Kimmerlin, Mael; Plauth, Max; Heikkilä, Seppo; Niemi, Tapio

A Practical Evaluation of a Network Expansion Mechanism in an OpenStack Cloud Federation

Published in:
Proceedings of the 2017 IEEE 6th International Conference on Cloud Networking, CloudNet 2017

DOI:
10.1109/CloudNet.2017.8071540

Published: 19/10/2017

Document Version
Peer reviewed version

Please cite the original version:
Kimmerlin, M., Plauth, M., Heikkilä, S., & Niemi, T. (2017). A Practical Evaluation of a Network Expansion Mechanism in an OpenStack Cloud Federation. In *Proceedings of the 2017 IEEE 6th International Conference on Cloud Networking, CloudNet 2017* [17281412] (IEEE International Conference on Cloud Networking). IEEE. https://doi.org/10.1109/CloudNet.2017.8071540

A Practical Evaluation of a Network Expansion Mechanism in an OpenStack Cloud Federation

Maël Kimmerlin*, Max Plauth†, Seppo Heikkilä‡ and Tapio Niemi‡
*School of Electrical Engineering, Aalto University, Finland
†Hasso Plattner Institute, University of Potsdam, Germany
‡Helsinki Institute of Physics, CERN, Geneva, Switzerland

***Abstract*—The SSICLOPS consortium recently designed a transparent virtual network expansion mechanism for OpenStack. In this paper, we build up on this mechanism to propose features to improve the interconnection for inter-cloud federations. Based on distributed in-memory databases and *High Energy Physics* (HEP) workloads as two representative cloud computing workloads, we performed extensive performance evaluations to demonstrate that in a setup comprised of five sites across Europe, the performances of our interconnection agent are similar to the performances of the legacy VPNaaS feature provided by OpenStack. However, the interconnection agent is not restricted to the exemplary uses cases, as it is applicable to arbitrary workloads. This enables us to work in the direction of transparent inter-cloud live migration from a networking point of view.**

Keywords*—Cloud federation; OpenStack; Network Expansion; Interconnection;

I. Introduction

Cloud computing has gained a notable prevalence rate. More and more services are running in the cloud. In the meantime, the necessity to distribute applications across several clouds has increased tremendously. The main reason is that distributing an application over several clouds resolves issues such as single points of failure and varying latency between clients and a central server. Running an application in several cloud instances permits to be independent of a single cloud provider while bringing the service closer to the users by allowing them to use the most suitable service. However, achieving application distribution is a complex issue. In order to help solving this issue, cloud federations have started to develop.

Cloud federations allow users to use several cloud installations in a unified manner. They also permit to leverage the idle resources of other clouds in case of load bursts. Based on this, a topic that has drawn some attention lately is the cloud federation from a networking perspective. The predominating solution is a *Virtual Private Network* (VPN). However, the integration of VPN in OpenStack offers only a layer 3 interconnection. There is still a gap when it comes to layer-2 inter-cloud network expansion. In the SSICLOPS project, we proposed an interconnection service for OpenStack clouds to expand virtual networks to peer clouds at layer 2 [1], allowing transparent inter-cloud migrations for virtual machines, containers or applications.

This service integrates with Neutron and creates encrypted layer 2 tunnels between clouds, providing isolation and encryption for the virtual networks. The performance of this solution is degraded due to a high level of packet fragmentation. Additional features can improve the performances of the interconnection agent, such as a *Maximum Segment Size* (MSS) clamping service and a rate limiting service.

Here, we propose an extension of the interconnection agent to provide a VNF framework for services. We detail the implementation of the two services mentioned above, improving the resource utilization of the interconnection. We then extend the evaluation from a local testbed to a testbed distributed across Europe. We consider two use-cases that we deem representative of cloud workloads. The first use-case is an in-memory database replica distribution scenario for medium-sized analytic queries. The second is a data intensive *High Energy Physics* (HEP) job from CERN. We also integrate live migrations metrics and compare them within and between clouds. We have set up a cloud federation composed of five clouds that are geographically separated and interconnected over the Internet. Our extended solution has comparable performances to the legacy VPNaaS. Even though tested with only two use-cases, our approach is applicable to any workload running on the cloud.

