Computer Networks 168 (2020) 107041

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

FT-Aurora: A highly available IaaS cloud manager based on replication

Gustavo B. Heimovski [a], Rogério C. Turchetti [b,*], Juliano A. Wickboldt [c], Lisandro Z. Granville [c], Elias P. Duarte Jr [a]

[a] Department Informatics Federal University of Parana (UFPR) Curitiba PR Brazil
[b] CTISM Federal University of Santa Maria (UFSM) Santa Maria RS Brazil
[c] Institute of Informatics Federal University of Rio Grande do Sul (UFRGS) Porto Alegre RS Brazil

ARTICLE INFO

Article history:
Received 19 December 2018
Revised 25 October 2019
Accepted 29 November 2019
Available online 30 November 2019

Keywords:
Fault tolerance
Robust cloud
Data center

ABSTRACT

In this work we describe FT-Aurora, a highly available IaaS (Infrastructure as a Service) cloud manager that allows cloud resources to be accessed even if the manager itself crashes. FT-Aurora provides flexible and efficient resource management by supporting network programmability. FT-Aurora is based on clusters of cloud managers running on multiple datacenters. After a manager crashes, the corresponding resources remain accessible from any other manager in the cluster. A cluster consists of a group of managers that use fine-grained multi-master replication to share information. Replicated data and resources include both management information stored at the Aurora database, and information used to keep virtual machine images and processes. Replication and monitoring the multiple Aurora instances are available as services that can be easily activated through a GUI button. Experimental results are presented for both the performance and robustness of FT-Aurora.

1. Introduction

A very large number of systems and applications are currently based on cloud computing platforms, and this trend is likely to continue growing for several years [1]. Clouds are very convenient, as they allow users to share resources that can be rapidly provisioned and released and are maintained by the cloud provider [2,3]. Among the different types of clouds, in this work we focus on those that offer Infrastructure as a Service (IaaS), which enables users to access computing resources, storage, and networking, to deploy applications and systems [4–7].

As the number of organizations and individuals that rely on public and private cloud platforms for running their systems and applications grows, clouds face serious performance and dependability challenges [8–10]. Not only security has been considered a top dependability priority, but also high availability is essential [11,12]. Cloud outages can be very damaging for businesses that critically depend on the availability of the cloud to offer their services. Although there are several aspects of current cloud architectures that must be addressed to increase cloud dependability, in this work we focus on a critical component which is the *cloud manager*. This is usually a centralized component through which cloud resources are both accessed and managed. If the cloud manager crashes, the whole cloud becomes unavailable.

We propose a robust cloud infrastructure based on a crash-tolerant cloud manager. Multiple manager instances located on multiple datacenters employ replication to form a cluster of cloud managers. Replicated data and resources include management information data as well as virtual machine information and other components required to keep the resources of a given datacenter available even if the local cloud manager crashes. Two different scenarios are possible. In the first scenario, two or more replicated instances of the cloud manager run on the same datacenter. In the second scenario, replicas of the manager run on two or more datacenters, each responsible for the resources of the local datacenter. In both scenarios, each replication instance of the manager has access to all cloud resources. In case the manager crashes, another manager instance allows access to the corresponding resources.

In the scenario based on two or more datacenters, they are assumed to be connected across the Internet. If one of the managers crashes, another manager running on a different datacenter is used to access resources of the datacenter in which the cloud manager crashed. Note that all managers have access to all datacenters; thus all resources are available to all managers, which also allows to share resources across different clouds. Thus the solution can also perform load balancing among the several managers.

We implemented our highly available cloud manager on the Aurora cloud platform [13]. Aurora is an IaaS platform that

* Corresponding author.
E-mail addresses: gbheimovski@inf.ufpr.br (G.B. Heimovski), turchetti@redes.ufsm.br (R.C. Turchetti), jwickboldt@inf.ufrgs.br (J.A. Wickboldt), granville@inf.ufrgs.br (L.Z. Granville), elias@inf.ufpr.br (E.P. Duarte Jr).

https://doi.org/10.1016/j.comnet.2019.107041
1389-1286/

allows flexible resource management through network programmability features. A simple object-oriented API can be used by administrators to describe and run personalized programs for both application deployment and optimization. Multi-master replication [14,15] was used so that replicas form a cluster of cloud managers running on multiple datacenters. Not only the database that stores key cloud management information is replicated, but also the directories with files that keep virtual machine images, optimization programs, among other configuration information. Finally, we note that replication and monitoring of the multiple Aurora instances were implemented as services that can be easily activated by pushing a GUI button. We note that although some other cloud platforms offer similar functionalities, they all require knowledge of minute details and configuration of the system to achieve the same results.

The performance and robustness of the proposed system were evaluated experimentally. Results show the time to (1) incorporate a new manager instance to a cluster; (2) recover after a failure and (3) replicate data in different scenarios. The impact of the proposed solution was evaluated by measuring CPU and network usage. A stress test in which the link delay between two datacenters grows up to the limited supported by the replication solution is also reported. Finally, we also evaluated the system availability as a function of the MTBF (Mean Time Between Failures).

The remainder of this work is organized as follows. In Section 2 we present the Aurora Cloud Manager. Next, in Section 3 we describe FT-Aurora, the proposed robust IaaS cloud platform. The implementation and experimental results are presented in Section 4. Section 5 gives a overview of how some current cloud platforms deal with high availability.The conclusion follows in Section 6.

2. The aurora cloud manager

Aurora [13] is an IaaS cloud management platform that has 'programmability' features that allow flexible resource management. A simple object-oriented API allows the network operator to specify and run personalized programs for both protocol/application deployment and optimization. Aurora resources are those usually provided by IaaS cloud managers, i.e. computing, storage, and networking. Computing resources correspond to virtual machines and has features such as the number of CPUs and memory available. Storage corresponds to persistent data volumes which are allocated and managed on the datacenter. Finally, networking allows virtual resources to communicate.

