Position Paper: Edge Clouds Multiple Control Planes Data Replication Challenges

Bohdan Dobrelia
OpenStack platform
Red Hat
Poznan, Poland
bdobreli@redhat.com

***Abstract*—Fog computing is an emerging paradigm aiming at bringing cloud functions closer to the end users and data sources. Its control plane and deployments data synchronization is a major challenge. Autonomy requirements expect even the most distant edge sites always manageable, available for monitoring and alerting, scaling up/down, upgrading and applying security fixes. Whenever temporary disconnected sites are managed locally or centrally, some changes and data need to be eventually synchronized back to the central site(s) with having its merge-conflicts resolved for the central data hub(s). While some data needs to be pushed from the central site(s) to the Edge, which might require resolving data collisions at the remote sites as well. In this paper, we position the outstanding data synchronization problems for OpenStack platform becoming a cloud solution number one for fog computing. We define the inter-cloud operational invariants based on that Always Available autonomy requirement. We show that a causal consistent data replication is the best match for the outlined operational invariants and there is a great opportunity for designing such a solution for Edge clouds. Finally, the paper brings vision of unified tooling to solve outstanding state synchronization problems the same way for infrastructure owners, cloud operators and tenants running stateful workloads hosted on OpenStack IaaS or OpenShift/Kubernetes PaaS deployed in Edge clouds as multi-cloud workloads abstraction and unification layer, to make it truly cloud-vendors agnostic and portable.**

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

I. Introduction

OpenStack is an Infrastructure-as-a-Service platform number one for private cloud computing, and it becomes being so for fog computing as well. Hybridization and multi-cloud trends for private clouds interconnected with public clouds and Platform-as-a-Service (PaaS) solutions, like OpenShift/Kubernetes, allow the containerization of micro-services oriented workloads to emerge in a highly portable, self-contained and the hosting cloud-agnostic way. Giving it massively distributed scale of fog computing and bringing the data it operates closer to end users, opens great opportunities for Internet of Things (IoT) and 5G telecommunication technologies, which first of all requires low-latency and highly responsive interfaces always available for end users.

Speaking of an always available, back to the system administration realities, the Edge clouds control and management plane capabilities in such a massively distributed world shall not fall behind as well. This paper positions challenges associated with replicating state, like cloud or deployment data and/or operations, and defines operational capabilities to have it always available as the best effort. It finally brings vision of unified systems design approach for future data replication tooling to be implemented for or integrated into OpenStack IaaS, OpenShift/Kubernetes PaaS that may optionally reside on top of it, or as a separate compute platform, and stateful workloads designed for massively distributed scale, which is natural for Edge computing and IoT.

II. Glossary

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[1] of a deployment, or arbitrary data chunks/operations required to alter that state.

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. For example, 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.

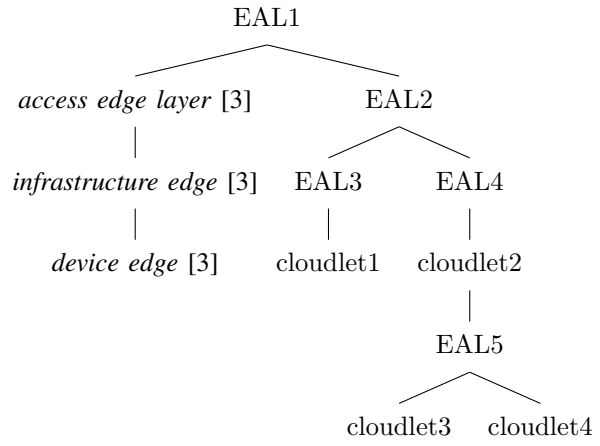
Replication Topology: represents a hierarchy for allowed state replication flows and targets for operations on interconnected cloudlets, including *edge aggregation layers* [3].

So effectively there is only 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.

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

[1] when there is unresolved data merging conflicts or queued operations pending for future execution, the most recent state becomes the best known state

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, when we mean that the latter two are potential subordinates of either of the former two. Finally, we can say that EAL4 controlls/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 data centers* [3], or edge clouds. E.g. upgrading or reconfiguring cloudlets in a *virtual data center* [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 ensures 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:

B. Maintain multiple control/management planes

Autonomous cloudlets may only operate as always available when having multiple control/management planes. For example, when a disconnected cloudlet cannot start virtual machines or containers, that violates the autonomy requirements.

1) Provide no read-only or blocking access for inter-cloudlet operations: Any operations performed on cloudlets state[2] 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[3]. Internal state of cloudlets though, is allowed to keep its failure modes unchanged. E.g. for a local Galera/MariaDB database or Etcd cluster, its standard quorum requirements apply for internal operations. That is a transitioning requirement until the internal cloudlets state can be migrated to causal consistent data storages as well. Operations may be queued for future processing in order to meet this autonomy requirement.