In the remainder of this paper, Section II reviews the related work and Section III introduces the use-cases used for testing the performances of our interconnection agent. Section IV presents the extension of our agent while Section V presents the testbed setup, the test methodologies and the results.

II. Related work

The predominant solution for network federation in OpenStack is VPNaaS [2]. This service enables the users to create layer 3 VPNs to connect remote sites. However, this approach is not transparent for the users or the applications as the networks are independent. The aim of VPNaaS is to provide a secure remote connection, not a transparent inter-connection. The BEACON project pursues a very similar goal [3]. They designed an interconnection agent, that is however based on the Opendaylight controller, while our solution works with the default configuration of OpenStack. Furthermore, we provide an additional framework for interconnection services.

The OPNFV NetReady [4] project and the X-OS based on ONOS [5] are also related to our work as they are frameworks to integrate NFV in OpenStack. However, they offer the feature to the users while our purpose is to integrate the NFV in the features offered by the cloud providers. Moreover, none of them is related to cloud federation and allows for a simple feature adding possibility to connect the virtual networks of different clouds.

LISP[1], together with e.g. VXLAN, is one approach to extend networks between data centers. The main difference to our approach is that LISP requires a global mapping database where all virtual machines (VMs) are registered. Our approach uses MAC learning which does not require any extra configuration or modifications to OpenStack.

A project that already brings intercloud network federation and opportunity to run NFV for the users together is Contrail by Juniper [6]. An open-source fork is available. However, this project completely replaces the underlying networking mechanism in OpenStack, while our approach is based on the legacy system provided by OpenStack. Moreover, even though Contrail offers a framework for the users for NFV, it does not provide a framework for the cloud provider to offer services to its users.

Tricircle [7] is also a cloud federation project. However, the approach taken is quite different since it creates a hierarchical cloud structure. This structure relies on a central federated Keystone service to which all the clouds need to have access. The central OpenStack services then distribute the requests to the other clouds. However, this approach is not suitable in our case as our clouds are all independent from each others.

Nested virtualization is one approach for federating separate cloud infrastructures. With Kangaroo framework [8], VMs in separate clouds are used to run OpenStack and the associated VMs. These nested VMs are then connected with layer 2 overlay network between clouds. An advantage of the approach is the possibility to live migrate the nested VMs between clouds. We test live migration of Linux Containers, instead of nested VMs, between different clouds in the section V-B.

III. Use-cases presentation

We selected two use-cases that we consider as somehow representative of the broad range of workloads in clouds, Hyrise-R and High Energy Physics. However, the applicability of the interconnection agent is not restricted to these use cases.

A. In-Memory Database

Hyrise[2] is an open source in-memory research database which features a main delta architecture, dictionary encoding as well as an insert-only approach and thus shares certain characteristics with SAP HANA [9]. The major conceptual differences is that *Hyrise* is not limited to column-based storage, as it supports a flexible hybrid table layout [10]. A columnar arrangement is suited for attributes which are often accessed sequentially, e.g. via scans, joins, or aggregations. Attributes accessed in OLTP-style queries, e.g., projections of few tuples, can be stored in a row-wise manner.

Following the scale-out paradigm, Hyrise-R extends the research database to a cloud-based in-memory prototype employing lazy master replication [11]. Hence, a functional setup consists of a master node, an arbitrary number of replica nodes and one dispatcher node [12]. The master is the only node accepting write operations, propagating them to replicas. The dispatcher merely acts as a load-balancer that redirects write operations such as *OLTP* (Online Transaction Processing) queries to the master node, whereas read operations such as *OLAP* (Online Analytical processing) queries can be spread across to all replicated nodes and therefore can increase the throughput of the entire system.