Aurora supports virtual computing and storage by using the *Libvirt*[1] library. Networking is based on both OpenFlow [16] and the cloud monitoring tool *FlexACMS* (Flexible Automated Cloud Monitoring Slices) [17] which also employs *Nagios*[2] for monitoring both the virtual and physical layers of the managed infrastructure.

The architecture of the Aurora cloud manager is shown in Fig. 1. Two types of actors interact with the system: End-users and Administrators. An End-user requests virtual resources for deploying services or applications on the cloud. The Administrator works on behalf of the cloud provider, which is on the other side of a business relationship with the End-user. The main Aurora modules include: *Aurora Platform GUI* – this module corresponds to the system interface and that provides access to the system functionalities as well as Cloud Slice information to both Administrators and End-users; *Slice Space* – module that processes user requests for cloud resources; and *Programmable Space* – module that enables programmability. Other modules include the *Unified API* that provides a simple interface for manipulating all types of resources that may compose a *Cloud Slice*, including virtual machines, networking interfaces, virtual storage, among others; *Event Space* which is used to set up alarms; and the *Drivers* that implement the virtual device abstractions.

A *Cloud Slice* corresponds to an internal representation of the set of resources that compose the virtual infrastructure on which applications are deployed. After the creation of a *Cloud Slice*, the components of the *Programmable Space, Event Space*, and *Slice Space* interact to set up and manage resources throughout the application lifecycle. The operation of all these components is defined by the *Resource Management Programs & Metrics* specified by the Administrator.

The *Unified API* contains all operations for handling virtual resources, which are performed to set up and manage *Cloud Slices*. This API is *unified* in the sense that it provides a simple and single interface for resources that can be of four types: *Computing, Storage, Networking* and *Monitoring*. The *Computing* operations are those related to the usage and management of virtual machines, including operations to create, start (and re-start), stop, and migrate VMs. There are also the operations related to the management of operating system images to be deployed on the virtual machines. The *Storage* component allows the allocation of virtual storage volumes. The *Networking* component implements interfaces for the creation of virtual links and virtual routers. Finally, the *Monitoring* component of the *Unified API* is set up as the *Cloud Slice* is deployed. Among other monitoring features, this component includes the *Event Manager* that defines the event abstraction, which sets up alarms that are triggered as specified conditions occur.

Virtual device abstractions are implemented by a set of drivers. These drivers are technology-specific, and play two main roles: they provide an abstraction for using and setting up parameters for the configuration of devices. The *Libvirt*[3] library is employed for *Computing* and *Storage* components, while *Openflow* and *OpenVSwitch*[4] are used for *Networking*. Finally, *FlexACMS* and *Nagios* are the core of the *Monitoring* component. Most of the platform code is written in *Python* and implemented as a Web-based application built with the Django.[5]

3. FT-Aurora: A robust cloud manager

In this section, we describe FT-Aurora, the proposed robust cloud manager based on replication. A cluster of cloud managers is defined that share the data and information required to operate and manage any cloud managed by the cluster. The purpose is to guarantee the continuous operation of the cloud, even if a cloud manager fails. Two different types of replication are employed. The first is the replication of the database which is responsible for storing management information specifying the hosts (IP address and DNS names), the list of virtual machines installed, as well as information about routers, switches, the list of Cloud Slices deployed, among other resources. Furthermore, the directories with files that keep virtual machine images, optimization programs, among other configuration information are also replicated.

Fig. 2 shows the architecture of FT-Aurora, in the figure three replicated cloud managers are running on two datacenters. We assume that crashes do not disrupt the communication between the Aurora instances. In addition, the communication channel does not create, alter or loose messages. The cloud *Administrator* can access any of the three Aurora instances by using a single Virtual IP address. This single IP address is implemented with the *HAProxy*[6] load balancer, which is also responsible for enabling load balancing

[1] http://www.libvirt.org.
[2] http://www.nagios.org/.
[3] http://www.libvirt.org.
[4] http://openvswitch.org/.
[5] https://www.djangoproject.com.
[6] http://www.haproxy.org/.

