Spreading the heat: Multi-cloud controller for failover and cross-site offloading

Simon Kollberg, Ewnetu Bayuh Lakew, Petter Svärd, Erik Elmroth, Johan Tordsson

Department of Computing Science, Umeå University, Sweden
Email: $(dv11skg, ewnetu, petters, elmroth, tordsson)@cs.umu.se$

Abstract. Despite the ubiquitous adoption of cloud computing and a very rich set of services offered by cloud providers, current systems lack efficient and flexible mechanisms to collaborate among multiple cloud sites. In order to guarantee resource availability during peaks in demand and to fulfill service level objectives, cloud service providers cap resource allocations and as a consequence, face severe underutilization during non-peak periods. In addition, application owners are forced to make independent contracts to deploy their application at different sites. To illustrate how these shortcomings can be overcome, we present a lightweight cross-site offloader for OpenStack. Our controller utilizes templates and site weights to enable offloading of virtual machines between geographically disperse sites. We present and implement a proposed architecture and demonstrate its feasibility in both a typical cross-site offloading, as well as a failover scenario.

1 Introduction and Background

Cloud computing has become a dominant computing model enabling cost-effective and flexible access to a wide pool of resources, and many classes of applications, such as web services, big data analytics, distributed applications for social networking, storage and many other applications, have been smoothly integrated into cloud services. However, to accommodate application demands and minimize the risk of losing the constantly increasing flow of customers, Cloud Service Providers, CSPs, have been relying on the continuous expansion of their IT facilities while employing simplistic policies to manage resources [15]. Numerous studies have shown that CSPs overprovision their datacenter and fail to utilize the available resources efficiently, with their utilization ranging from 10% to 50% [19, 3, 6, 16, 7]. This means that, in order to guarantee availability during short time frames of peak demands and adhere to SLAs, CSPs overprovision resources resulting in *underutilization* of resources during non-peak periods [19].

Despite cloud computing's ability to provide flexibility and elasticity in resource provisioning, this so far holds true only within a single data center or administrative zone (site). The standard approach is to horizontal scale where additional Virtual Machines (VMs) are allocated to satisfy increased demands and is limited to a single administrative site. A better approach to utilize the cloud efficiently and flexibly, is that different cloud sites collaborate and interact in an autonomic fashion based on changes in the infrastructure or customers requirements. This collaborative approach can lead to improved resource utilization, i.e. sites can offload applications to another site during peak periods instead of keeping its own resources for such events, and service locality. The locality aspect means that applications can be placed or moved with regards to the locality of its traffic, or based on quality/latency requirements such as regulatory constraints, cost optimization, service continuity and geo-redundancy. Moreover, application owners might prefer to deploy their applications at different sites to serve geographically diverse end-users as well as for fault tolerance reason.

Collaboration or federation [4, 23, 17] between cloud sites can be beneficial both to the cloud provider and the service owners. For cloud providers, it offers opportunities to reduce total cost of ownership (TCO) through effective resource utilization and provide better maintenance. CSPs can efficiently utilize their resources and reduce resource overprovisioning for temporary peaks by offloading some of the workloads to collaborating sites during such events. However, realizing multi-cloud collaboration can be very challenging as it typically requires changes to the core part of the cloud infrastructure management system in each site, and in order to make decisions the state of each site should be known in advance. For service owners, it offers the opportunity to deploy their service in different clouds which can help them service locality and fail-over mechanisms. For example, failure of one or more cloud sites may not affect their service as new instances can be automatically spawned in a different cloud site.

In this contribution, we design and implement a proof of concept solution for capacity offloading and fail-over mechanisms across multiple sites. The multi-cloud controller has two main components: 1) An overload controller, including an overload agent, that detects overload across nodes within a site, mitigates overloaded nodes by load-balancing the load across nodes within the site and delegates to the second component if it cannot handle the load on the local cloud and 2) A HeatSpreader which performs offloading of VMs from the overloaded cloud to cloud sites with less load, detects cloud site failure and spawns new instances of inaccessible VMs in different cloud sites. Our solution works in conjunction with OpenStack [12] and OpenStack Heat [11], transparent to the management system where a single configuration is made to manage offloading of services across multiple sites. To realize efficient and flexible resource management locally and between geographically distributed sites, the proposed solution monitors the data center's resource utilization and application footprints in a non-intrusive fashion through OpenStack Heat and decides which application to offload to another site during peak periods and spawns new instances in a different site during site outage. The concept is then evaluated in two typical usage scenarios.