[1]RFC 6830

[2]https://github.com/hyrise

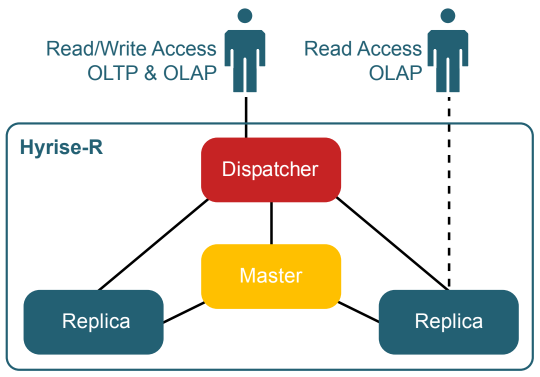

Figure 1: Replication architecture of the Hyrise in-memory research database.

In this work, we consider a database setup that employs replication for the main purpose of increasing availability. Given this scenario, we want to evaluate the performance impact of employing inter-cloud replication compared to local intra-cloud replication. An additional motivation for employing inter-cloud replication is that replicas hosted in different locations can be leveraged to provide reduced response times for read queries issued by clients situated in the close proximity of the respective replicas.

B. Data-intensive jobs

High Energy Physics (HEP) studies the nature of elementary particles by analyzing high-energy particle collisions. The data of high-energy particle collisions from the CERN Large Hadron Collider (LHC) experiments are stored in more than 150 separately managed data centers spread around the world. Networking solutions play an important role for determining possible network topologies and minimizing the delays for the analysis jobs.

By running this use-case, we aim to compare different network connections and examine how they affect different HEP workloads. First, we examine the performance of data intensive HEP job that selects relevant HEP data and transforms it into a more compact form [13]. Second, we examine the performance of live migration of Linux containers used for HEP data processing accross clouds.

IV. Expansion agent

We have previously created a cloud interconnection solution for OpenStack. Considering several clouds that are part of the cloud federation, each cloud would run an interconnection setup. The setup inside each cloud consists of several local agents, each of them running on an OpenStack node, and a central server. The clouds are interconnected using encrypted layer 2 GRE, VXLAN or Geneve tunnels. Those tunnels are set up upon agreement between the clouds individually in a peer-to-peer mode. The tunnels are then available for the cloud users to expand their network transparently to other clouds. We have now extended this service.

In order to improve the scalability, we modified the structure of the agent to support interconnection from multiple OpenStack nodes. Local agents register themselves to the central server upon start and create the base infrastructure on the node. The server and the agents then interact to create the tunnels and expand virtual networks. The server processes only the setup operations, through a REST API. Hence, the number of nodes is not a determining factor for the scalability, but rather how frequently inter-cloud expansion operations are taking place. In order to scale up for larger clouds such as public clouds, modifications of the database interactions would be required to allow running several central servers in parallel to scale up.

In a typical deployment with at least three OpenStack controllers, the networking elements for a specific virtual network will be provisioned on a subset of nodes. However, the endpoints of the tunnels between federated clouds might be located on different nodes. Hence a mechanism is needed to forward traffic from the nodes where the virtual network is provisioned to the nodes where the tunnel endpoints are located. This is achieved with a full mesh topology that enables us to decouple the tunnel endpoints that require external connectivity from the virtual networks bridging nodes. The first ones do not have to be related to OpenStack deployment, while the second ones need to be connected to the controller or compute nodes, either with VLAN or VXLAN depending on the technology used in the deployment.

We integrated our solution with OpenStack by connecting it to the internal bridge of the node. The architecture is displayed in Figure 2. Based on this architecture, we provide several features and services to improve the connection between the clouds. We integrated a VNF-based framework to include services operating as virtual network functions (VNF). We based our implementation on *Network Function Virtualization* (NFV) to deploy easily the different services needed to forward the traffic to other clouds. The VNF framework is bi-directional. Based on the VLAN tag used by OpenStack to internally identify the network, we redirect the traffic coming from the cloud and going through the expansion mechanism to a chain of functions. The traffic coming from the other clouds is similarly redirected to the correct chain of functions based on the tunnel key identifier. For each of the virtual networks, a set of VNF is instantiated. The two main functions currently implemented are: MSS clamping to avoid fragmentation as much as possible; and Rate limiting. However, more functions could be added, such as a firewall, a proxy or a cache.