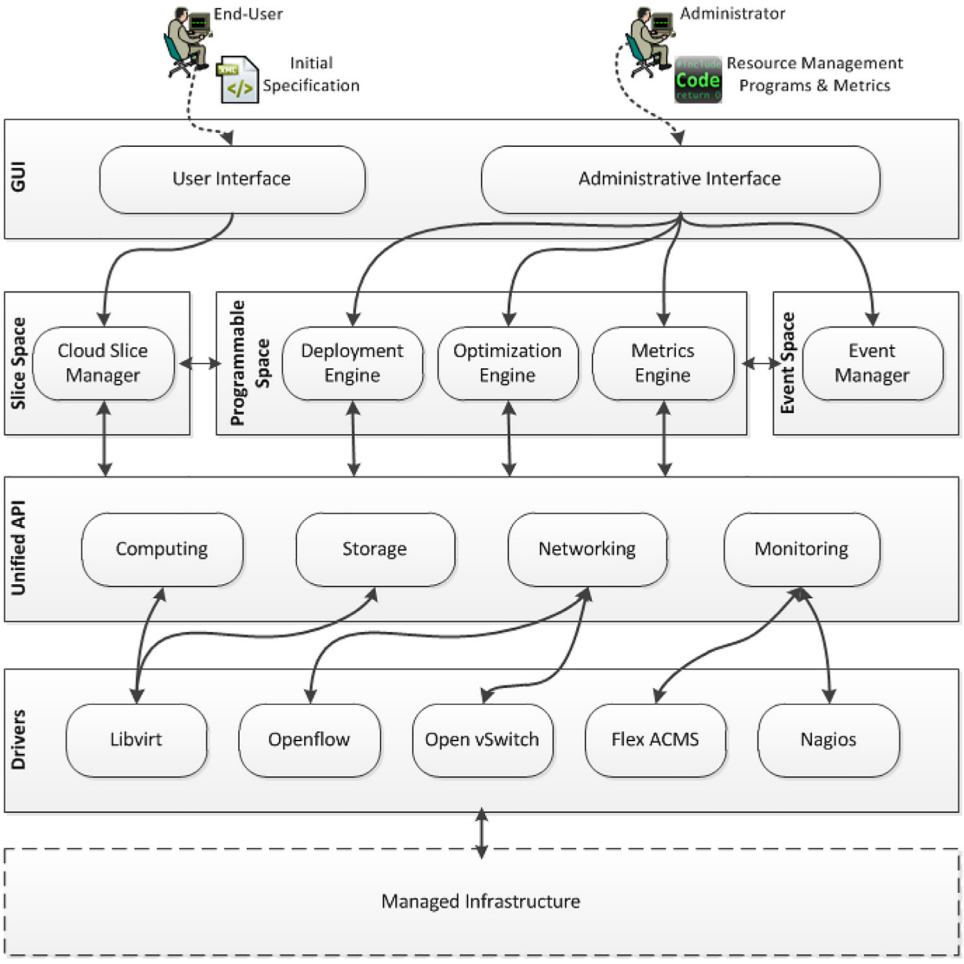


Fig. 1. Architecture of the aurora cloud manager [13].

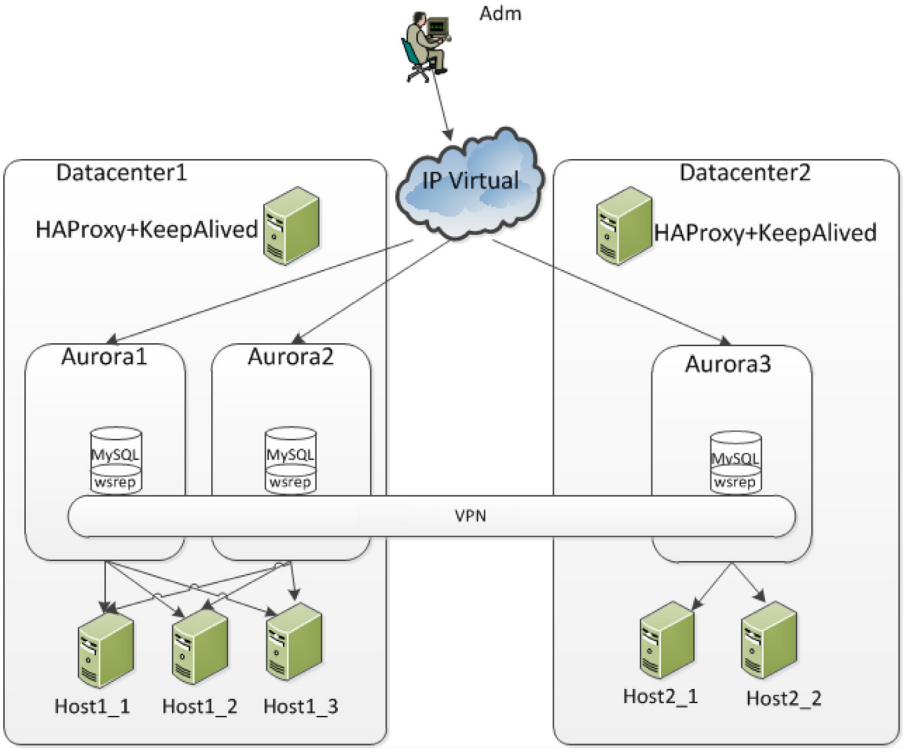


Fig. 2. FT-Aurora architecture: replicated cloud managers running on multiple datacenters.

by distributing user accesses across the Aurora instances that run on a single datacenter. By default, *HAProxy* implements load balancing using a *round-robin* strategy. In case there is only a single Aurora instance running on a given datacenter, a client always accesses that Aurora instance unless that instance has failed, in this case an instance on another datacenter is accessed providing transparent failover. Of course this also happens if there are multiple Aurora instances and all have failed. As shown in Fig. 2 an *HAProxy* instance runs on each datacenter, so that instances can fail but the service remains available. These processes are monitored with the *Keepalived*[7] tool which is used to detect and report crashes.

FT-Aurora replicates information about new resources added to a cloud managed by any one of the Aurora instances. For example, in the figure if the administrator adds a new host to the cloud in datacenter1, this information is replicated to a cloud manager in datacenter2, thus all Auroras instances have access to all the hosts, five in the figure, *Host1_1, Host1_2, Host1_3, Host2_1* and *Host2_2*. A Virtual Private Network (VPN) link is established between the datacenters to allow any cloud manager to communicate securely with any resource on any datacenter. The communication between instances on a single datacenter employs an encrypted *ssh* tunnel.

FT-Aurora employs multi-master replication, which keeps the multiple Aurora instances consistent and also allows reading and writing data to/from any instance. The main purpose is to keep each instance independent, but in case one of the instances crashes, the other instances can be used instead to access the corresponding resources. As mentioned above two types of replication were implemented, both of the database and of other directories/files keeping multiple resource information. We employed the MySQL database and for replication the Galera Cluster MySQL plugin[8], which implements multi-master replication among other strategies. The *rsync* [18] tool is used for remote file synchronization.

In order to use the replication features, the user just presses the *Aurora Cluster* button of the system GUI. Among other functionalities, the cluster is displayed including the list of participating Aurora instances and the state of each instance (crashed/alive). During the activation of the multi-master replication, it is necessary to restart the MySQL service. At this point, a new file is be created with information such as: *cluster* name, IP address of each Aurora instance that belongs to the *cluster*, among others. Galera provides notification features that are used to monitor the Aurora instances that belong a cluster. Notifications are generated whenever a cluster changes, e.g. a new Aurora instance joins a cluster or an instance crashes. Monitoring information is disseminated to all instances in the cluster. The Administrator is also notified of changes by e-mail.