2 Background and Motivation

As described in the previous section, collaboration between multiple cloud sites helps to utilize the existing distribution of a cloud configuration to further balance loads between collaborating geographically distributed cloud sites. Using this model, CSPs can resort to much more cost efficient capacity planning, utilizing support from collaborating cloud sites to accommodate the demands during peak periods. Such collaborations could therefore be used both between separate public cloud vendors, and between an enterprise private cloud and their public cloud partners. Application owners can also use such configurations to dynamically adjust the number of VMs at each site based on end-users demand.

Moreover, cloud maintenance is currently one of the major challenges for CSPs. Supporting mass migration, for example, is currently a complex task that usually requires manual intervention. It also relies on having spare equipment for migrating workloads away from the system targeted for maintenance. Upgrading between different versions, for example in OpenStack, may require that the whole platform is brought down, the components upgraded and the cloud brought back online again. To allow for continuation of services, this usually requires instances be offloaded to a standby system, while the upgrade is carried out. While upgrading a single machine is manageable, upgrading multiple machines becomes more complex and resource demanding. Collaboration between multiple sites can ease the challenge of reserving extra resources for maintenance purpose.

From the service owner perspective, having multi-cloud support what is often attractive in order to spread services across multiple clouds to increase fault-tolerance and deploy services closer to end-users. By using only a single region or cloud provider, fault tolerance is limited by the availability of that provider. Besides, end-users will experience varying performance depending on their location with respect to the region where the service is deployed. Having a multi-cloud deployment allows for more graceful recovery of the loss of a region or entire provider as well as providing relatively better performance to end-users. Another use-case for offloading is vertical scaling. When using disaggregated hardware systems [24], VMs can be larger than the size of a physical node, in which case they need to run on a fat node [8]. If space is limited, smaller VMs can then be offloaded to non-disaggregated nodes to allow the large VM to grow.

However, despite these benefits, cross-site offloading and failover is still not widely used. One reason might be that it often requires installation and maintenance of complex software stacks, that must be compatible between sites. We argue that there is a case for a simpler solution.

3 Proposed architecture

In order for CSPs to collaborate transparently, they need to run compatible software stacks or define communication protocols. OpenStack, is a widely used cloud operating system, that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed and provisioned through APIs with common authentication mechanisms. In addition to Infrastructure-as-a-service functionality, OpenStack also provides fault- and service management, and orchestration [11]. Our solution is designed to be used in conjunction with OpenStack, without requiring any modifications to OpenStack components.

It is important to continuously observe the state of the infrastructure and proactively make decisions at runtime in order to provide performance guarantees in an environment where frequent and unpredictable changes are the norm. Our proposed Multicloud controller is a system that continuously monitors the state of the system and tries to reduce performance degradation and prevent loss of virtual resources due to overloaded compute nodes or cloud outages through collaboration with multiple cloud sites. It is

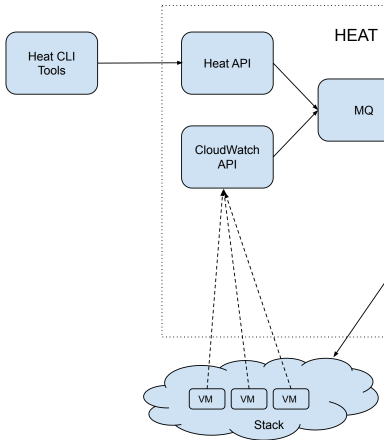

Fig. 1. Heat workflow

composed of three different components, the HeatSpreader, the overload controller and the overload agent. All components are implemented in Python and the source is available on GitLab on request.

The HeatSpreader component keeps track of and manages OpenStack Heat (see internal workflow in Fig. 1) stacks in multiple clouds. Specifically, it does this by continuously monitoring and driving the amount of virtual resources in each cloud towards its internal desired state. The HeatSpreader itself does not update its desired state, rather it exposes an API which external components can utilize. However, in the case of a cloud outage the HeatSpreader performs fail over on any lost virtual resources to configured clouds which are still available.