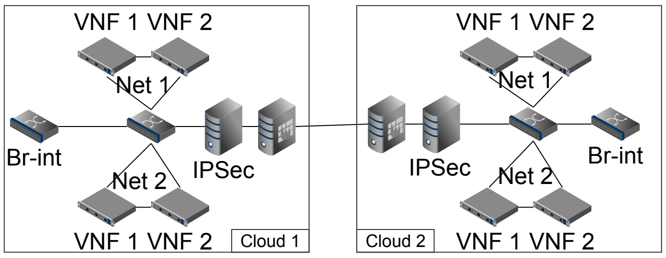

Figure 2: Overview of the VNF framework

A. MSS clamping VNF

Issue description: A well known issue when setting up layer 2 connectivity is related to the maximum transmission unit (MTU). Due to tunneling and encryption, the payload size in the tunnels between clouds is reduced compared to the capacity of the links between VMs. For example, if two clouds are connected over the Internet, it is highly probable that the path MTU amounts to 1500 bytes. The maximum MTU that would then fit in the tunnel using VXLAN and IPSec in transport mode would be 1416 bytes, while the MTU of a virtual network is usually 1450 bytes. The packets would then get dropped.

Mechanisms such as Path MTU Discovery (PMTUD) are inefficient in this case, because they rely on answers from the routers. They proceed by sending full packets (padded to fill the entire payload allowed by the sending interface MTU) with the bit ”Don't fragment” set in the IP header. This triggers ICMP messages ”Fragmentation-needed” on the routers that have a smaller MTU on the following hop interface. Thus the sender gets information on the path and is able to reduce the size of the payload in the packets to avoid issues. However, since we are using switching at layer 2, the equipment are not able to process IP packets and silently discard the oversized packets. This is called a PMTUD blackhole, as the packets seem to simply disappear.

A simple solution would be to reduce the whole network MTU. However, this might be problematic with OpenStack as the MTU is fixed at the time of network creation and cannot be changed afterwards. Moreover, most of the cloud providers (either public or private) use bigger MTUs in their internal networks. Thus the MTU of private tenant networks are usually 1500 bytes or could be much bigger than the usual MTU (up to 8948 bytes when using VXLAN in OpenStack). In that case, the MTU reduction would be of 83%. This solution is not acceptable.

Mere fragmentation also presents issues. If we consider the case where the VMs try to send packets of 1500 bytes in a VLAN-based setup, the packet will be fragmented to fit the VXLAN header, and the first segment of each fragmented packet will be fragmented again to fit the IPSec header. We consider that the networks expanded have a consistent MTU over the different clouds.

We designed a solution to tackle those issues by processing the packets before fragmentation. We want to enforce MSS-clamping for TCP to avoid fragmentation as much as possible, for all networks independently, depending on the destination cloud, and fragment all IP traffic correctly.

Design: A Linux bridge combined with Iptables and Ebtables rules is used to pre-process the packets going through the tunnel. This solution is very close to the implementation of the firewall in OpenStack. Since Iptables cannot be applied on an interface provided by Open vSwitch, like the internal *br-int*, OpenStack uses an intermediary Linux bridge to apply the Iptables rules before the traffic reaches the VMs. In our case, we instructed the Open vSwitch switch to divert the traffic to an intermediary Linux bridge. We then set up Iptables and Ebtables, so that the MSS of all the TCP traffic is clamped to fit in the tunnel, and all remaining oversized unicast, broadcast and multicast IP traffic is fragmented. If there are some non-IP

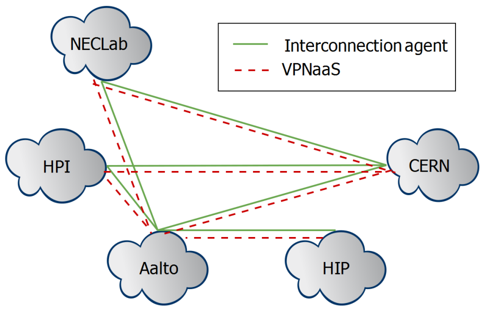

Figure 3: Topology of our federated testbed.

packets of bigger size than the MTU, those are also fragmented. The fragmentation happens on the tunnel outer IP header, meaning that the original packet is not modified. This enables us to fragment non-IP packets.