4. Experimental results

In this section we report results of experiments executed to evaluate FT-Aurora. Three sets of experiments are presented. The first set was executed to evaluate the time that a new Aurora instance takes to join the Aurora cluster. The second set of experiments was designed to evaluate the replication latency, including the time to replicate the database and all the remote files that represent virtual machine images, optimization programs and metrics. We first evaluated the case in which all Aurora instances run in the same datacenter, and then evaluate a case in which replicas are in two different datacenters. The last set of experiments was executed to evaluate the availability of the system as the MTBF varies. Each Aurora instance was executed on a virtual machine with 512 MB of memory running Linux Ubuntu 14.04.02 LTS.

[7] http://www.keepalived.org/.
[8] http://galeracluster.com

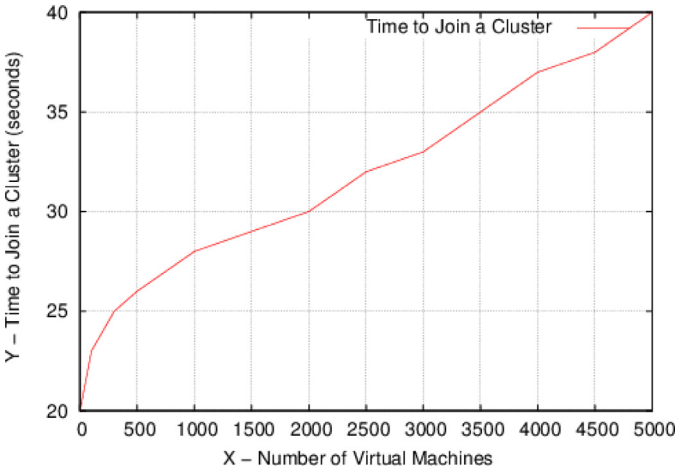


Fig. 3. Time for a new instance to join an Aurora cluster.

The first experiment was executed to measure the time that a new Aurora instance takes to join a cluster that consists of two Aurora instances. The process involves synchronizing information within the Aurora cluster. The new instance communicates with an instance which is already in the cluster. In order to evaluate the time it takes to complete the process, we increased the cluster size and thus the size of the corresponding database as shown along the x-axis of Fig. 3. It is clear that as the number of virtual machines increases, the time to join in the cluster also increases, but moderately. When there is only a single virtual machine, the time that the new instance takes to join in the cluster is about 20 s. As the number of virtual machines grows to 5000, the time to join the cluster only doubles to 40 s. This result shows that the replication strategy scales well as the number of resources running under the cloud manager grows.

The second set of experiments was executed to measure the replication latency. Three different experiments were run. In the first experiment we first measured the latency to replicate database information just after new configuration information had been added to an instance. Then, also in the first experiment, the latency to replicate virtual machine images was measured after a new virtual machine had been created on an Aurora instance. The size a virtual machine image is approximately 2.5 MB. The second experiment was executed to measure the replication latency as the amount of new data added to an Aurora instance increases. In the third experiment the replication latency was measured as the delay of the communication channel connecting the Aurora instances grew, considering a scenario with a remote Aurora instance.

The first experiment of the second set was executed on a single datacenter, initially with two Aurora instances, and next this number was increased to three instances. The time to synchronize the database by replicating information across two Aurora instances in on the order of milliseconds: as shown in Table 1 the average was 21.63ms. Each experiment was executed 30 times. Then to replicate the virtual machine images which were done using the *Rsync* tool, the latency was in average 4.7s. As the number of Aurora instances increases from two to three, the replication time also increases, reaching up to 36.95ms in average for the database replication while the image replication takes about 5.96s.

Next, an experiment is reported that was executed to investigate the latency (replication time) as the amount of information to replicate increases by the addition of new virtual machines. Furthermore we also measured the impact on CPU and network (bandwidth) usage. Note that memory usage kept constant, around 95% of the 512MB available at each virtual machine running an Aurora instance. This experiment was also executed first on two and then on three Aurora instances. Results are shown in Figs. 4 and 5.

Table 1
Data replication latency.

Replication Type/Scenario	2 Auroras		3 Auroras	
	Average	Standard Deviation	Average	Standard Deviation
MySQL Replication	21,63 ms	11,45 ms	36,95 ms	17,33 ms
Rsync Replication	4,7 s	0,95 s	5,96 s	1,56 s

Fig. 4 shows the replication time as the number of virtual machines increases. There are two curves, one corresponds to a system running two Aurora instances, and the other with three Aurora instances. We can see that the replication time grows roughly linearly as the number of virtual machines increases. The replication time also grows as the number of Aurora instances increases, reaching up to approximately 750ms using two Aurora instances and 4000 virtual machines, and up to approximately 1290ms for three Aurora instances. It is worth mentioning that these results reflect exactly the expected behavior.

Next we evaluate the overhead of the proposed strategy. Fig. 5 shows (a) CPU utilization and the (b) network bandwidth used as the number of virtual machines increases. As shown in Fig. 5(a), CPU utilization increases up to 92%, after which it remains roughly constant. The same pattern was observed for two and three Aurora instances. The amount of network bandwidth used shown in Fig. 5(b) also increases as the number of virtual machines grows. It is clear that the bandwidth required is roughly proportional to the number of virtual machines managed by the system. In terms of the number of Aurora instances, network bandwidth roughly doubles as the number of instances grows from two to three. The reason is that the time for three instances to run the replication requires an extra step in which data received from an instance has to be replicated to the other instance. Thus with three instances the time to replicate all the data among all instances doubles: the bandwidth required for a system with three instances is 9600 kB/s which it is 4360 kB/s for a system with two instances.