The overload controller and the overload agent works in conjunction. The overload agent sits on each compute node and monitors the load. Once the load on a a host exceeds a specified threshold the agents notifies the overload controller, which runs external to the stack with one instance per site. The overload controller then tries to rectify the situation by subsequently spreading the load between nodes in the same site. In case of an event when the entire cloud site is overloaded, the overload controller updates the the desired state in the HeatSpreader. The HeatSpreader then takes the responsibility of reducing the number of virtual resources in the overloaded site, and increasing them in the less overloaded cloud site(s). Fig. 2 shows an overview of how the multicloud controller interacts with multiple sites. The overload controller and the overload agent communicate via a messaging system, RabbitMQ [21], while the overload controller and the HeatSpreader communicate via a REST [10] interface.

3.1 The HeatSpreader

The HeatSpreader, which is the core component of our multicloud controller, is implemented as a standalone service. It is connected to multiple OpenStack deployment sites simultaneously and scales the number of VMs in each site through the OpenStack

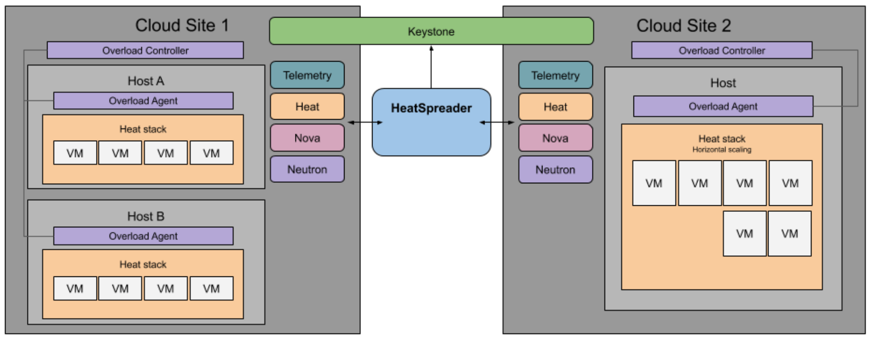

Fig. 2. MultiCloud controller overview

Heat API. The user configures it with the OpenStack Heat credentials for each site it should control. The HeatSpreader is configured using its REST API endpoint which it uses to reconfigure the site scaling weights. Whenever the HeatSpreader service sees a change in the site weights, it will scale-up or scale-down the number of VMs in each Heat stack. If the HeatSpreader service is unable to operate on a specific cloud site, for example in the case of a site outage, the weight will be re-distributed on the other available sites. The HeatSpreader itself exposes an interface which can be used within a site to offload less prioritized VM instances to a different site when it no longer finds a local solution. This interface is used by the overload controller when the controller detects site overload. In this way, the overload controller manages the cloud within a single site while the HeatSpreader is responsible for redistributing load across multiple cloud sites. A high level description of the algorithm is shown in Algorithm 1, and reasons for adjusting the cross-site balance include:

- Site imbalance: Some production compute nodes are heavily utilized, while others are not (e.g., the difference between the node with the heaviest workload compared to the node with the lightest workload exceeds a threshold).
- Site overload: Site resources are over-utilized resulting in application demand not being met. To circumvent such situations some VMs are offloaded to different sister sites.
- Notification from remote sites: One or more of the sibling remote sites report over-utilization.

Note that in order to be a candidate for offloading, the application must be stateless as it is too expensive to undertake state migration across geographically distributed sites. This approach can still be beneficial to stateful applications though, as stateless applications can be offloaded to allow the stateful ones to scale.

Algorithm 1 High level description for Multicloud-Controller Algorithm

```
1: repeat
2:     cloudList= getCollaboratingClouds()
for each: $c \in \rfloor loudList$
3:     alerts = c.getAlert()
4:     if c.hasFailed()=True then
5:         SELECT candidate cloud sites
6:         OFFLOAD VMs in c to the candidate cloud sites
7:     end if
8:     if c.isOverLoaded()=True then
9:         SELECT candidate applications to offload in cloud c
10:        SELECT candidate cloud sites
11:        OFFLOAD candidate application to candidate cloud sites
12:    end if
13:
```

4 Experimental Evaluation

4.1 Experimental Setup

To evaluate the feasibility of our design, we performed two experiments, testing both the cross-site offloading/overload mitigation, and the failover scenario. The experimental setup consists of two cloud sites, running our OpenStack and our overload controller and the HeatSpreader running external to both clouds. Cloud site 1 is composed of three PMs (Nodes A, B and C) and cloud site 2 of 1 PM.