B. Rate limiting VNF

This design allows us to isolate the tenants. However the resource sharing must also be considered. The VNFs are set up per tenant, but the traffic from all tenants use the same path and compete for the same bandwidth. Several resources might be subject to exhaustion, such as bandwidth, in case of a limited uplink bandwidth, or CPU resources, for encryption and virtual switching. In both cases, the traffic from the users can be shaped to offer each of them a quality of service related to their agreement with the cloud providers. This can enforce both limitation of traffic and minimum service, depending on the type of Service Level Agreement (SLA) the cloud providers offer.

We have designed and implemented a VNF for traffic shaping. The VNF is connected transparently on the network. It takes the aggregated traffic and shapes it in both directions. The isolation is based on VLANs. The VNF can be used by the cloud provider to limit the bandwidth of its users individually depending on the contract between the user and the cloud provider, or to ensure a minimum available bandwidth to certain users by reserving part of the bandwidth and allowing users to share the remaining bandwidth depending on criteria set in the agreement. The VNF also offers the end users the possibility to shape their traffic within the bandwidth allocated by the provider and add some policies (such as QoS, different queuing disciplines).

Several other services could be provided using this VNF framework. For instance, in a future work we will detail the integration of an MPTCP proxy.

V. Testbed Setup and Test Methodology

We deployed the extended solution in a set of five clouds with different peering agreements. The map of the testbed is presented in Figure 3. The five cloud providers are Aalto University (Finland), CERN (Switzerland), Helsinki Institute of Physics (Finland), Hasso Plattner Institute (Germany) and NECLab (Germany). All the clouds are using OpenStack (two with Liberty, one with Mitaka and two with Newton). We performed a complete set of bandwidth and latency

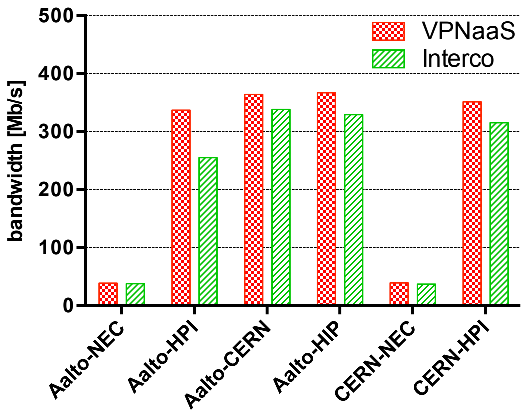

Figure 4: Bandwidth measurements.

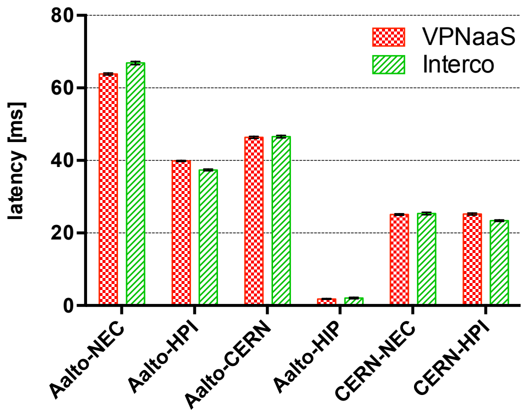

Figure 5: Latency measurements.

measurements to evaluate the link between the different clouds. The available bandwidth between the clouds is displayed in Figure 4 while the latency is displayed in Figure 5. The bandwidth is slightly reduced when using the interconnection agent compared to the VPNaaS solution due to the overhead of layer 2 encapsulation. The latency results for HIP are different since Aalto and HIP are connected by a LAN (same geographical location).

We used this setup to test the performances of our extended solution, using the two use-cases introduced earlier, Hyrise-R and HEP jobs.

