Disaster Recovery Layer for Distributed OpenStack Deployments

L. Tomás, P. Kokkinos, V. Anagnostopoulos, O. Feder, D. Kyriazis, K. Meth, E. Varvarigos, T. Varvarigou

Abstract— We present the Disaster Recovery Layer (DRL) that enables OpenStack-managed datacenter workloads, Virtual Machines (VMs) and Volumes, to be protected and recovered in another datacenter, in case of a disaster. This work has been carried out in the context of the EU FP7 ORBIT project that develops technologies for enabling business continuity as a service. The DRL framework is based on a number of autonomous components and extensions of OpenStack modules, while its functionalities are available through OpenStack's Horizon UI and command line interface. Also, the DRL's architecture is extensible, allowing for the easy and dynamic integration of protection, restoration and orchestration plug-ins that adopt new approaches. A distributed disaster detection mechanism was also developed for identifying datacenter disasters and alerting the DRL. For the evaluation of the DRL, a two (active and backup) datacenters testbed has been setup in respective sites in Umeå and Luleå, 265km apart and connected through the Swedish national research and education network. In case of a disaster, traffic is redirected between the datacenters utilizing the BGP anycast scheme. The experiments performed, show that DRL can efficiently protect VMs and Volumes, with minimum service disruption in case of failures and low overhead, even when the available bandwidth is limited.

Index Terms— Disaster Recovery, OpenStack, disaster detection, traffic redirection, testbed, MAN/WAN, inter-datacenter network

1 Introduction

Our absolute dependence on information technology resources along with a number of catastrophic (e.g. Great East Japan earthquake, 9/11) and other events (human errors and failures, such as Amazon's EC2 service disruption [1]) have put Disaster Recovery (DR) in the spotlight.

DR involves a set of practices and activities aiming at the integrity or the continuity of operation of the physical and virtual information technology assets of an organization, despite significant disruptive events. Particularly, DR practices are based on the continuous protection of resources (VM-image, VM-storage, storage, application), using a primary and a secondary datacenter connected through a Metropolitan, a Wide Area Network (MAN/WAN) or an inter-datacenter network. The secondary datacenter is ready to pick up work in case the primary one fails. In particular, DR life cycle has three main phases [2]: i) deployment, where the primary site and the secondary site(s) are set up for supporting DR; ii) synchronization, which is characterized by continuous data replication from the primary site to the secondary site; and iii) failover, referring to the recovery of the primary site at the backup.

In this work, we describe a Disaster Recovery Layer (DRL) that we implemented for OpenStack [3] -based datacenters that enables Virtual Machines (VMs) and their data storage (volumes) to be protected and recovered in another datacenter, in case of a disaster, as shown in Figure 1. Our work has been performed in the context of the EU FP7 ORBIT project [4][5] whose purpose was to develop technologies for the provision of business continuity as a service. Business continuity views an organization from a more general point of view, focusing on what needs to be done in order to keep the business running in the aftermath of a disaster, identifying triggering events, required procedures, involved entities (physical or virtual), and defining related priorities [6][7]. Every successful business continuity strategy needs an effective DR plan and respective mechanisms.

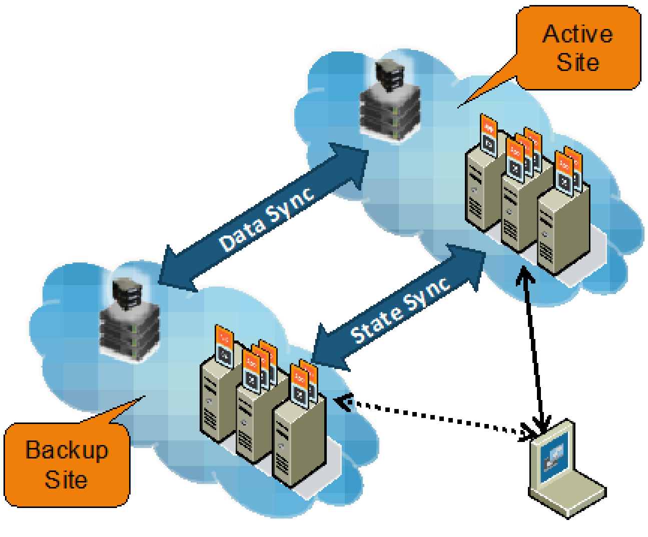


Figure 1 Disaster Recovery (DR) overview.