[2]despite the cloudlets aliveness or failure conditions as it's shown in the global view of hierarchically upper associated edge aggregation layers

[3]operations may be queued for future processing in order to meet this autonomy requirement

2) Provide a local control/management plane, whenever possible: Operations on cloudlets can be scheduled at any given moment of time, despite of inter-cloudlets network connectivity[4]. The same transitioning requirements for internal cloudlets state applies.

3) Support fully autonomous or disconnected cloudlets: Aggregation edge layer cloudlets should allow for arbitrary or all of its managed/controlled cloudlets running fully autonomous long-time or indefinitly long, queuing any operations targeted for such cloudlets. That is, to have failure domains size of a 1. For a permanently diconnected 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.

4) Queue operations to keep it always available at best effort: Queued operations have to be eventually replied if/after the control/management plane capabilities restored, or dropped (e.g. expired) otherwise. That poses a lazy 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[5]. That should reduce the associated memory and disc pressure for aggregation layers.

5) Provide a gloval view for cloudlets aliveness state: Global view of cloudlets needs to be periodically presented for at least one of the hierarchically upper associated 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.

C. 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 hierarchically upper associated aggregation layers. The same queuing requirements apply as it is defined for the autonomy requirements.

2) Do not wait for edge agregation layers control planes for local operations: Local CRUD operations for disconnected/autonomous cloudlets, if its control plane exists[6] and is not failed, can be processed without waiting for the upper aggregation layers to recover its control over the cloudlets.

3) Allow local scaling of infrastructure edge nodes without waiting for management planes of edge 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[7], or via its hierarchically upper associated aggregation layers. Same queuing requirements apply.

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.

D. State Replication Consistency Requirements

As it follows from the defined always available autonomy requirements and operational invariants, we define the following data replication requirements[8]:

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

[4]for disconnected/partitioned cloudlets, data can be modified via local control/management plane, if it exists and not failed. Despite the quorum requirements of its hierarchically upper associated aggregation edge layers

[5]or optionally, queued operations may be distributed across not shared queues

[6]a cloudlet may be running only compute/storage resources. For such a case it cannot meet the always available requirements, when disconnected/partitioned from its remote control plane

[7]a cloudlet may be relying on the remote configuration management only

[8]when we refer to just *data* or *state*, we intentionally do not differentiate either that is deployment or cloud data, or queued API/CLI operations, to be replicated/replayed for management or control planes. That poses the **unified approach principle**

from cloudlets to its hierarchically upper associated aggregation edge layers.

OpenStack and OpenShift/Kubernetes, have yet support for neither causal consistent data backends[9], nor client libraries that could drive replication of casual related state. That poses an open opportunity for developers and system architects to design and implement such state replication tooling for multiple control/management planes.

OpenStack cloud data is normally stored in databases via transactions based on stronger than causal *unavailable* [4] data consistency models, e.g. *serializable* [4], or *repeatable read* [4]. OpenShift/Kubernetes clusters state, some of SDN (Software Defined Network) solutions are backed with Etcd, which also only supports the stronger than causal consistency models. Those cannot tolerate high network latency, serve two-way convergence only, therefore do not scale for potentially dozens of data replicas.

From the other side, the weaker consistency models, which is *total available* [4], provide a poor alternative that brings greatly increased implementation complexity. E.g. *monotonic atomic view* [4] would require handling of *fuzzy reads* [4], *phantoms* [4], discarded write-only transactions, empty state returned for any reads.

E. Data Replication Challenges

All that brings us to challenges that need to be addressed for multiple control/management planes[10]:

- categorizing control/management operations and data flows associated with it, then grouping those into particular replication topologies. Such groups may be identified by multiple metrics, like communication latency, network partition duration tolerance for disconnected/autonomous cloudlets, one- or two-way convergence based, a shared causal data storage or client libraries implementation specific. For the latter case, either it should allow replicating data low level or synchronizing in-flight/queued operations at API level, or both. And for which data storage and message queuing backends to support such replication mechanisms[11]. Finally, that grouping should also be done for built-in state replication capabilities of a component/backend[12] versus if it requires external assistance from state replication tooling.
- in the end, the final design should not bring excessive operational overhead, like if maintaining all of the identified replication topologies simultaneously for a deployment, and require not too much of human care, but still meet the unified approach principle as we defined it earlier.
- picking/combining identified replication topologies to use with each of the involved system components, like identity provider or images serving services. Ideally, (almost) stateless components will not require data replication aside of built-in capabilities of a single distributed control plane.
- designing strategies and rules for conflicts resolving based on picked replication topologies identified the previous steps. A "last writer wins" may be a good fit for database/KVS conflicting data synchronization, while "return them all" seems the best choice for manual or artificial intelligence driven/assisted "smart" conflict resolvers.
- keeping state replication topologies efficient, e.g. distributing locally queued operations targeted for disconnected cloudlets by nesting more of the edge aggregation layers underneath. Also, stateless or almost stateless components may be happy with only built-in capabilities of a single distributed control plane.
- abiding the unified approach/architecture principles for Edge clouds IaaS, optional Paas, and edge-native workloads as well.