Next we describe an experiment designed to evaluate the replication time across two remote datacenters. We emulated the link connecting the remote sites by controlling the network delays with the *tc (Traffic Control)* tool. Database replication was triggered by the addition of new configuration to an Aurora instance. The database replication time was measured through the database *logs*. Fig. 6 shows the replication time as the link delay gradually increases. Note that we increased the delays up to the limit, i.e. a point from which the replication stops working. We found out that as the link delay reaches 4000ms the system stops working. When this limit is reached, Galera is not able to execute the replication process, in particular it cannot obtain the lock and write the transaction to the database anymore. In the figure it is also possible to see that as the delay reaches 1000ms the replication latency surges, which corresponds to the fact that it is increasingly hard for Galera to obtain the lock and write the transaction to the database.

In another experiment we increased the amount of RAM memory of each virtual machine to 1024 MB. The main purpose of this experiment was to investigate whether this causes any improvement on the replication latency. Actually, we observed that there is an improvement, in particular the limit of the link delay from which the system stopped working changed from 4000ms to 5000ms.

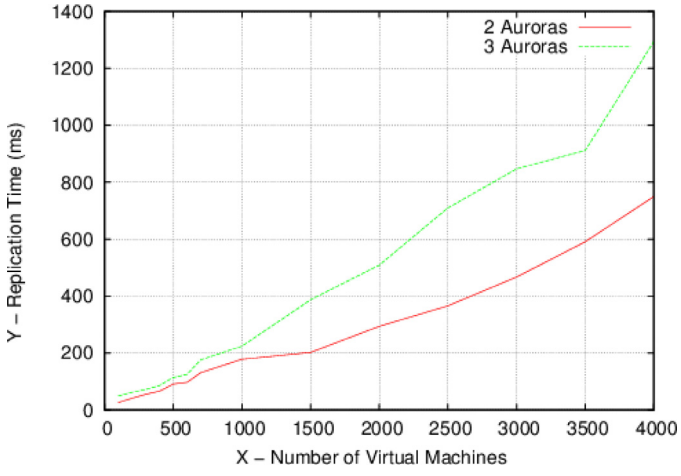
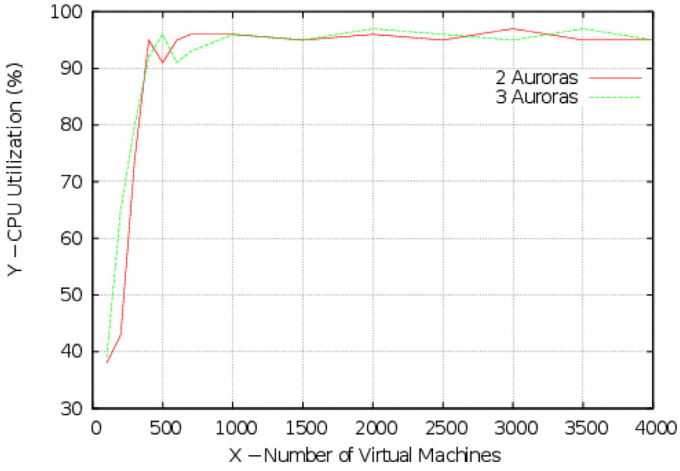
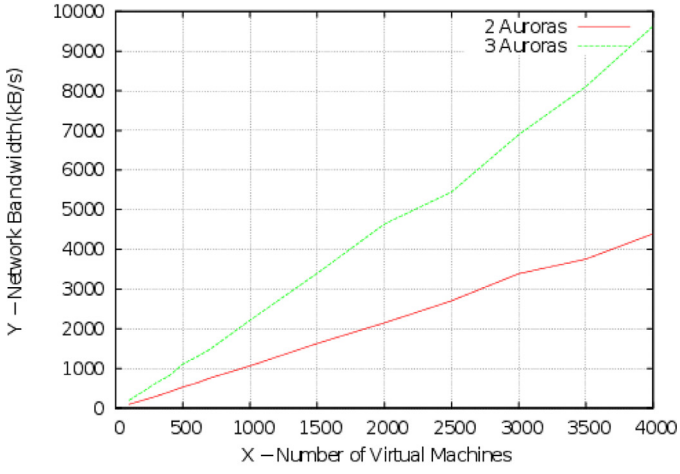


Fig. 4. Replication time as the number of managed virtual machines increases.



(a) CPU utilization.



(b) Network bandwidth required.

Fig. 5. Database replication: CPU utilization and network bandwidth required.

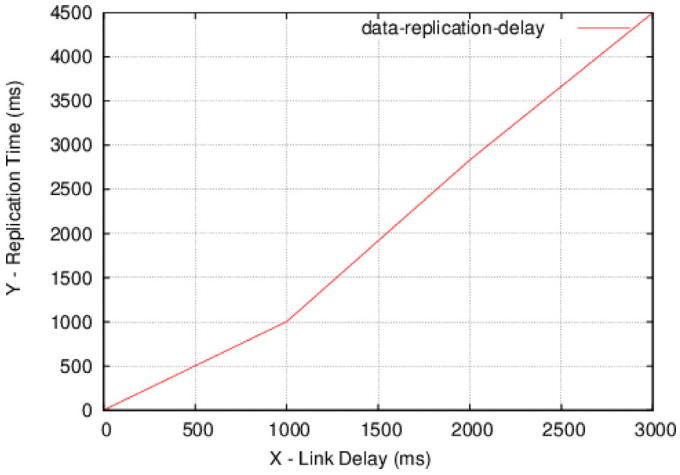


Fig. 6. Replication across two remote datacenters: reaching the limit.