Overload Mitigation Our evaluation demonstrates how our solution performs load-balancing within a single site and offloading during overload situations. The multicloud controller which is composed of the overload controller and the HeatSpreader, manages the cloud within a single site and across multiple sites. The overload controller performs load-balancing within a single site and delegates responsibility to the HeatSpreader when policies cannot be met withing a site.

Fail-over In this scenario, the ability of the HeatSpreader to detect a failure in one of the cloud sites is tested. To emulate an outage, the Heat API service is stopped on cloud site 1. When the Heat service on one of the clouds is unreachable, the HeatSpreader will assume it is down, and automatically adjusts the site weights to bring up VMs on the offload site to cover for those that are no longer reachable. The experimental hardware setup is the same as in the overload scenario, and to simulate an outage, the Heat API service is killed at cloud site 1. In order not to completely overload the small offload site, the number of instances was decreased from the overload test.

4.2 Results and Discussion

This section presents experimental results for the detection and mitigation of infrastructure overload as the fail-over scenario under multi-cloud environments.

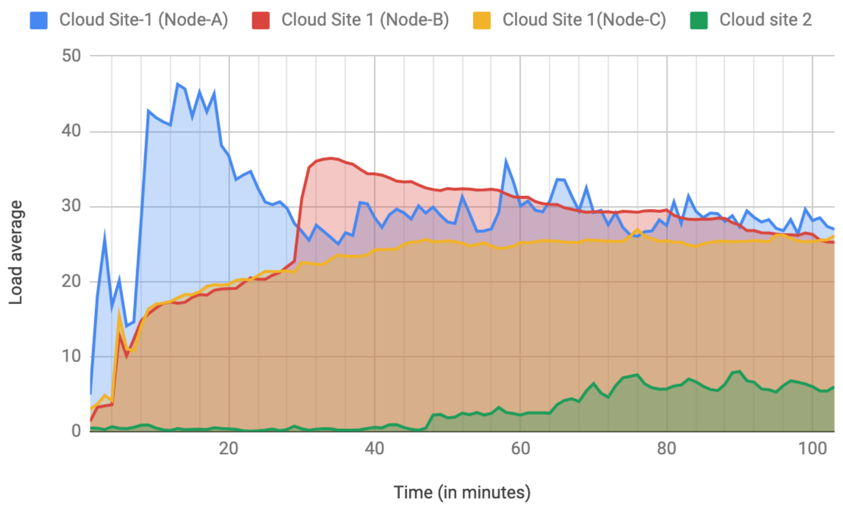

Fig. 3. Load average on the sites, overload scenario

Overload Mitigation The evaluation in this section shows how the controller mitigates overload locally and offloads candidate VMs to a different site when local decision cannot meet policy requirements. In Fig. 3, the load average [13] is plotted for all of the nodes in the cluster. The HeatSpreader is configured with an objective to maintain the overall utilization around 80% which is equivalent to a load average of approximately 26 for cloud-site 1.

From the figure, we can observe a load surge from minutes 8 to around 18 on Node-A. Following the load surge, our overload controller tries to perform load-balancing within cloud-site 1 by migrating VMs from Node A to Nodes B and C (reflected between minutes 18 to around 48). This is manifested by the load increase on Nodes B and C and load decrease from Node A during the interval.

The load average of cloud site 1 is still above the threshold so the HeatSpreader starts offloading to cloud site 2, at around T=40 to T=80. As cloud site 2 is much smaller and less powerful than cloud site 1, it reaches maximum capacity at around T=70 which means that the overall load average for cloud site 1 is still quite high, however it is now stabilized under the threshold. Using a offload site of similar size would as the main site would be more feasible, but the experiment indicates that the mechanism works.