F. 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 Replication-as-a-Service. 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[13] as well, i.e. resolve conflicting

[9]that is, data backends used for the cloud/deployment data storage and not for the data plane workloads needs

[10]due to complexity and enormous amount of required changes for all IaaS/PaaS cloud components associated with that, such advanced state replication solutions should be the next iteration after a minimum viable product (MVP) perhaps

[11]for OpenStack/Kubernetes, that is Etcd, MySQL/MariaDB Galera, RabbitMQ/Qpid

[12]for example, depending on the upper latency and replica numbers constraints, a single control and management plane may fit all the cases of some virtual data centers [3]

[13]For OpenStack Nova, there had been an example for such an API level replication, that is cells v1 protocol. But it had been deprecated as real cells v1 deployments required constant human care and feeding operationally

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 disconnected/autonomous cloudlets, may be also processed that way.

COPS formally proves implementation of a client library and highly scalable tooling for causal+ data operations. By design, it does not impose any real-time constraints and supports a single edge data center failure. The real tooling made off that base, may be operating on top of the not shared local cloudlets databases, or KVS, that provide the stronger consistency guarantees by the costs of reduced local availability[14]. That would work as weaker consistency guarantees work well, when built on top of the stronger ones, and provide an always available global view of cloudlets for hierarchically upper associated aggregation edge layers. Replicating the state changes via causal related operations and conflicts resolving via custom handlers is that COPS covers as well.

Global causal consistent databases [6] is an alternative[15] approach to client libraries. That is the most unified and also the most simple way of exposing it for edge-native workloads as Replication-as-a-Service. The downside is such a database has to be supported as a control/management planes data backend for IaaS and PaaS itself and local cloudlets would have to be switched to use that data backend globally as well.

- **Open questions**: does COPS work for multiple data centers failure events, up to failure domains of a size of a 1?
- **Open questions**: is COPS applicable for two-way convergent systems, in terms of [2], for bidirectional causal+ replications?
- **Open questions**: how much of all of the cloudlets data replication cases can be performed one-way? That would drastically simplify future implementation: "Although most implementations use bidirectional communication, the communication from the update-receiver to the update-sender is just a (significant) performance optimization used to avoid redundant transfers of updates already known to the receiver. One-way convergence is also important in protocols that transmit updates via delay tolerant networks" [2].
- **Open questions**: what solutions can we propose for operations that still do require a bidirectional data synchronization, like unique cloud users ID, without breaking the always availability autonomy requirements for the control and management planes? Causal consistent databases [6] may be a good choice for that. Alternatively, instead of bidirectionally replicating such data, API-to-API based synchronization mechanisms may become a replacement for such cases.
- **Open questions**: which of the existing causal consistent databases [6] can be ingegrated as a global database/KVS solution that fits all?
- **Open questions**: which of the existing client library tooling for causal database or API level replication, like RTC or COPS, can be ingegrated for at least OpenStack and Kubernetes control/management planes to operate on top of Etcd and Galera clusters, RabbitMQ broker or Qpid distributed routers, or API endpoints?

[14]that is, the local view for a cloudlet and have no impact onto global views

[15]if it may be a truly global one, while scaling well and providing two-way convergence over high latency networks. Or otherwise, multiple shared instances of it, each serving its dedicated a virtual data center

IV. Conclusion

We defined autonomy requirements for multiple control/management planes of massively distributed edge clouds and imposed operational invariants off those autonomy requirements. That brought us to consistency requirements for cloudlets state replication and associated challenges. Finally, we posed vision of key design principles, like queuing and lazy replication, aliveness of the control and management planes and a unified approach for the subject tooling. That is, state replication tooling to be based on either shared causal consistent databases, or client libraries that replicate not shared local data low level. Or client libraries that replicate operations at an API level.

We want to position the unification principle as the most important thing and the greatest opportunity for developers to do it "the right way", which is to bring the best of two IaaS and PaaS worlds for end users whom such data replication tooling might benefit as a service, i.e. cloud tenants, infrastructure owners and operators.

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.