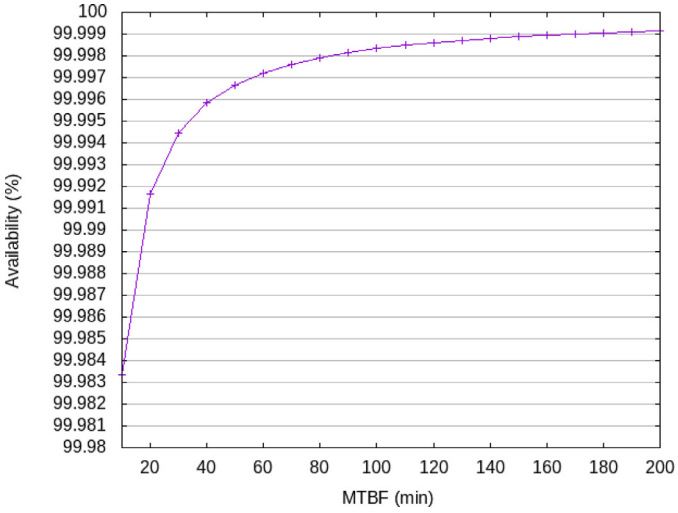


Fig. 7. FT-Aurora Availability: reaching the 99.999% level.

Table 2 shows how the replication delay and the time to access an Aurora instance increase as the communication delay increases. Each experiment was executed 50 times, averages are shown. As mentioned above, as the communication delay reaches 4s the system stops working.

The last set of experiments was executed to evaluate the system availability. Both the failure detection time and the time to recover after a failure were measured. The availability was computed as a function of the Mean Time Between Failures (MTBF). Monitoring was set up so that an Aurora instance is polled in intervals of 2s, and is considered to have crashed when polling fails twice; thus the detection time is 4s. After an Aurora instance crashes, all accesses are directed to other instances. On the other hand, Galera is employed to determine whether MySQL has crashed. In this case the detection time is shorter, in fact it has been measured to be at most 15 ms. Note that Galera also detects partitions. If an Aurora instance gets unreachable for 5 s it is suspected to have failed, but it is only removed from the cluster after 15 s.

The MySQL recovery time also was measured. In this experiment the crash of a MySQL server was forced, and then this server was restarted. The experiment was executed in clusters with two and three Aurora instances. The recovery time was 16 s in the system with two Aurora instances, and 18 s for 3 instances.

Table 2
Impact of the communication delay on the replication time.

Delay (ms)	Communication Delay (ms)	Replication Latency (ms)
0	0.485	0.521
10	10.834	10.914
20	20.736	20.875
30	30.713	30.754
40	40.749	40.813
50	50.753	50.824
60	60.886	60.905
70	70.452	70.678
80	80.742	80.905
90	90.797	90.956
100	100.814	101.894
200	200.812	201.252
300	300.930	302.083
400	400.714	401.800
500	500.768	502.681
1000	1000.883	1002.478
2000	2000.740	2832.474
3000	3000.905	4498.797
4000	4000.857	–––

Table 3 shows the system availability computed as the MTBF varies. The availability was computed as follows:

$$availability = \frac{(MTBF - (im * ns)) * 100)}{MTBF}$$

In the expression, im is the monitoring interval, ns represents the number of times an instance must be considered suspect before it is classified as crashed. We consider that the monitoring interval corresponds to the time the time the system is unavailable. We computed the availability for a MTBF of 1 minute, 30 min, 1 h, 2 h and 4 h. The number of suspicions required (ns) was equal to 1, 2 and 5. For example, if the MTBF is 30 min (18000 s), im is 2 s and ns is 5 s, the availability is equal to $((1800 - (2 * 5)) * 100)/1800 = 99,44\%$. The parameters do have an impact on the availability and accuracy, and have to be set accordingly. For example, when $ns = 5$ the availability is lower in comparison to $ns = 2$ or $ns = 1$, however the accuracy is greater in the sense more suspicions are required to conclude that a crash has actually happened.

According to the IEEE document [19], the power industry – which has some of the strictest specifications – requires an availability is 99.999% (five9's). The FT-Aurora architecture actually allows parameters to be set so that the required availability levels are reached. In particular, the monitoring interval can be set low enough so that failures are quickly detected and the system is reconfigured to keep the downtimes as low as possible. Nevertheless it is important to see that the availability is also dependent on the mean time between failures (MTBF). Thus if failures are extremely frequent the availability will drop, no matter how short the monitoring interval is and how efficient the recovery procedures are. We ran a new experiment and show results for the availability in Fig. 7, Section 4. The experiment shows that FT-Aurora reaches the 99.999% availability level even if the MBTF is as low as 170 min with the monitoring interval set to 100 milliseconds.

As a final comment, we would like to recall that, as described in Section 3, a user communicates with the cloud manager using a virtual IPaddress. As the local manager crashes, the IP address is reconfigured as the address of another working manager of the cluster. Thus accessing virtual resources with a failed local manager is completely transparent to the user. Now consider the time it takes to access resources if the manager fails. This corresponds to the round trip time to the working manager. If this instance is running on the local datacenter, the round trip time is less a millisecond. If, on the other hand, it is on a remote datacenter, the round trip time takes from a few milliseconds to a few hundreds of milliseconds depending on how far the datacenters are in the

Table 3
System availability.

MTBF	Monitoring Interval (im)	Number of Suspicions (vf)	Availability (%)
1 min	2 s	1	96,66
		2	93,33
		5	83,33
30 min	2 s	1	99,88
		2	99,77
		5	99,44
1 h	2 s	1	99,94
		2	99,88
		5	99,72
2 h	2 s	1	99,97
		2	99,94
		5	99,86
4 h	2 s	1	99,98
		2	99,97
		5	99,93

Internet. As a comment, if required one can place replicas as close as possible in order to reduce the latency in case of failures.

5. Related work

In this section, we briefly describe related work, with a focus on the strategies adopted by popular cloud computing platforms to deploy highly available resource management.