A. Distributed in-memory database

1) Test Methodology: The test setup is constrained to the bare minimum necessary to test replicated performance in inter-cloud deployments. The test setup is comprised of one master node, one replica node, and one dispatcher node. All nodes are hosted in individual virtual machines. Furthermore, a dedicated virtual machine is used to execute the OLAP query benchmark. All virtual machines are running Ubuntu Linux 14.04.5 LTS and are equipped with 4 vCPUs and 4GB RAM.

The primary goal of this evaluation is to compare the performance of our custom layer 2 interconnection agent to VPNaaS. Here, the replica node is hosted at the Aalto University (Finland) site and both interconnection techniques are employed to establish a connection to the HPI (Germany) site, where all other VMs reside. To quantify the performance impact of distributing *Hyrise-R* instances across multiple cloud installations, we also measure the baseline performance of a local setup, where all virtual machines are hosted within the HPI site.

The application performance is obtained by measuring query throughput for OLAP queries against a TPC-C [14] conformant data warehouse containing $300,000$ line items. In order to retrieve a sufficiently meaningful dataset, each condition was benchmarked 100 times. Furthermore, each benchmark was preceded by a warm-up procedure. For a statistically meaningful evaluation of the collected data, an ANOVA test and a post-hoc comparison using the Tukey method were applied.

2) Results: As illustrated in Figure 6, local replication within one data-center yielded statistically significant gains in performance compared to inter-cloud replication ($p < 0.0001$). However, no statistically significant performance differences were detected between the layer 3 based VPNaaS interconnection technique and the custom layer 2 interconnection agent.

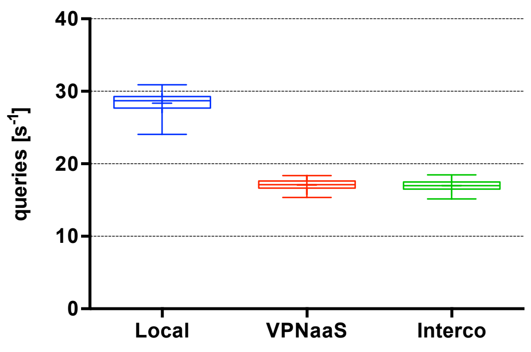

Figure 6: OLAP query throughput in a replicated setup.

Considering the characteristics of long-running OLAP queries, *Hyrise-R* should not be sensitive to network latency, especially not for reasonably low latencies as it occurs on the Aalto-HPI link (roughly 40ms, see Figure 5). Hence, it was surprising to see the significantly decreased performance of the inter-cloud setups compared to the local installation. Further investigations revealed that the decreased performance in the inter-cloud scenarios is caused by flaws of the *Hyrise-R* dispatcher. In its current state, the dispatcher is incapable of distributing requests evenly in cases where the processing speed of the involved nodes is heterogeneous. Even though the same VM dimensions were utilized in all sites, the underlying hardware offered different levels of performance. As a result, we always ended up with one *Hyrise* node being congested, whereas the other node remained partially idle.

With no statistically significant differences between layer 3 based VPNaaS and the layer 2 based interco agent, we demonstrated that the increased overhead of the latter does not hurt application performance. At the same time, layer 2 connectivity across inter-cloud setups introduces opportunities for entirely novel features such as inter-cloud live-migration of replicas based on the current demands and capacities in the respective sites. This demonstrates that our concept and implementation are suitable for this kind of workloads.

B. High Energy Physics workload

1) Test Methodology: The HEP workload tests are performed between two testbed sites located at Aalto University (Finland) and CERN (Switzerland). Both sites have three VMs with Scientific Linux 6.7. The CERN site VMs (2vCPU, 4GB RAM) run three HEP jobs in parallel and Aalto site VMs (1vCPU, 1GB RAM) host a HEP data file with an xRootD server. In this test, we use a 1.4GB HEP ROOT file containing 20,000 events and a HEP job transforming these real physics events into a more compact form. This means that the workload is data intensive and network performance is usually the bottleneck. The remote HEP data access is tested with three different network connections: a direct public IP address, the OpenStack layer 3 VPNaaS, and the layer 2 interconnect agent presented in this paper. In addition, a HEP job with local data is ran to determine the performance without network limitations.

