Edge Clouds Control Plane and Management Data Consistency Challenges: Position Paper for IEEE International Conference on Cloud Engineering, 2019

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

***Abstract*—Fog computing is emerging Cloud of (Edge) Clouds technology. 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 causally consistent data storage 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 the vision of unified tooling to solve the data synchronization problems the same way for infrastructure owners, IaaS cloud operators and tenants running workloads for PaaS, like OpenShift or Kubernetes deployed on top of Edge clouds.**

***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 Mutlicloud trends for private clouds interconnected with public clouds and Platform-as-a-Service (PaaS) solutions, like OpenShift/Kubernetes, allow the containerization of microservices 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 nextgen global telecommunication technologies, which first of all requires low-latency and higlhy responsive interfaces always available for end users.

Speaking of always available, back to the system administration realities, the Edge clouds control and management plane capabilities in such a perfect world shall not fall behind as well. This paper is about to position the associated data replication challenges and to bring vision of future development trends on that topic, both for OpenStack and hopefully for anything residing on top of it, e.g. PaaS and/or workloads designed for massively distributed scale and following the IoT/fog computing best practices.

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 represents the most recent state of a deployment.

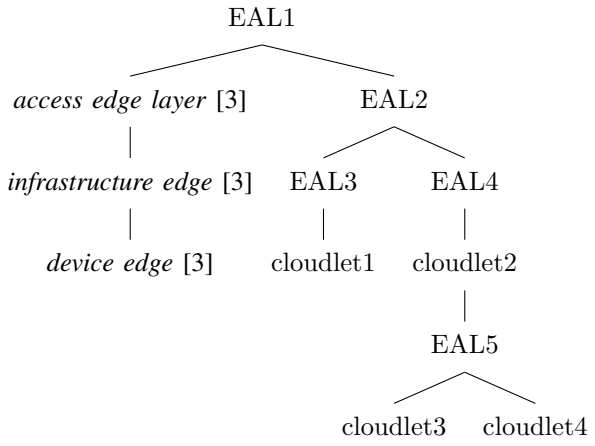
Cloud Data: represents the most recent[1] internal and publicly visible state of cloudlets, like cloud users or virtual routers. Cloud data also includes logs, performance and usage statistics, state of message queues and the contents of databases.

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 hierarchy for allowed data replication flows for interconnected cloudlets, including *edge aggregation layers* [3]. When we refer to an edge aggregation layer and cloudlets under its control, we mean exactly any operations executed via the control plane of that edge aggregation layer and targeted for cloudlets sitting the next hop (which is adjacent up or down on the connections graph), or going deeper through the nested graph connections). If the next hop is represented by another edge aggregation layer, the same applies to its adjacent cloudlets.

[1] when there is unresolved data merging conflicts, the most recent state becomes the best known state

Fig. 1. Example Replication Topology



So effectively there is only a limitation for horizontally interconnected cloudlets cannot be replicating its state to each other. This corresponds to the acyclic graph (tree) topology.

For the example graph (Fig. 1), the 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 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 can also see that EAL4 is hierarchically associated with EAL2 and EAL1, which is under control and/or management of either of two. But it is not associated with EAL3, cloudlet2 or anyting sitting down of two.

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* [3] and cloudlets under its management, we mean exactly administrative/deployment related operations executed via the management plane. Similarly to the control plane, we impose the same data replication topology without nesting limitations but restricted only for horizontal graph connectiongs.

Data Repliciation 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 causally 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 causally related writes can be seen in the same order by all processes (connected to the same server). All that provides the best causal consistency guarantees we can get for today for always available systemts.

III. Analysis and Discussion

A. Autonomy Requirements

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

- any operations performed on cloudlets state[2] fit data consistency models that allow the involved control/management planes operating as always available, and there is no limitations, like read-only or blocking access.
- cloudlets data can be modified at any given moment of time, despite of inter-cloudlets network connectivity[3].
- aggregation edge layer cloudlets should allow for arbitrary or all of its managed/controlled cloudlets running fully autonomous long-time, having all the targeted outgoing operations queued.
- the queued operations have to be eventually applied if/after the control/management plane capabilities restored, or dropped/expired. That poses the lazy data 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[4]. That should reduce the associated memory and disc requirements for aggregation layers.
- the global view of cloudlets needs to be periodically presented for at least one of the hierarchically associated aggregation edge layers. For example, with the state marks, like "unknown/autonomous", "synchronizing", "connected", "failed/disconnected/fenced", if and only if it is confirmed as failed, or manually disconnected, or fenced automatically. That poses the aliveness of control/management plane principle.

