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 (C/M) 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 specific availability requirements for distributed data stores, or data replication middleware to keep operating as an autonomous and centralized system at once. Systems must retain its availability while being offline 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.

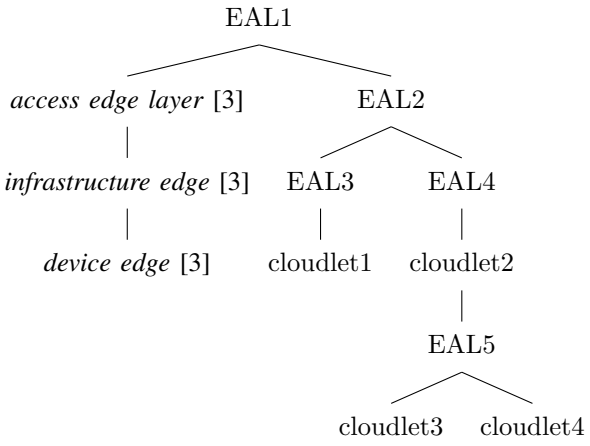
Cloud Data: similarly to deployment data, represents the most recent or the best known internal and publicly visible state of cloudlets, like cloud users or virtual routers. Cloud data also includes logs, performance and usage statistics, state of message queues and the contents of databases. It may 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 VM instance or a Kubernetes pod. 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 view of hardware and performance metrics, to the the edge aggregation layer $\mathrm{EAL1}$ omitting the $\mathrm{access\ edge\ layer}$. Meanwhile the latter can be pushing something, like deployment data changes, to the $\mathrm{device\ edge}$ and $\mathrm{infrastructure\ edge}$. On

Fig. 1. Example Replication Topology



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 C/M-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 C/M 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 C/M 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.

Causal Consistency [6] ensures ordering of relative operations only, where all related writes issued by X can be seen in the same order by all processes connected to all or at least the same server X. Dependencies are tracked when changing versions of objects, like key-value pairs, and checked before making it visible elsewhere. This provides non-blocking operations, i.e. a server (or W and R groups of ones, where $W + R = N$) commits data or returns results without waiting for the remaining $(N - W)$ or $(N - R)$ members. And $W = R = 1$ fully overcomes communication latency - the primary limit of strong consistency. Autonomous cloudlets rely on such non-blocking operations.

Data Replication Conflict: according to [1], two operations on the same target are in conflict if they are not causal related. Such operations require further reconcilation, i.e. resolving the conflict. There is also techniques to avoid such conflicts via immutable Conflict-free Replicated Data Types (CRDTs).

Always Available (AA): the operational mode of the C/M planes or applications that corresponds to the *sticky available causal consistency* [4] model, i.e. *Real-Time Causal*(RTC) [2], or *causal+* [1]. The key constraints and properties are:

- the stickiness property applies so long as clients only talk to the same of total N healty servers, instead of switching to new ones.
- the real-time constraint requires the system time synchronized across cloudlets. That is a mandatory constraint for RTC.
- *one-way convergence* [2] is a mandatory constraint for RTC.

III. Analysis and Discussion

A. Always Available Autonomy Requirements

We define AA autonomy for all cloudlets, including edge aggregation layers, as the following strict requirements:

1) Maintain multiple C/M planes: Autonomous cloudlets may only operate as AA system when having multiple C/M planes. F.e, when an offline cloudlet cannot start virtual machines or containers, that violates the autonomy requirement.

2) Provide non-blocking inter-cloudlet operations: Operations issued by edge aggregation layers are never blocked. Depending on the cloudlets' aliveness state shown in the global view of a particular aggregation layer, operations may be queued for future processing but must return futures for its state tracking. Internally, cloudlets may keep its quorum reguirements and restrict read/write operations, if needed.

3) Provide autonomous C/M plane, whenever possible: For offline/partitioned autonomous cloudlets, operations can be executed via local C/M plane, if it exists and is healthy. Further steps for synchronizing that local state with the aggregation layers or neighbor peers depend on each particular replication topology, like either that is one-way convergent or bidirectional, low-level data or API-level state exchange.

4) Support long-term or permanently offline cloudlets: Aggregation edge layers allow running an arbitrary subset of its managed/controlled cloudlets fully autonomous for indefinitely long but keeping the failure domains as small as possible. For causal consistent data stores this involves sloppy quorums and hinted handoffs [17]. While locally queued API operations may be served in a distributed fashion and/or given expiration timers to alleviate the increased DRAM pressure.