- *L. Tomás is with the Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden. E-mail: luis@cs.umu.se (now at Red Hat, Madrid, Spain)*
- *P.Kokkinos and E. Varvarigos are with Computer Technology Institute and Press Diophantus, Patras University Campus, Rio Patras, Greece. E-mail: kokkinop@ceid.upatras.gr and manos@ceid.upatras.gr*
- *V. Anagnostopoulos, T. Varvarigou are with National Technical University of Athens, Iroon Polytechniou 9, Athens, Greece, E-mail: dimos@mail.ntua.gr*
- *D. Kyriazis is with Department of Digital Systems, University of Piraeus, Piraeus, Greece, E-mail: dimos@unipi.gr*
- *O. Feder and K. Meth are with IBM Haifa Research Lab, E-mail: meth@il.ibm.com and oshritf@il.ibm.com*

The main design goals of the Disaster Recovery Layer (DRL) were the following: efficient OpenStack integration, extensibility of the protection and restoration approaches, low-resource overhead, fast recovery and transparent operation. These goals where achieved through the completion of four major tasks:

- Extend the OpenStack cloud management platform so as to enable DR. The DRL framework is based on a number of autonomous components and extensions to OpenStack modules, whose functionalities are available through OpenStack's Horizon UI and command line interface.
- Develop mechanisms for the synchronization of VM state and data as well as for their recovery in case of a disaster. The DRL's extensible architecture allows easy and dynamic integration of protection, restoration and orchestration plug-ins that adopt new approaches (e.g., taking into account available bandwidth between sites) and serve (protect and recover) additional kind of resources.
- Enable accurate and in time detection of when and where a failure has occured over geographically distributed datacenter deployments. A robust and distributed disaster detection mechanism has been developed for this purpose, identifying datacenter disasters and alerting the DRL.
- Support the operation of the DRL over low latency and variable bandwidth networks, enabling a smooth and transparent DRL operation. For the evaluation of the DRL, a testbed consisting of an active and a backup datacenter has been setup in respective sites in Sweden, Umeå and Luleå, which are 265km apart and are connected through the Swedish national research and education network. In case of a disaster, traffic is redirected between the datacenters utilizing the BGP anycast scheme.

The results of our evaluation show that DRL can efficiently protect the workloads (VMs and volumes), with minimum service disruption in case of failures/disaster and low overhead, even when the two sites are connected over a long distance public or inter-datacenter network.

The remaining of this paper is organized as follows. In Section 2 we report on previous work. Section 3 presents the DRL architecture and the protection and restoration policies implemented. The integration of DRL in OpenStack is presented in Section 4. In Section 5 we describe the implemented failure detection framework. The testbed setup, the traffic redirection mechanism adopted and the experiments performed are described in Sections 6 and 7. Finally, in Section 8 we conclude our work.

2 Related Work

Disaster Recovery (DR) is not a new concept; related techniques first appeared around late 70s when large computer mainframes became an important part of many businesses, and tapes were the cornerstone of these DR strategies. The works in [8][9] present indicative cost savings from cloud based DR, involving virtual resources, in relation to "physical" DR. In our work, when we use the term DR we implicitly refer to cloud DR, except if stated otherwise.

A survey on cloud-based DR is presented in [10], while [11] provides an overview of the networking issues of live migration and DR over long distance networks (e.g., MAN/WAN, inter-datacenter), and discusses memory and storage issues, traffic redirection techniques, enabling networking technologies, along with the related scientific publications and commercial products. The tradeoffs involved in the design of DR mechanisms are studied in [16], where the authors consider parameters such as the cost of data transfers and storage, the organization's objectives, and the type of disasters.