Fail-over Fig. 4 shows the results of the fail-over test. As in the overload scenario, the load average for the nodes are plotted against time. Around T=140, the Heat API service was killed on cloud site 1, causing the measured load average of those nodes drop to zero. When the HeatSpreader's health check mechanism detects that cloud site 1 is unreachable, and flags it as down. It then starts failing over all instances from cloud site 1 to cloud site 2 by adjusting the site weights. This causes the load on cloud site 2 to increase sharply as seen from T=200.

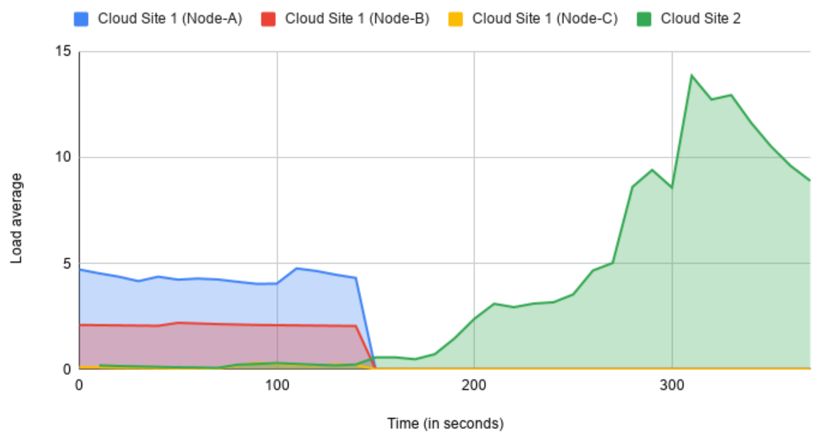

Fig. 4. Load average on the sites, failover scenario

5 Related Work

Cloud federation has been discussed in many contributions since the rise of cloud computing. For example by Ferrer et al [9] who propose a federation software stack for cloud providers, Li et al. [18] who investigate the concept of a cloud federator to act as a middle man between cloud sites and clients, Horn et al. [14] who present an autonomous middleware approach, and Moreno et al. [20] who utilize the concept of virtual network to enable federation. The idea of a Hybrid cloud, i.e. pairing a smaller private cloud, with a larger public cloud for offloading, together with software to facilitate its use has also been investigated [2, 1, 22]. Fog and Edge computing uses similar ideas to lower the network load by running often used, latency sensitive, applications close to the users while offloading heavier computations to a core datacenter [5, 25]. The main benefit of our solution is that it is lightweight, requiring no changes to openstack, or the provisioning of complex software stacks. The overload controller runs as a separate process and the HeatSpreader runs externally from the cloud. Together they provide an easy way to enable two of the most attractive features of cloud federation, and if they should fail, the cloud sites continues to operate as normal.

6 Conclusions and Future Work

In this short contribution, we designed, implemented and evaluated a lightweight multicloud controller consisting of three small components, the overload controller, the overload agent and the HeatSpreader. Together, they are aimed at addressing two common problems in multicloud scenarios, i.e. cross site offloading and failover. Our evaluation tested these two scenarios and indicated that the mechanisms work, the overload controller managed to offload to another site when the load was increased above the threshold and the HeatSpreader detected and handled failover when one of the sites went down. For future work, we plan to investigate better healthchecking mechanisms that would improve the failover scenario and more effective algorithms for overload detection to improve the offloading scenario. It would also be interesting to include underload detection to complete the cycle of redistribution.

7 Acknowledgements

We acknowledge Atle Vesterkjær and Daniel Blueman at Numascale for technical support regarding the disaggregated hardware, Tomas ”Stric” Forsman for various systems wizardry, and the staff at the CSLab of NTUA for their insightful comments and input. We also thank the reviewers for their valuable feedback. Financial support has in part been provided by the European Community's Horizon 2020 Programme ([H2020/2014-2020]) under grant agreement no. 732366 (ACTiCLOUD), and the Kempe Foundation.