OpenStack[9] is an IaaS cloud platform which can be deployed on large systems and can manage a large number of resources which can be dynamically allocated on demand. OpenStack implements high availability by duplicating all the components of a given service and running them on different controllers.[10] That is a major difference to our approach as we employ fine grained replication of the basic resources needed for a cloud manager to be able to manage resources of another cloud manager. OpenStack allows different strategies to be used such as basic master-slave replication and also multi-master replication which defines replication clusters. But all the software required to run the replication and the OpenStack high-availability set up must be manually configured step by step by the user that needs the feature. This is another key difference to FT-Aurora.

CloudStack[11] is a another cloud platform designed to deploy and manage large IaaS clouds. In order to increase manager availability, the human manager can replicate the management server. However, replication is only allowed within one datacenter.[12] Also related to high-availability is the CloudStack strategy to replicate the storage which defines data zones, one of which runs a *primary database*, while the others run backups, which are synchronized with the primary.

Eucalyptus[13] is an IaaS infrastructure designed to implement, manage and provide both private and hybrid clouds. The Eucalyptus architecture provides fault tolerance mechanisms at the host level, i.e. a host (and the services running on that host) can be replicated so that if the original host crashes it is replaced by the backup in a transparent way. Regarding the availability of the cloud manager, it is not yet provided but seems to be planned for the future.[14]

OpenNebula[15] is a cloud computing platform that presents a simple architecture to allowing the creation and management of public and private IaaS clouds on a datacenter. OpenNebula provides strategies for the replication of both virtual machines and hosts by using the basic master-slave approach. OpenNebula uses a distributed consensus protocol to provide fault-tolerance and state consistency across the replicas.

Related work also includes [12], which presents a performance evaluation of different data replication strategies for the cloud. A survey of data replication techniques for cloud computing is presented in [11], with a taxonomy that encompasses multiple aspects of the different strategies. In [10], Nielsen and others propose a protocol for replicating data across data centers with a focus on the sensitivity to packet losses and varying round-trip times. Another work that explores data replication in the context of cloud computing is [9], which presents a strategy based on parallel state machine replication. In [7], the authors highlight that improving the availability of cloud computing is one of the main challenges for the future.

6. Conclusion

In this work we presented FT-Aurora, a highly available IaaS cloud manager that employs fine-grained replication across clusters of managers running on multiple datacenters. After a manager crashes, the corresponding resources remain accessible from any other manager in the cluster. All functionality related to replication and high-availability can be easily actived through the GUI interface. The performance and robustness of the proposed solution were empirically evaluated. We performed experiments to measure the time to incorporate a new manager instance to an Aurora cluster as well as the latency to replicate data in different scenarios, including clusters with managers located in different datacenters, each responsible of a set of resources. We also computed the system availability based on the time to detect failures and recover. Future work includes building clusters of multiple remote datacenters that share resources through a cloud federation.

Declaration of Competing Interest

This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue.

The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript.

[9] http://openstack.org/.
[10] http://docs.openstack.org/ha-guide/.
[11] https://cloudstack.apache.org/.
[12] http://docs.cloudstack.apache.org/projects/cloudstack-administration/en/4.8/reliability.html#limitations-on-database-high-availability.
[13] http://www.eucalyptus.com/.
[14] https://docs.eucalyptus.com/eucalyptus/4.0.2/install-guide/ha_planning.html.
[15] https://opennebula.org/.

CRediT authorship contribution statement

Gustavo B. Heimovski: Conceptualization, Methodology, Software, Validation, Data curation, Writing - Original Draft, Writing - Review & Editing, Funding acquisition. **Rogério C. Turchetti:** Conceptualization, Methodology, Software, Validation, Data curation, Writing - Original Draft, Writing - Review & Editing, Funding acquisition. **Juliano A. Wickboldt:** Conceptualization, Methodology, Software, Validation, Data curation, Writing - Original Draft, Writing - Review & Editing, Funding acquisition. **Lisandro Z. Granville:** Conceptualization, Methodology, Software, Validation, Data curation, Writing - Original Draft, Writing - Review & Editing, Funding acquisition. **Elias P. Duarte Jr:** Conceptualization, Methodology, Software, Validation, Data curation, Writing - Original Draft, Writing - Review & Editing, Funding acquisition.

Acknowledgement

This work was partially supported by the Brazilian Research Council - CNPq grants 311451/2016-0, 312392/2017-6, and 313893/2018-7.

References

[1] R. Buyya, S.N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L.M. Vaquero, M.A.S. Netto, A.N. Toosi, M.A. Rodriguez, I.M. Llorente, S. Vimercati, P. Samarati, D. Milojicic, C. Varela, R. Bahsoon, M.D. Assuncao, O. Rana, W. Zhou, H. Jin, W. Gentzsch, A.Y. Zomaya, H. Shen, A manifesto for future generation cloud computing: research directions for the next decade, ACM Comput. Surv. 51 (5) (2019) 1–38.

[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58, doi:10.1145/1721654.1721672.

[3] Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-The-Art and research challenges, J. Internet Serv. Appl. 1 (1) (2010) 7–18, doi:10.1007/s13174-010-0007-6.

[4] P.M. Mell, T. Grance, SP 800-145. The NIST Definition of Cloud Computing, Technical Report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011.

[5] K. Tsakalozos, M. Roussopoulos, A. Delis, VM Placement in non-Homogeneous IaaS-Clouds, in: G. Kappel, Z. Maamar, H. Motahari-Nezhad (Eds.), Service-Oriented Computing, Springer Berlin Heidelberg, 2011, pp. 172–187, doi:10.1007/978-3-642-25535-9_12. volume 7084 of Lecture Notes in Computer Science

[6] R. Jhawar, V. Piuri, Fault tolerance management in iaas clouds, in: 2012 IEEE First AESS European Conference on Satellite Telecommunications (ESTEL), 2012, pp. 1–6.