Complete cloud based DR systems are presented in [2][12][14][15]. In [2] authors describe their experiences from providing Disaster Recovery (DR) on IBM's managed cloud platform, Cloud Managed Services (CMS), which is distributed globally across multiple sites. SecondSite [12] is a cloud-based service based on [13], for disaster tolerance of VMs across cloud datacenters and over WAN, which considers the issues of reducing the replication traffic, of failure detection and of traffic redirection. The DR approach in [14] focuses only on storage migration between the primary and the backup sites. The proposed mechanism pipelines storage replication operations and computations performed in a multi-tier application, resulting in throughput and response time performance close to that of asynchronous replication, with the same Recovery Point Objective (RPO) guarantees as synchronous replication. In [15], the authors introduced the Cloud Standby DR approach where the backup system is started from time to time and it is updated with new replication data. Also, fault detection techniques and mechanisms have been developed for cloud and other distributed environments in order to detect failures, malicious behaviors and other anomalies [12][22][23]. In all cases the reliability and the speed of the detection are of paramount importance. Additionally, a number of works [25][26][27] are motived by cloud DR scenarios and focus on related resource management issues, some of which were also identified in [16]. [27] considers the problem of designing a disaster-resilient network of datacenters, operating over a WAN, with zero VM loss of state (e.g., loss of disk and memory content) in case of a disaster. There are also few open source based efforts, such as the multi-region disaster recovery with OpenStack and Ceph [39].

Today, several companies provide DR software products and services, targeting zero data losses and recovery time in the event of a disaster [18]. [19][20] present a comprehensive list of backup and DR platforms. DR can also be provided as a service (Disaster Recovery as a Service, abbreviated DRaaS), replicating physical (host) or virtual (VM) servers to cloud resources. Using DRaaS an organization can reduce its DR related expenses, since it does not have to invest in owning off-site DR facilities that are generally underutilized till a disaster hits. Gartner reports in [21] the most important DRaaS offerings. The DR operation also requires increased networking resources that grow as replication needs increase. Under the term “WAN optimization” or “WAN acceleration” we find a set of technologies and techniques to improve the efficiency of data flow in long distances. WAN optimization is either part of a DR software or comes in separate (software and hardware) products, which can be used in combination with it. Gartner reports in [24] the major players in the “stand-alone” WAN optimization marker. Software Defined Networking (SDN) has also been suggested as an enabling technology for disaster-resilient services and corresponding operations, such as traffic redirection [11][28][29].

In our work, we implemented a disaster recovery layer (abbreviated DRL) for OpenStack. This is one of the first scientific works presenting a complete DR solution for OpenStack-based sites, connected through a MAN/WAN or inter-datacenter network. Our work generally, considers similar topics as [12], protecting both memory/state and data/storage in an asynchronous but continuous manner (so the backup site is always up, in contrast to [15]). Our work, also implements a distributed by-design failure detection mechanism as opposed to the centralized approach of [12]. Also, though our work is more practical, a number of theorical research challenges are raised in protection, restoration and orchestration policies. One such challenge is the trade-off that needs to be achieved between the network resources used for data replication and the network resources used by the applications hosted in the VMs. In addition, the modular based implementation of DRL and the fact that it is open sourced [35][36][37], allows new such policies to be implemented and integrated in DRL in the form of plug-ins. DRL can also be provided as a service, assuming Openstack-based cloud datacenters. [39] proposes a similar solution to ours but based on Ceph instead of DRBD for data replication. In our case, using DRBD allows us to extend the protection modules to account for different priorities in replicated traffic, as presented in [34], that way reducing the chances of losing data upon a disaster, as well as limiting the impact due to replication on running applications.

3 DRL Architecture Overview

Disaster Recover Layer (DRL) is based on a number of autonomous components and extensions to OpenStack’s [3] main modules. Figure 2 shows an architecture overview of the DRL. Its main components are: i) the DR-Orchestrator, ii) the DR-Engine and iii) the DR-Logic.

The DR-Orchestrator is aware of the different resources set in Disaster Recovery (DR) mode and orchestrates their continuous protection and recovery when a major failure actually occurs, as notified by the failure detection (watchdog) framework (see Section 5). The DR-Orchestrator queries DR-Engine, in order to setup, manage and execute protection and recovery policies. The DR-Logic selects the resources (VM, images, volumes, ssh keys, networks, etc) to be included in a recovery policy and the actions to invoke on the various datacenters. Note a recovery policy is a logical entity that includes a reference to all the data protected (e.g., networks, VMs, volumes) and that is needed for the recovery actions. It mainly consists of a heat template defining how the different OpenStack resources link to each other and pointing to the right data sources. The DR-Logic also assists the DR-Orchestrator in performing various synchronization actions efficiently, for example by deciding when a new protection action needs to be triggered in order to update the information available at the back-up datacenter.

