state[2]. OpenStack cloud data is stored in a distributed database[3] based on *unavailable* [4] strongly consistent models, e.g. *serializable* [4]. In OpenShift/Kubernetes clusters state is backed with a KVS data store, which also supports only the strong consistency. These does not scale across datacentres, which poses an open opportunity for designing causal consistent state replication systems.

From the other side, the *total available* [4] consistency models weaker than causal or eventual, leave the programmers of end systems to deal with *fuzzy reads* [4], *phantoms* [4], discarded write-only transactions or empty state returned for any reads.

D. Data Replication Challenges

All that brings us to challenges that need to be addressed for multiple control/management planes:

- identifying types of control/management operations and replicated data, when grouping those into particular replication topologies. Such groups may be identified by multiple metrics, like communication latency, tolerated duration of network partitions for offline cloudlets, one- or two-way convergence based replication of either low-level data or operations at the higher API-to-API levels.
- the unified replication topology should not bring excessive operational overhead, like maintaining all of the identified types of operations simultaneously for the end system, and require not too much of human care.
- picking the right replication topologies for each of the involved system components, like an identity provider or images streaming services. For example, would a one-way convergent per-key causality (provided by distributed systems, like Riak[4] or Cassandra[5]), fit the data replication needs for a distributed cloud identity service? Or should caching of images maintain a peer-to-peer mesh topoligy replicated across cloudlets?
- programming the conflicts-free shims or "smart" conflicts resolvers based on the picked replication topologies. The "last writer wins" may be a good fit for some basic cases for objects stored in distributed databases/KVS, while "return them all" could benefit the more advanced cases, but might require assistance of artificial intelligence.
- keeping the state replication topologies always efficient and adjusting itself dynamically. E.g., when executing a handover for a mobile subscriber in a 5G network, an orchetrator must identify the adjacent endpoints based on the subscriber location and define the numbers of replicas to place there. Or distributing the queued control plane operations targeted for the associated offline cloudlets might require more of the nested edge aggregation layers or peers to be added dynamically or statically, when the load exceeds hardware capabilities of a particular aggregation site.
- programming middleware that abides the unified approach and fits the targeted Replication-as-a-Service (RaaS) cases for IaaS/Paas control and management planes as well as the edge-native, like NFV, workloads. For the most of the cases, one size does not fit all, so future RaaS solutions should be tightly coupled with its suggested replication topologies.
- **Open questions**: what of the known Edge computing cases common for OpenStack IaaS (and/or Kubernetes PaaS) and NFV applications, can be expressed via one-way converged data replication, which simplifies implementation of end-to-end solutions? For the remaining two-way convergent cases, may an API-to-API based synchronization fully address the needs for bidirectional data replication?

E. Vision of a Unified Deployment/Cloud State Replication Design

The definition we made for always available distributed systems self-explains why the causal consistent state replication is the best match for the massively distributed cloudlets autonomy requirements and operational invariants as we defined those.

The vision of the unified architecture for future state replication tooling imposes it should be solving the multiple control/management planes data synchronization problems for IaaS, PaaS and end users consuming it as RaaS. Although generic version control systems, like Git, might fit all cases for deployment data replicating and conflicts resolving, that would break the unified design approach for cloud data/state replication.

Client libraries implementing causal consistent data replication and customizable conflicts resolving rules may provide a unification layer for different underlying databases/KVS (Key Value Storage). The replication will be effectively acting as database/KVS-to-database/KVS data synchronization tooling syncing data at a database level. The main benefit for such an approach is no a shared data storage needed. Instead, the underlying local to cloudlets data storages may keep operating as is, share nothing and provide unavailable consistency models stronger than causal consistency. And cloudlets may keep using different solutions for its local data storages as far as the state replication tooling may support such backends.

Additionally, client libraries may replicate not only data but operations at an API level as well, i.e. resolve conflicting operations on-fly, then apply the resulting causal related operations for its original targets, effectively replicating changes at an API level. Operations queued by the control/management planes, including those targeted for offline cloudlets, may be also processed that way.