[2]despite the cloudlets aliveness or failure conditions

[3]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 associated upper aggregation edge layer(s)

[4]or optionally, the queued operations may be distributed across nonshared queues

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):

- CRUD (Create, Read, Update and Delete) operations on cloud data can always be requested via API/CLI of local cloudlets or associated aggregation layers sitting on top of the replication hierarchy. The same queueing requirements apply as it is defined for the autonomy requirements.
- local CRUD operations for disconnected/autonomous cloudlets, if its control plane exists[5] and is not failed, can be processed without waiting for the upper aggregation layers to recover its control over the cloudlets.
- 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[6], or via its hierarchically associated aggregateion layers. Same queueing requirements apply.
- 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 queueing requirements).
- similarly, major versions of system components, like OpenStack or OpenShift/Kubernetes platforms, can be always scheduled for upgrades, using the same meanings as above.
- 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. Data Replication Consistency Requirements

The operational invariants dictate inevitable presense of shared state and sophisticated data replication mechanisms[7] among cloudlets.

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

- data replication conflicts can be resolved automatically, or by hand. The conflicts resolving strategies and rules should be customizable, like “last writer wins” or “return them all”. After the conflics resolved, the data may be considered causally related, that is by definition [1] of the data conflicts in eventually consistent systems.
- as far as the replication topology and queueing capabilities allow that, causally related data can be replicated across cloudlets.
- 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 hierarchically associated upper aggregation edge layers.

OpenStack and OpenShift/Kubernetes, have yet causally consistent data backends[8] supported. 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].

The weaker than causal+ and RTC *total available* [4] consistency models may be considered as an alternative. Transactional global databases [5], may technically support it[9]. A weaker consistency model provides a really poor alternative though as it brings increased implementaion complexity, like corner cases handling for either the storage replicas, or client sided, or both, associated with relaxed constraints. E.g. *monotonic atomic view* [4] does not impose any real-time constraints, while RTC does, which somewhat simplifies the end system design. Additionally, monotonic atomic view would require sophisticated handling of *fuzzy reads* [4], *phantoms* [4], discarded write-only transactions, empty state returned for any reads. All that makes that the strongest option for totally available data backends less preferable than causally consistent ones.

Kubernetes clusters state is backed with Etcd, which only supports the stronger than RTC consistency models.

D. Vision of a Unified Deployment/Cloud Data Storage Design

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

The vision of the unified architecture based on such an always available data storage dictates us to not consider different backends for cloud and deployment data. Although generic version control systems, like Git, might fit all cases for deployment data versioning, replicating and conflicts resolving, that would break the unified design approach for cloud data replication.

Client libraries implementing causally consistent data replication and customizable conflicts resolving rules may provide a unification layer for different underlying databases/KVS

[5] 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

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

[7] when we refer to just *data* or *state*, we do not differentiate either that is deployment or cloud data, which poses the unified approach principle

[8] that is, for cloud/deployment data only

[9] not Galera/MariaDB cluster though, as it has a strict quorum requierements for database writes

(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 global 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 tooling supports such backends.

Additionally, client libraries may replicate not only data but operations at an API level[10] as well, i.e. resolve conflicting operations on-fly, then apply the resulting causally related operations for its original targets, effectively acting as API-to-API retranslators for cloudlets replicating to each other 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 nonshared local cloudlets databases, or KVS, that provide the stronger consistency guarantees by the costs of reduced local availability[11]. 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 associated upper aggregation edge layers. Replicating the state changes via causally related operations and conflicts resolving via custom handlers is that COPS covers as well.

Global causally consistent databases [6] is an alternative approach to client libraries operating on top of cloudlets databases/KVS/API. The downside is such a database has to be supported as a control/management planes data backend for OpenStack/OpenShift/Kubernetes and/or any stateful cloud applications leveraging such a global database as a Replication-as-a-Service solution. And local cloudlets databases/KVS have to be switched to that global database.

Open questions:

- Does COPS retains operations causal+ related when executed over multiple datacenters failure events (or extended time of being network partitioned?)
- Is COPS applicable for two-way convergent systems, in terms of [2], for bidirectional causal+ replications? Given that [1] proves RTC provides the strongest causal consistency for one-way convergence only, and given that it provides stronger consistency than causal+, we can conclude that causal+ cannot provide the strongest causal consistency for bidirectional communication neither.
- 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 updatesender 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].
- 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? Global causally consistent databases may be a good choice for that. Alternatively, instead of bidirectionally replicating such data, inter-cloudlets API-to-API based synchronization mechanisms may become a solution.

TODO: find a use for RTC and [6] alternatives to form more options for vision. Finally, make preferences for causal databases vs KVS, if possible?

IV. Related work

TBD maybe?

V. Conclusion

TBD

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.

[10] 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

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