The DR-Engine invokes for each of the protected resources a relevant policy, pulling needed metadata, configurations and resource data and saving them in a predefined vault. In our work, we used OpenStack Swift on the backup site as this vault. The metadata are saved as an OpenStack Heat Template (HOT), simplifying the automate operations performed during recovery. Flame open source tool [38] was used in order to generate HOT from an existing infrastructure. Upon a disaster, the failure detection (watchdog) framework (Section 5) notifies the DR-Orchestrator, which in turn drives the recovery operations by utilizing the DR-Engine to recreate all the resources at the chosen backup site. This includes pulling the metadata from the vault, populating them with the new site variants and then invoking Heat to recover the resources’ workloads while ensuring data consistency. This last step includes importing the resources on the backup OpenStack-based site and reconnecting them with the same configuration (e.g., network) as at the primary site and up-to-date data (e.g., attach replicated volumes).

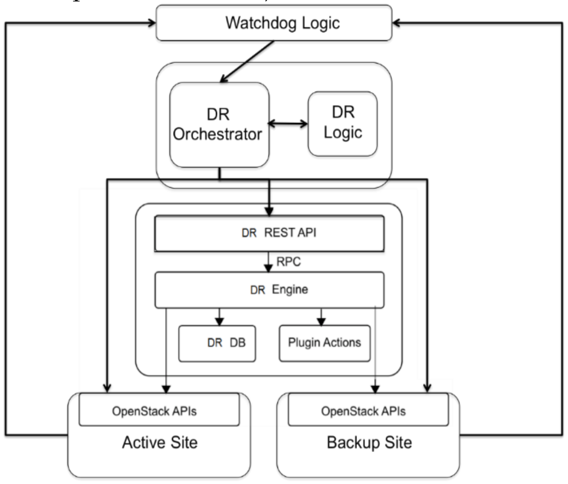


Figure 2: Disaster Recovery Layer (DRL) architecture overview.

DR-Orchestrator’s functionalities are accessible through a set of APIs, while in turn, the DR-Orchestrator makes use of the APIs offered by the DR-Logic, DR-Engine and OpenStack, in order to perform actions requested by the users or the failure detection module, query VMs and volumes status, enable or disable various actions, and clean up previous backed up data that are no longer needed.

The DR-Engine backend is extendable and supports multiple resources and plugin actions (DR policies). Similar to the DR-Engine, the DR-Logic also follows a plug-in based architecture in which plug-ins can be easily implemented and dynamically loaded by the DR-Orchestrator. In this way, new types of protection (e.g. snapshot, mirroring), restoration and orchestration policies can be added, while

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2745560, IEEE Transactions on Cloud Computing

also allowing new types of resources to be protected/recovered that need special handling. The recovery time using each of the policies may vary, affecting the downtime of the system (resource, site, datacenter etc).

In the ORBIT project [4][5] several protection policies for VM images and for persistent storage were implemented, including point in time snapshot of instances/volumes, image backup and volume replication:

- VM snapshot takes a snapshot of the local VM ephemeral disk and creates a copy that will be used for recovery. In this way, an instance is restored from a point in time. This is useful when patches are regularly applied to the VM, so that the snapshot captures the updated version, which includes the latest patches. Note that taking the snapshot is an intrusive method that, even though it may save some extra information about the status of the VM, it will impact the performance of the VM and the hosted applications. In addition, it will make the overall protection process longer than Image copy.
- Image copy action provides the option to start a fresh instance from a backup image, i.e. not from a given state, using the base image that was used to create the VM (a one-time backup operation). The "protect" and "recover" logic are the same as in VM snapshot, except that a snapshot of the instance is not taken and not exported to the destination, so that only the current base-image is sent.
- Volume snapshot action is used to take a point-in-time copy of a volume on the primary site and save it on the secondary site. Again, as for the image snapshot, a certain amount of time is needed to take the snapshot. The "recovery" action recreates the volume from the snapshot on the secondary site, and associates it with the respective VM instance.
- Volume replication action keeps a continuous mirror of the volume on the secondary site. The "protection" logic enables the replication mechanism and exports the volume's metadata, while the "recovery" logic imports the volume to the secondary site and connects it to the appropriate VM. Volume replication minimizes the lost data in case of a failure compared to volume snapshot, while it also avoids big chunks of data to be sent during the protection actions, since the volume is already replicated. What is more, the recovery process is remarkably faster since the volume in question, just needs to be imported into OpenStack, unlike the snapshot case where it also needs to be recreated, which takes a potentially long amount of time based on the size of the volume (s).