[2] StarlingX:https://storyboard.openstack.org/#!/story/2002842 builds on top of the current strongly consistent database backend

[3] See an evaluation of CockroachDB:https://www.cockroachlabs.com/docs strongly consistent KVS data store:https://github.com/BeyondTheClouds/juice

[4] https://docs.riak.com/riak/kv/latest/learn/concepts/

[5] https://github.com/wlloyd/eiger

IV. RELATED WORK

COPS [1] provides theoretical fundamentals for causal+ consistency and focuses on highly scalable middleware libraries implementing causal consistent data operations. The similar approach is taken for Indigo middleware [10] that gives strong fundamentals on creating application-centric programming methodologies that leverage invariant-repair/violation-avoidance techniques and rely on immutable data structures (CRDTs). Ultimately, that should help programmers to enforce Explicit Consistency by extending existing applications logic and building middleware libraries that operate on top of the underlying causal consistent storage backends. That is, like Walter [11] or SwiftCloud [12].

Walter [11] KVS uses a set-like CRDTs. It provides a strong consistency guarantee within a site and causal ordering across sites. It introduces Parallel Snapshot Isolation (PSI) property that extends snapshot isolation by allowing different sites to diverge with different commit orderings. This is also known as sticky available causal consistency model, where applications should avoid changing its connection endpoints and maintain sticky sessions. Walter performs asynchronous replication across multiple sites and supports partial replication for multi-cluster scenarios to address scalability bottlenecks.

SwiftCloud distributed object database [12] brings geo-replication to the client machines instead. It supports interactive transactions that span multiple CRDT types. To its authors' knowledge, asynchronous replication systems ensure fault-tolerant causal consistency only within the boundaries of the DC, while SwiftCloud guarantees convergent causal consistency all the way to resource-poor end clients. SwiftCloud also proposes a novel client-assisted failover mechanism that trades latency by a small increase in staleness of data.

Global/stretched Causal Consistent Databases [6] work presents potential applications and databases that could use the causal consistency and shows possible methods to implement that model. It also compares serializability, eventual and causal consistency using a running example. There is an important conclusion that to the authors knowledge there is no commercial or mature systems using the causal consistency.

Bolt-on [13] shim takes another approach and allows to leverage existing production-ready eventually consistent data stores virtually upgrading it to provide convergent causal consistency.

STACK Research Group [8] provides a list of the features required to operate and use edge computing resources. The listed requirements are complementary to this work and represent the operational invariants approached from OpenStack developers and operators (DevOps) angle, the view point that also covers interoperability between multiple operators. The latter is an important requirement for hybrid clouds and NFV Edge cases, like Virtual Customer Premises Equipment (vCPE).

In Mobile Edge Computing environments, there is also high demand for novel lightweight data replication and applications live migration solutions. Those must perform well over WAN and not being DC-centric. Proactive data replication techniques to cope with user mobility considered in the related work [14]. It poses challenges of efficient scheduling of data replicas over the edge nodes. According to ETSI reference architecture, MEC Orchestrators are responsible for solving these problems via user mobility prediction, while virtualization infrastructure management (VIM) should implement the proposed procedures of proactive migration. It is notable that containerbased VIMs are considered the most promising solutions for MEC environments, mostly due to reduced footprint of containers. Virtualized 5G in Evolved Packet Core (vEPC) Architectures [15] serves another good example of high demand emerging from the world of Telcos. The work describes a state sharing mechanism across different datacenters that leverages Edge Synchronization Protocol(ESP) and Abstract Syntax Notation.1 (ASN.1). None of these two works mention causal consistency but it reads between the lines as the such, that answers the questions, like which state portions should be replicated, to which cloudlets and under which conditions. E.g. predicted users' trajectories or operations involving particular mobile subscribers may help to establish causal relations and produce ultimate replication decisions at the applications level. StatelessNF [16] rearchitects NFV applications to decouple its internal dynamic state, like connections tracked by firewalls, into a low-latency DRAM storage tier, like one provided by the underneath VIM. StatelessNF relies on RAMCloud[6], where all data is stored in DRAM. That provides 100-1000x lower latency than disk-based systems and 100-1000x greater throughput, which greatly reduces possibility of concurrent updates causing non-mergable data conflicts. While this benefits NFV cases a lot, RAMCloud may not meet well the defined autonomy requirements for control/management planes, where concurrent data updates and cross-datacenter replication is inevitable. In such setups, RAMCloud as a potential RaaS solution has no more its low-latency advantages.