[7] A. Taherkordi, F. Zahid, Y. Verginadis, G. Horn, Future cloud systems design: challenges and research directions, IEEE Access (2018).

[8] M.A.K. Kholghi, A. Abdullah, R. Latip, S. Subramaniam, M. Othman, Disaster recovery in cloud computing: a survey, Comput. Inf. Sci. 7 (4) (2014) 39–54, doi:10.5539/cis.v7n4p39.

[9] L. Wu, W. Wu, N. Huang, Z. Chen, Pdfe: flexible parallel state machine replication for cloud computing, in: 2018 IEEE International Conference on Cluster Computing (CLUSTER), 2018, pp. 456–465.

[10] L.H. Nielsen, B. Schlie, D.E. Lucani, Towards an optimized cloud replication protocol, in: 2018 IEEE International Conference on Smart Cloud (SmartCloud), 2018, pp. 105–110.

[11] B.A. Milani, N.J. Navimipour, A comprehensive review of the data replication techniques in the cloud environments: major trends and future directions, J. Netw. Comput. Appl. 64 (2016) 229–238.

[12] S. George, E.B. Edwin, A review on data replication strategy in cloud computing, in: 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2017, pp. 01–04.

[13] J.A. Wickboldt, R.P. Esteves, M.B. de Carvalho, L.Z. Granville, Resource Management in IaaS Cloud Platforms Made Flexible Through Programmability, Comput. Netw. 68 (0) (2014) 54–70. Communications and Networking in the Cloud doi:10.1016/j.comnet.2014.02.018

[14] , Replication: Theory and Practice, B. Charron-Bost, F. Pedone, A. Schiper (Eds.), Springer-Verlag, Berlin, Heidelberg, 2010.

[15] B.P. Gautam, K. Wasaki, A. Batajoo, S. Shrestha, S. Kazuhiko, Multi-master replication of enhanced learning assistant system in iot cluster, in: 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), 2016, pp. 1006–1012.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74, doi:10.1145/1355734.1355746.

[17] M. Barbosa de Carvalho, R. Pereira Esteves, G. da Cunha Rodrigues, L. Zambenedetti Granville, L.M. Rockenbach Tarouco, A Cloud Monitoring Framework for Self-Configured Monitoring Slices Based on Multiple Tools, in: 9th International Conference on Network and Service Management (CNSM), 2013, pp. 180–184.

[18] A. Tridgell, Efficient Algorithms for Sorting and Synchronization, Australian National University Canberra, 1999 Ph.D. thesis.

[19] I.R. Practice, IEEE Recommended practice for the use of probability methods for conducting a reliability analysis ofindustrial and commercial power systems, IEEE Std 3006.5–2014 (2015) 1–50, doi:10.1109/IEEESTD.2015.7034995.

Gustavo B. Heimovski received the M.Sc. degree in computer science from Federal University of Parana, Brazil, 2015, where he was also a student member of the Computer Networks and Distributed Systems Lab (LaRSis). His recent research is focused on the dependability in Distributed Systems.

Rogerio C. Turchetti is an Adjunct Professor at Federal University of Santa Maria, Santa Maria, Brazil. He received a Ph.D. degree in computer science from Federal University of Parana, Brazil, 2017, the M.Sc. degrees in production engineering with emphasis on information systems from Federal University of Santa Maria, Santa Maria, Brazil, in 2006. His research interests include Computer Network and Distributed Systems, their Dependability and Algorithms. His recent research is focused on the dependability in Network Function Virtualization and Software Defined Network.

Juliano Araujo Wickboldt is an associate professor at the Federal University of Rio Grande do Sul (UFRGS) in Brazil. He holds both M.Sc. (2010) and Ph.D. (2015) degrees in computer science from UFRGS. Juliano was an intern at NEC Labs Europe in Heidelberg, Germany for one year between 2011 and 2012. In 2015, Juliano was a visiting researcher at the Waterford Institute of Technology in Ireland. His research interests include softwarized networking and 5G technologies.

Lisandro Zambenedetti Granville is a professor at the Federal University of Rio Grande do Sul. He served as the TPC Co- Chair of IFIP/IEEE DSOM 2007 and IFIP/IEEE NOMS 2010, the General Co-Chair of IFIP/IEEE CNSM 2014, the TPC Co-Chair of IEEE NetSoft 2018, and the TPC Vice-Chair do IEEE ICC 2018. He is President of the Brazilian Computer Society (SBC). Lisandro's interests include network management, software-defined networking, and network functions virtualization.

Elias P. Duarte Jr. is a Full Professor at Federal University of Parana, Curitiba, Brazil, where he is the leader of the Computer Networks and Distributed Systems Lab (LaRSis). His research interests include Computer Networks and Distributed Systems, their Dependability, Management, and Algorithms. He has published nearly 200 peer-reviewer papers and has supervised more than 125 students both on the graduate and undergraduate levels. Prof. Duarte is currently Associate Editor of the IEEE Transactions on Dependable and Secure Computing, and has served as chair of more than 20 conferences and workshops in his fields of interest. He received a Ph.D. degree in Computer Science from Tokyo Institute of Technology, Japan, 1997, M.Sc. degree in Telecommunications from the Polytechnical University of Madrid, Spain, 1991, and both BSc and MSc degrees in Computer Science from Federal University of Minas Gerais, Brazil, 1987 and 1991, respectively. He has chaired the Special Interest Group on Fault Tolerant Computing of the Brazilian Computing Society (2005–2007); the Graduate Program in Computer Science of UFPR (2006–2008); and the Brazilian National Laboratory on Computer Networks (2012–2016). He is a member of the Brazilian Computing Society and a Senior Member of the IEEE.