In our work, we have created a simple driver that performs synchronization actions with a fixed frequency, using DR-Engine protection API. More advanced drivers can also be implemented, for instance triggering synchronization actions when a certain amount of data has been modified, and/or actuating on the bandwidth ratio dedicated per traffic flow, in order to differentiate the replication traffic for image copy, image snapshot, volume snapshot and image metadata, as well as from the application and other traffic. However, in any case, it is up to the protection policies to minimize the amount of replication data sent and the interference caused to the rest of the datacenter traffic, for example by distributing over time the transfer of replication data instead of sending them all together, which may lead to network bottlenecks. Also, the various protection/recovery actions can be grouped into "protection workload policies", where all the actions defined in a policy are executed in order to accomplish the desired behavior.

4 DRL in OpenStack

4.1 OpenStack Internal and Traffic Flows

OpenStack is the most known platform for building an Infrastructure as a Service that is a cloud. Various OpenStack deployments can be setup, based on the users' needs, by selecting appropriately the respective OpenStack components. A minimal OpenStack configuration (Figure 3) requires, at least one controller, with the following components [3]: Keystone, for authentication and authorization; Nova, for managing the VMs operation (create, migrate, delete); Cinder, for managing the persistent block storage (Volumes); Neutron, for managing virtual network creation and configuration; Glance, for VMs images storage; and Swift, for object storage (similar to Amazon S3). In addition to the previous main components, there are a few more that are usually very convenient to have, such as Ceilometer (billing and monitoring service), Horizon (dashboard to manage projects, users and resources), and Heat (orchestration service -- template based). Note that each component of the controller node can be deployed in a different server, and most components can be replicated for high availability purposes within the same datacenter. Besides the controller node(s), there may be as many computing nodes (i.e., workers) as desired, which just need a minimal configuration, mainly regarding Nova and networking agents.

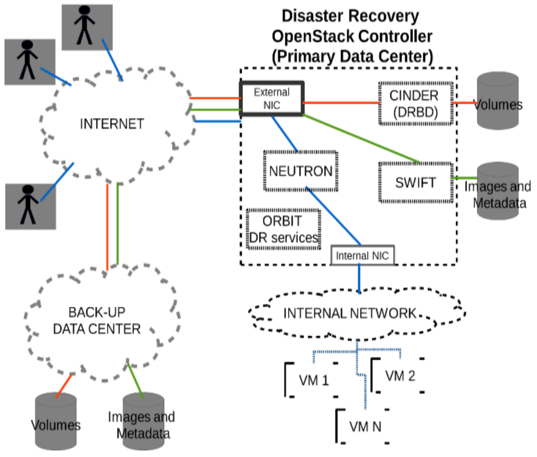


Figure 3: Disaster Recovery Layer (DRL) in OpenStack.

Also, for the replication purposes, it is important to keep in mind the inbound/outbound network traffic flows of the basic OpenStack deployments. The internal communication between VMs allocated in the same virtual network is performed directly between the hosts containing the VMs, while the communication between different VMs allocated in different virtual networks goes through the Neutron component (i.e., through the networking server). Nevertheless, this traffic only uses the internal NIC as the traffic does not go out of the datacenter facilities. In most recent versions this communication can be directly redirected to the destination host if Distributed Virtual Routing (DVR) is used. By contrast, when the VMs communicate with external entities, e.g., for replication, the traffic goes from the server hosting the VM, to (one of) the Neutron server (internal NIC), and from there it is redirected to the external NIC of the Neutron server, and sent to the exterior (see Figure 3). The opposite procedure is performed for the incoming traffic. As all the outgoing traffic, including normal datacenter operation traffic and replication traffic, passes through this external NIC at the Neutron server(s), the replication traffic overhead needs to be minimized so that the operational part of the datacenter traffic is affected as little as possible. This is also needed in order to ensure that replication data is sent to the backup datacenter with minimum delay.