Two metrics are used to evaluate performance of HEP jobs: the job duration and energy usage. The energy usage is measured using an external power meter. The local data and the three network connections are tested one after the other, and this is repeated 30 times. A one-way ANOVA test and a post-hoc comparison using the Tukey method are used to determine if the results have statistically significant difference.

We also test the possibility of transparent live migration between testbed sites. Since the virtual networks are expanded across clouds, it is now possible to migrate a container, keeping the same IP address. The live migration of Linux containers used for HEP analysis is also done between CERN and Aalto University sites. Both sites have identical Ubuntu 16.04 VMs (1vCPU, 2GB RAM) that are running CentOS 7.3 Linux container. The container is used to run a CPU intensive HEP benchmark job (ParFullCMS). The LXC version is 2.12 and CRIU version is 2.0. For comparison, the same migration test is also performed between two compute nodes within the CERN testbed.

Four metrics are used to evaluate Linux container migrations: migration time, amount of data transferred, container downtime, and HEP job run time increase due to migration. The HEP job run time increase is the difference between the job runtime with migration and without migration. The container migration is repeated ten times both between testbed sites and OpenStack compute nodes. The Student's t-test is used to determine if the results have statistically significant difference.

2) Results: Figure 7 shows job runtimes when data location is varied. Each of the three parallel jobs accessed around 410MB of HEP data. The job with local data was the fastest ($p < 0.001$). There was no statistically significant difference between OpenStack VPNaaS and layer 2 interconnect. The energy usage results were almost identical to processing time. This was expected because processing time is known to directly increase the energy usage [13].

The HEP container live migration results are shown in Table I. Live container migrations worked correctly over the layer 2 interconnect, but there was statistically significant difference ($p < 0.01$) with all the four metrics, except with

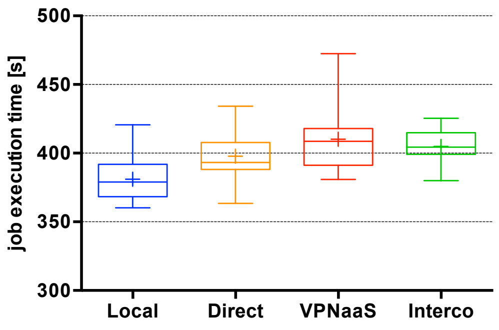

Figure 7: HEP job execution time.

the amount of data transferred, when compared to migration between compute nodes. The migrations took around 12% longer and there was also 16% longer downtime. The OpenStack

	Between nodes	Between sites
Migration time [s]	123.2 ± 2.4	137.6 ± 2.5
Downtime [s]	15.4 ± 0.7	17.8 ± 1.3
Job runtime increase [s]	81.7 ± 8.4	70.2 ± 7.4
Transferred data [MB]	1070.2 ± 6.1	1067.3 ± 5.9

Table I: HEP container live migration performances

VPNaaS and layer 2 interconnect raw throughput and latency were similar, as shown in Figure 4 and Figure 5. Thus, it was quite expected that there is no significant difference in the HEP job performance. In particular, this means that the layer 2 interconnect does not have any disadvantageous behaviour or features that would affect HEP job processing.

The layer 2 interconnect worked correctly also with container migrations. Increased migration time and container downtime can be explained with differences in the network throughput and latency. The migration duration between sites increased in average by 14.4 seconds, when compared to migration between computing nodes. The HEP job execution times instead increased by less than two minutes. Both of these are acceptable durations since live migrations are not expected to happen very often and a typical HEP job duration is several hours. The advantage of layer 2 interconnect with the live Linux container migration is that we were able to use the same IP address for the container even after migrating to another site. This means that the migration can be in practice transparent for the end user.