A. Lebre et al. [9] bring vision of a Peer-to-Peer (P2P) OpenStack IaaS Manager, which is opposed to hierarchical distributed topologies. It is stated that P2P saves maintenance costs associated with hierarchies, while the latter also require complex operations in case of failures. The unified IaaS Manager is positioned as an alternative to stretched control plane and federated clouds. It emphasizes on challenges of data locality, efficient cloud storage, interoperability peering agreements, autonomous lifecycle tooling and new edge-native cloud APIs as paramount requirements. Such a view-point naturally complements the vision of unified management plane as we introduced it for this paper.

The edge-cloud native APIs may be also designed with the approaches that Indigo [10] or RainbowFS [7] takes. The latter work focuses on Just-Right (modular) Consistency and flexible service level agreements, like latency requirements, enforced data locality and smart storage placement strategies unique to Fog/Edge platforms and dictated by applications designed for it on case-by-case basis. "Whereby an application pays only

[6]https://ramcloud.stanford.edu

the consistency price that it strictly requires" [7].

- **Open questions**: are there open-source projects to benefit OpenStack/Kubernetes (and NFV apps) for edge cases that fit into multiple autonomous sites? Either mature ones or in active development, implementing a causal consistent database/KVS, like Walter/SwiftCloud, or midleware/shims operating on top of such weakly consistent systems, like RTC/COPS/Indigo/Bolt-on? For example, SwiftCloud is an open-source project available under Apache 2.0 license and might be a good start for future work of adopting it for OpenStack/Kubernetes and edge-native applications as RaaS. Combining it with RAMCloud might end up in a powerful NFV option.
- **Open questions**: what of the stretched/global clusters can support multi-site failure events, assuming at the edge there may be a lot of small failure domains? And which of those can provide two-way convergent causal consistent data replication across edge sites?

V. Conclusion

We defined autonomy requirements for multiple control/management planes of massively distributed Fog environments and imposed operational invariants off it. That brought us to consistency requirements for cloudlets state replication and associated challenges. We introduced a replication topology building concepts. We posed vision of key design principles, like queuing and delayed replication, aliveness of the control and management planes and a unified approach for the middleware tooling. Possible subject solutions may be based on either of global/stretched causal consistent data stores, or DC-centric shims that operate on top of eventually or causal consistent replicated autonomous data stores. Or client-side middleware placed at the edge. That is all to synchronize data low-level. Alternatively, that may be middleware that replicates operations at an API-to-API high-level.

We want to emphasize on the unification principle as the great opportunity for developers to bring the best of IaaS and PaaS worlds for end users whom such data replication tooling might benefit as a service, i.e. DevOps and tenants workloads, like MEC environments and NFV. It is also an opportunity for Telco side to leverage the commodity replication services of infrastructure layers for the hosted virtual network functions (VNFs). And StalessNF benefits from such common RaaS solutions the most. The unification principle also applies to any fog-based system and is not limited to OpenStack, Kubernetes or NFV ecosystems. We only focus on those as a good starting point for the future work.

To draw the line for where/when future work starts, we should remember that "retain workloads operational as the best effort" principle provides a good start for minimum viable products (MVP), like Distributed Compute Node (DCN) scenario. It is also known to be highly wanted by Telco segment for its next-generation Virtual Radio Access cellular Networks (vRAN).