4.2 DRL Openstack integration

The final ORBIT project demo [4][5] illustrated the complete integration of the DR-Orchestrator and DR-Engine with OpenStack (Figure 3), and the Fault Detection framework (Section 5). The integration with OpenStack is based on the use of the available REST APIs and OpenStack's project design principles. In particular, for the DR-Orchestrator, three different OpenStack components were extended: Horizon, Python-novaclient, and Nova. For setting up the DRL environment, the administrator starts DR-Orchestrator and DR-Engine on both primary and the backup OpenStack-based datacenters as an additional OpenStack service. DRL is registered as a service in the OpenStack system (through Keystone) and it can access the list of resources (instances, volumes, security groups, public keys) of each tenant accessible by the admin user using OpenStack's APIs.

The VM protection mode (image copy or VM snapshot) is based on the administrator configuration, while the volume one is based on the volume type. That is Volume snapshot is used for ISCSI volumes and volume replication for DRBD (Distributed Replicated Block Device) [30] volumes. For the latter, we employ the replication structure and paradigm provided by the DRBD tool [30] to enable continuous yet asynchronous replication. Of course, DRBD can operate both synchronously and asynchronously. However, as synchronous replication is not feasible at MAN/WAN or inter-datacenter network scale, due to the generally long network delays, we decided to choose the DRBD replication protocol A. This performs the replication in a continuous but asynchronous manner: write operations in the local virtual devices are considered completed as soon as they are copied in the local disk and the replication packet is placed in the TCP send buffer, as opposed to waiting for the packet to be sent and acknowledged. It must also be noted that in cases where the bandwidth available for this replication traffic is not enough to keep up with the rate of write operations, the default behavior for this replication protocol is to block the I/O system when the local TCP send buffer is full.

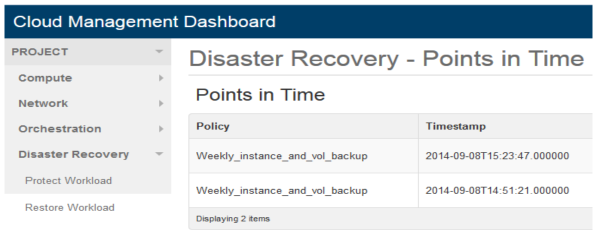
The protection actions kick in periodically with protected data being transferred in the backup datacenter. In particular, in the Instance/VM snapshot the "protection" logic creates a snapshot of the instance and exports the cloned data and instance metadata to a remote Swift Object Store. Conversely, the "recovery" logic imports the snapshot data and metadata from Swift into a new Nova instance and then invokes the restoration of the instance from Swift on the backup datacenter. Also, the protect action results in a Heat Orchestration Template (HOT) being produced, which captures all the user's resources and characterizes their relationships so that the application can be resumed on the secondary site. The HOT template includes the OpenStack identifiers of various resources, such as VM instance ID, volume ID, network entity ID, etc. However, since these identifiers refer to the primary OpenStack cloud-site, they need to appropriately mapped when the HOT template is reproduced on the secondary cloud-site. This is accomplished by preprocessing the HOT template on the secondary site, in order to plug the proper values of the IDs before passing the HOT template to Heat for recreating and deploying the specified resources (VM, attached volume, network, etc.). The DR-Engine is responsible for managing and updating the HOT templates.

Generally, upon failure of the primary site, restoration actions associated to each protected resource (i.e., group of VMs and volumes) are employed in the backup site. Each protected VM is also booted with the same configuration as in the primary site (volumes attached and both internal and floating IPs) and soon the VM connectivity is restored, while keeping all the original data and having applications work normally. The recovery scenario requires authenticated access to the backup site. During restore, the metadata of the protected resources are also loaded and the restore steps for each action associated to a resource are invoked. Finally, the deployment of the protected workload is triggered through OpenStack Heat and old recovery data are cleaned up.

The DR-Orchestrator and the DR-Engine are available as open source software [35][36].