VI. Conclusion

We have extended the interconnection agent and proposed and extensible framework for VNF. We introduced the framework and detailed the implementation of two example functions. We then presented a testbed setup comprised of five parties spread across Europe. A performance evaluation was conducted to validate the architecture using two representative workloads, a large volume data transfer scenario and a high-throughput analytics database scenario. In addition to these use-cases, we analyzed the performance of a container live migration scenario. We can conclude that our solution does not have a negative impact on performances or create any drawbacks, compared to the legacy solution, while it allows to add many new features from a networking point of view, such as inter-cloud transparent live-migration. Furthermore, the interconnection agent is not restricted to the evaluated use cases, as it is applicable to arbitrary use cases.

In the future, we will focus on the transparent live migration. While migrated VMs need to keep using the same gateway in order to keep the on-going connections and the availability of floating IP addresses, enabling migrated VMs to use the local gateway for new connections would reduce latency and bandwidth consumption.

Acknowledgement & Disclaimer

We would like to express our most sincere gratitude to Jesus Llorente Santos for his great help during the design of our solution. This paper has received funding from the European Union's Horizon 2020 research and innovation programme 2014-2018 under grant agreement No. 644866. This paper reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

References

[1] M. Kimmerlin, P. Hasselmeyer, S. Heikkilä, M. Plauth, P. Parol, and P. Sarolahti, "Network Expansion in OpenStack Cloud Federations," to appear in Proceedings of 2017 European Conference on Networks and Communications (EuCNC), Jun. 2017.

[2] OpenStack Foundation. (2017, Jan.) Virtual Private Network as a Service. [Online]. Available: https://wiki.openstack.org/wiki/Neutron/VPNaaS

[3] BEACON consortium, "BEACON Deliverable 3.1," Jan. 2017. [Online]. Available: http://www.beacon-project.eu/s/D31-Scientific-Report-WP3-a.pdf

[4] OPNFV Community. (2017, Jan.) OPNFV NetReady wiki. [Online]. Available: https://wiki.opnfv.org/display/netready/NetReady

[5] ONOS Community. (2017, Jan.) XOS: Service orchestration for CORD. [Online]. Available: http://onosproject.org/wp-content/uploads/2015/06/Technical-Whitepaper-XOS.pdf

[6] OpenContrail. (2017, Jan.) OpenContrail webpage. [Online]. Available: http://www.opencontrail.org/

[7] OpenStack Foundation. (2017, Jan.) Tricircle. [Online]. Available: https://wiki.openstack.org/wiki/Tricircle

[8] K. Razavi, A. Ion, G. Tato, K. Jeong, R. Figueiredo, G. Pierre, and T. Kielmann, "Kangaroo: A tenant-centric software-defined cloud infrastructure," in *2015 IEEE Intl. Conference on Cloud Engineering*. IEEE, 2015, pp. 106–115.

[9] H. Plattner, "A Common Database Approach for OLTP and OLAP Using an In-memory Column Database," in *Proceedings of the 2009 ACM SIGMOD Intl. Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 1–2.

[10] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, "HYRISE: A Main Memory Hybrid Storage Engine," *Proc. VLDB Endow.*, vol. 4, no. 2, pp. 105–116, Nov. 2010.

[11] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The Dangers of Replication and a Solution," in *Proceedings of the 1996 ACM SIGMOD Intl. Conference on Management of Data*, ser. SIGMOD '96, 1996.

[12] D. Schwalb, J. Kossmann, M. Faust, S. Klauck, M. Uflacker, and H. Plattner, "Hyrise-R: Scale-out and Hot-Standby Through Lazy Master Replication for Enterprise Applications," in *Proceedings of the 3rd VLDB Workshop on In-Memory Data Mangement and Analytics*, 2015.

[13] J. Kommeri, S. S. Heikkilä, A. Vartiainen, and T. Niemi, "The effect of networking performance on high energy physics computing," in *Proceedings of the 6th Intl. Symposium on Business Modeling and Software Design, Special Session on Green IT Solutions (BMSD-ICGREEN)*, 2016.

[14] Transaction Processing Performance Council, "TPC Benchmark C Standard Specification Version 5.11," Feb. 2010.