A centralized control/management plane enables early implementations for DCN as it has no autonomy requirements. While post-MVP phases should be evaluated for the most advanced cases, like MEC vEPC, IoT, AI, autonomous aerial drones, big data processing at the edge networks etc. That is where the data replication challenges to fully arise for expensive handover procedures, mobile networks' subscribers state transfers and just generic Day-2 operations of multiple control/management planes of Edge clouds.

Acknowledgment

Thanks to all who reviewed this paper, including but not limited to: Reedip Banerjee, from Red Hat; Flavia Delicato and Paulo Pires, from Federal University of Rio de Janeiro. Special thanks to all of the participants of OpenStack Summit at Berlin, 2018.

References

[1] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Dont settle for eventual: Scalable causal consistency for wide-area storage with COPS," Proc. 23rd ACM Symposium on Operating Systems Principles (SOSP 11), Cascais, Portugal, October 2011.

[2] P. Mahajan, L. Alvisi, and M. Dahlin. "Consistency, availability, and convergence," Technical Report TR-11-22, Univ. Texas at Austin, Dept. Comp. Sci., 2011.

[3] The Linux Foundation, "Open Glossary of Edge Computing," [Online]. Available: https://github.com/State-of-the-Edge/glossary

[4] K. Kingsbury, "Consistency Models," [Online]. Available: https://jepsen.io/consistency.

[5] M. Bayer, "Global Galera Database," [Online]. Available: https://review.openstack.org/600555.

[6] M. M. Elbushra, J. Lindstrom, "Causal Consistent Databases", Open Journal of Databases (OJDB), Volume 2, Issue 1, 2015.

[7] PRCE (Projet de recherche collaborative Entreprises), "RainbowFS: Modular Consistency and Co-designed Massive File" [Online]. Available: http://rainbowfs.lip6.fr/data/RainbowFS-2016-04-12.pdf.

[8] R. A. Cherrueau, A. Lebre, D. Pertin, F. Wuhib, J. M. Soares, "Edge Computing Resource Management System: a Critical Building Block! Initiating the debate via OpenStack", USENIX HotEdge18 Workshop, 2018.

[9] A. Lebre, J. Pastor, A. Simonet, F. Desprez, "Revising OpenStack to Operate Fog/Edge Computing Infrastructures", IEEE International Conference on Cloud Engineering (IC2E), 2017.

[10] V. Balegas, N. Preguia, R. Rodrigues, S. Duarte, C. Ferreira, M. Najafzadeh, M. Shapiro, "Putting Consistency back into Eventual Consistency", Euro. Conf. on Computing Systems (EuroSys), Bordeaux, France, 2015.

[11] Y. Sovran, R. Power, M. Aguilera, and J. Li, "Transactional Storage for Geo-replicated Systems", SOSP'11, page 385-400. New York, USA, 2011. ACM.

[12] M. Zawirski, A. Bieniusa, V. Balegas, S. Duarte, C. Baquero, M. Shapiro, and N. Preguica, "SwiftCloud: Fault-Tolerant Geo-Replication Integrated all the Way to the Client Machine", Research Report RR-8347, INRIA, Oct. 2013.

[13] P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Bolt-on Causal Consistency", SIGMOD'13, pages 761772, New York, USA, 2013. ACM.

[14] I. Farris, T. Taleb, H. Flinck, A. Iera, "Providing ultrashort latency to usercentric 5G applications at the mobile network edge", Transactions on Emerging Telecommunications Technologies, Vol. 29, No. 4, e3169, 2018.

[15] E. Cau, M. Corici, P. Bellavista, L. Foschini, G. Carella, A. Edmonds, and T. Bohnert, "Efficient Exploitation of Mobile Edge Computing for Virtualized 5G in EPC Architectures", MobileCloud, page 100-109. IEEE Computer Society, 2016.

[16] M. Kablan, A. Alsudais, E. Keller, Franck Le,"Stateless Network Functions: Breaking the Tight Coupling of State and Processing", NSDI, 97-112, 2017. 44, 2017.