References

1. Africa, G., Chen, J., Shallcross, M.A., Thio, H.C.: Capturing configuration items from hybrid-cloud provisioning data (Apr 9 2019), uS Patent 10,257,289
2. Bagepalli, N.A., Chang, D.W.S., Patra, A., Anantha, M., Thumbargudi, P.: Programmable infrastructure gateway for enabling hybrid cloud services in a network environment (Sep 5 2017), uS Patent 9,755,858
3. Barroso, L.A., Hoelzle, U.: The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan and Claypool Publishers, 1st edn. (2009)
4. Bermbach, D., Kurze, T., Tai, S.: Cloud federation: Effects of federated compute resources on quality of service and cost*. In: 2013 IEEE International Conference on Cloud Engineering (IC2E). pp. 31–37 (March 2013). https://doi.org/10.1109/IC2E.2013.24
5. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing. pp. 13–16. ACM (2012)
6. Carvalho, M., Cirne, W., Brasileiro, F.V., Wilkes, J.: Long-term slos for reclaimed cloud computing resources. In: Proceedings of the ACM Symposium on Cloud Computing, 2014. pp. 20:1–20:13 (2014). https://doi.org/10.1145/2670979.2670999, http://doi.acm.org/10.1145/2670979.2670999
7. Delimitrou, C., Kozyrakis, C.: Quasar: Resource-efficient and qos-aware cluster management. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 127–144 (2014). https://doi.org/10.1145/2541940.2541941, http://doi.acm.org/10.1145/2541940.2541941
8. Eadline, D.: High performance computing for dummies, http://hpc.fs.uni-lj.si/sites/default/files/HPC_for_dummies.pdf
9. Ferrer, A.J., HernáNdez, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R.M., Djemame, K., et al.: Optimis: A holistic approach to cloud service provisioning. Future Generation Computer Systems **28**(1), 66–77 (2012)
10. Fielding, R.T., Taylor, R.N.: Architectural styles and the design of network-based software architectures, vol. 7. University of California, Irvine Doctoral dissertation (2000)
11. Foundation, O.: Openstack heat, https://docs.openstack.org/heat/latest/
12. Foundation, O.: Openstack pike version, https://releases.openstack.org/pike/
13. Gregg, B.: Linux load averages, http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html

14. Horn, G., Skrzypek, P.: Melodic: Utility based cross cloud deployment optimisation. In: 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA). pp. 360–367 (May 2018). https://doi.org/10.1109/WAINA.2018.00112
15. Hu, Q., Aazam, M., St-Hilaire, M.: Cloud resource allocation based on historical records: An analysis of different resource estimation functions. In: World Congress on Services. pp. 118–129. Springer (2018)
16. Kaplan, J.M., Forrest, W., Kindle, N.: Revolutionizing Data Center Energy Efficiency. Tech. rep., McKinsey & Company (2008)
17. Kurze, T., Klems, M., Bermbach, D., Lenk, A., Tai, S., Kunze, M.: Cloud federation. Cloud Computing **2011**, 32–38 (2011)
18. Li, H.: Cloud federation as a service (Dec 30 2014), uS Patent 8,924,569
19. Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., Kozyrakis, C.: Heracles: Improving resource efficiency at scale. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture. pp. 450–462. ISCA '15 (2015). https://doi.org/10.1145/2749469.2749475, http://doi.acm.org/10.1145/2749469.2749475
20. Moreno-Vozmediano, R., Huedo, E., Llorente, I.M., Montero, R.S., Massonet, P., Villari, M., Merlino, G., Celesti, A., Levin, A., Schour, L., Vázquez, C., Melis, J., Spahr, S., Whigham, D.: Beacon: A cloud network federation framework. In: Celesti, A., Leitner, P. (eds.) Advances in Service-Oriented and Cloud Computing. pp. 325–337. Springer International Publishing, Cham (2016)
21. Pivotal: Rabbitmq, https://www.rabbitmq.com/
22. Rimal, B.P., Choi, E., Lumb, I.: A taxonomy and survey of cloud computing systems. In: 2009 Fifth International Joint Conference on INC, IMS and IDC. pp. 44–51. Ieee (2009)
23. Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I.M., Montero, R., Wolfsthal, Y., Elmroth, E., Caceres, J., et al.: The reservoir model and architecture for open federated cloud computing. IBM Journal of Research and Development **53**(4), 4–1 (2009)
24. Rustad, E.: NumaConnect. Tech. rep., Numascale (2017)
25. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE Internet of Things Journal **3**(5), 637–646 (2016)