5) Queue operations for the delayed replication: Queued operations will be causally ordered and eventually replayed when the uplink/peer connectivity restored, or expired and dropped otherwise. That poses the *delayed replication* principle. Note that we induced ordering constraint for only related operations, like creating a VM and snapshotting it. Global unique IDs for per-key causality may be a good fit here.

6) Provide a global view for cloudlets aliveness state: Global view into cloudlets' health, performance and monitoring metrics is periodically presented for at least one of upper aggregation edge layers (or neighbor peers). For example, with the state marks, like "unknown", "autonomous", "synchronizing", "connected", "failed", "disconnected" or "fenced". That poses the principle of *aliveness of the cloudlets' C/M planes*. This case may not necessary involve bidirectional data replication and serve a good example of one-way convergent data streams.

B. Operational Invariants

To be always available, C/M planes of cloudlets must meet the following operational capabilities (invariants):

1) Keep C/M planes AA at best effort: Generic API/CLI or configuration managment operations, like software updates, can always be processed by cloudlets locally and by its associated peers or hierarchical upper layers as well. The same queuing and delayed replication considerations as above apply here.

2) Provide an extended global view for cloudlets: Additionally to the aforementioned basic cloudlets' aliveness state marks, there is extended periodic data collected and pushed for the upper edge aggregation layers or peers. At least for the adjacent (next-hop) cloudlets, that covers such metrics as hardware status and power management, logging, monitoring and alerting events, performance and metering statistics.

3) Leverage Replication-as-a-Service solutions for storing state: Replication-as-a-Service (RaaS) allows detaching state from C/M planes and edge-native applications, like StatelessNF [16]. That turns the such into generic stateless workloads easy to manage/migrate across different hybrid clouds and/or orchestration engines. While all the complexity of maintaining those to be AA and synchronized shifts to that particular RaaS solution.

C. State Replication Consistency Requirements and Recommendations

As it follows from the defined AA autonomy requirements and operational invariants, there are associated state replication requirements[1]:

1) Incorporate convergent conflict handling: According to [1], conflicts can be resolved automatically, or by hand and maintained as causal related only after that reconcilation is complete. The conflicts resolving strategies should vary on case-by-case basis. Last-writer-wins (LWW) systems [1] [13] may be best for reconciling operations issued by multiple control planes. While "return them all" [1] [17] strategy better fits the configuration management tasks performed on autonomous cloudlets. Whereby conflicts may have to be reconciled differently when a configuration was either applied locally and is replicated back to the upper aggregation layers, or pushed down after the autonomous cloudlet recovers its uplink connection to the management plane. Or when an NVF application performs a handover for user mobility cases in 5G networks. The source of truth then may be based on distance/placement instead of timestamps. Finally, CRDTs and DRAM stores used all the way, allow minimizing overall conflict handling work in end systems.

2) Prefer per-key causal and one-way convergent replication topologies: Inter-cloudlets data replication is costly and should be as minimal as possible. When it is inevitable, prefer one-way convergence [2] and avoid bidirectional data replication. This greatly simplifies end systems implementation. And per-key causality is well covered in the existing data store systems (see the related work section).

OpenStack/Kubernetes platforms have yet support for causal consistent storage backends nor mature middleware capable driving 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 key-value data store (KVS), which also supports only the strong consistency. These do not scale across multiple datacentres, which poses an open opportunity for adopting of either of the existing causal consistent data stores. Or maybe designing RaaS middleware for inter-cloudlets state exchange over weakly (eventually) consistent data stores.

Beware that the *total available* [4] consistency models are weaker than causal/eventual and 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 C/M planes of edge clouds:

- identifying types of C/M 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.

[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.**

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

[3] See an evaluation of CockroachDB https://www.cockroachlabs.com/docs, an alternative strongly consistent key-value store backend for OpenStack: https://github.com/BeyondTheClouds/juice

- picking the right replication topologies for each of the involved system components, like an identity provider or images streaming services. F.e., an efficient implementation of caching of VMs/containers images may require maintaining a dynamic peer-to-peer mesh topoligy replicated across adjacent cloudlets.
- programming CRDTs for the conflicts-free systems or "smart" conflicts resolvers based on the picked replication topologies.
- 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 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 Edge computing cases involving autonomous multiple control planes can be expressed via one-way convergent causal per-key consistency?
- **Open questions**: for the remaining two-way convergent cases, may an API-to-API state synchronization replace the low-level data replication?

E. Vision of a Unified State Replication As a Service

The vision of the unified architecture for data replication tooling is solving common problems for C/M planes of IaaS/PaaS, end users and edge-native applications. F.e., generic version control systems are well suited for code and might fit all cases for configuration data replication and conflicts resolving, but would violate the unified design approach for other cloud data/state replication cases. For OpenStack, rearchitecting its relational SQL-based datamodel as RaaS is quite challenging, whenever it's implemented as a global KVS cluster or middleware replicating state across multiple KVS clusters. But in the end that would allow the seamless integration of OpenStack and Kubernetes platfroms into a powerful combo to serve Telco NFV cases and edge computing needs.

Client libraries implementing causal consistent data replication and customizable conflicts resolving rules may provide a unification layer for different underlying databases or KV stores. The replication will be effectively acting as database/KVS-to-database/KVS data synchronization tooling syncing data at low-level across multiple DC/cloudlets. 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 C/M planes, including those targeted for offline cloudlets, may be also processed that way.

IV. Related Work

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. Authors of that work stated that there is no commercial or mature systems implementing the causal consistency for a global data store.

There are mature open-source systems that implement Dynamo [17] per-key causal consistency: Riak[4], Cassandra[5] and Voldemort[6]. Dynamo provides an "always writeable" data store where no updates are rejected due to failures or concurrent writes and is conceptually close to a "zero-hop" distributed hash table system. It is DC-centric and assumes all cluster nodes are trusted. To overcome such limitations of a single administrative domain, there is an additional middleware layer may be needed to perform data replication across multiple isolated clusters.

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 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 data stores, 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. 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

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

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

[6]https://github.com/voldemort/voldemort

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.

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 [18], 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 C/M 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 the consistency price that it strictly requires" [7].

- **Open questions**: as RaaS targeted for OpenStack, Kubernetes and NFV applications, what mature (or in active development) open-source causal consistent data stores support multi-site failures and may fit the edge computing cases, given the defined operational invariants? E.g., SwiftCloud and Voldemort are open-source projects available under Apache 2.0 license and might be a good start. Combining it with RAMCloud might end up in a powerful NFV option.

V. Conclusion

We defined autonomy requirements for multiple autonomous C/M planes of Fog environments and imposed operational invariants. That brought us to the causal consistency requirement for inter-cloudlets state replication. We highlighted challenges of building replication topologies and propposed design principles, like low-level data replication and API-level queuing with delayed replication, aliveness of the cloudlets' C/M planes. We brought vision of a unified approach for building ultimate RaaS solutions for Fog/MEC envirnoments. Such solutions may be based on global causal consistent systems, like [11] [12], or middleware shims providing multi-site replication on top of traditional DC-centric data stores. That is, "real" mature systems, like strongly consistent CockroachDB, eventual/per-key causal consistent Riak/Cassandra/Voldemort, or based on theoretical/research [1] [2] [7] [10] [13] systems. With an exception for novel StatelessNF applications, where its data stores must be DRAM based, like [18].

Alternatively, there may be middleware that queues and replicates C/M plane operations at an API-to-API high-level. This might better fit the interoperability requirements of hybrid cloud architectures and simplify end-to-end implementation.

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). The unification principle also applies to any fog-based system and is not limited to OpenStack and Kubernetes platforms or NFV cases. We only focus on those as a possible good start.

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 C/M 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 C/M 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, D. G. Andersen, "Dont settle for eventual: Scalable causal consistency for wide-area storage with COPS", SOSP, 2011.

[2] P. Mahajan, L. Alvisi, 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", IC2E, 2017.

[10] V. Balegas, N. Preguia, R. Rodrigues, S. Duarte, C. Ferreira, M. Najafzadeh, M. Shapiro, "Putting Consistency back into Eventual Consistency", EuroSys, 2015.

[11] Y. Sovran, R. Power, M. Aguilera, J. Li, "Transactional Storage for Geo-replicated Systems", SOSP, 385-400, 2011.

[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, 2013.

[13] P. Bailis, A. Ghodsi, J. M. Hellerstein, I. Stoica, "Bolt-on Causal Consistency", SIGMOD, 761772, 2013.

[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, T. Bohnert, "Efficient Exploitation of Mobile Edge Computing for Virtualized 5G in EPC Architectures", MobileCloud, 100-109, 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.

[17] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store", SOSP 2007: 205-220, 2007.

[18] RAMCloud: https://ramcloud.stanford.edu