4.3 DRL Usage

Users can interact with DRL through the provided REST API, the python client extension and the Horizon UI (Figure 4), allowing easy protection and recovery of workloads.



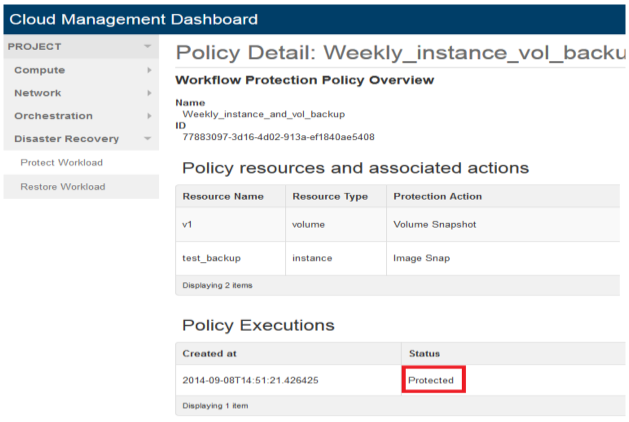
Disaster Recovery - Points in Time

Points in Time

Policy	Timestamp
Weekly_instance_and_vol_backup	2014-09-08T15:23:47.000000
Weekly_instance_and_vol_backup	2014-09-08T14:51:21.000000

Displaying 2 items

(a)



Policy Detail: Weekly_instance_vol_backu

Workflow Protection Policy Overview

Name
Weekly_instance_and_vol_backup
ID
77883097-3d16-4d02-913a-ef1840ae5408

Policy resources and associated actions

Resource Name	Resource Type	Protection Action
v1	volume	Volume Snapshot
test_backup	instance	Image Snap

Displaying 2 items

Policy Executions

Created at	Status
2014-09-08T14:51:21.426425	Protected

Displaying 1 item

(b)

Figure 4: DR panels integrated with OpenStack Horizon.

Through the Horizon UI, the user is able to select the protection and the recovery policy for a newly created VM and Volume or for an existing one. Similar functionality can be achieved through the command line, e.g., using the commands “nova protect-vm VM_ID|VM_NAME” and “nova protect-volume VOLUME_ID”. Next, the DR-Orchestrator includes the resources to be protected in a workload policy and uses the plug-ins available from the DR-Engine, for Instance/VM snapshot, Image copy, Volume snapshot and Volume replication. The workload policy is the “bag” where VMs and volumes that need to be protected are included, so that periodic protection actions are performed over them. Generally, an admin user is able to switch to a specific tenant and create one or more workload policies. It is also possible to create more protection policies through the Horizon UI, or update the existing ones that are created by default upon the DR-Orchestrator initiation or add more resources under protection. An admin user can also manually trigger a workload protection policy by pressing the “Trigger Policy” button on the “Protect Workload” screen of Horizon UI and check the status of execution of the protect tasks for each resource. Horizon UI also presents the protected data, which have been transferred in the backup datacenter. Moreover, it is possible to track the recovery progress using Horizon Heat panels as well as to recover a particular resource “manually” through the UI, even without a failure having occurred.

5 Failure Detection Framework

A failure detection framework was also implemented to identify when a datacenter fails due to a catastrophe or a network connectivity issue. The framework is based on the idea of Indicators and Pingers that advertise, from inside the datacenter, and learn, on behalf of a second datacenter, the health of the first. In the end, there are the Voters that actually decide on a datacenter status using information received from multiple pingers. These along with the orchestration entity called Updater, are the main components of the failure detection framework (Figure 5).

The Updater uses the framework’s database to update, initially and periodically, its Indicators, Pingers and Voters list. The database holds configuration data (timeouts, update intervals, voting thresholds), connectivity properties (IP, port, database url) and the lists of available Indicators, Pingers and Voters. The Indicators of a datacenter’s status can be physical servers or virtual machines (VMs), which are queried for liveness, using periodic UDP ping messages. Each Pinger queries a pre-specified set of Indicators at periodic intervals, while also acknowledging intermittent networking delays and errors, and retries pings after a number of failures. If all the Indicators are non-pingable, the Pinger sends a datacenter down signal to the paired Voter. Next, the Voter starts the opinion collection phase, exploiting the exposed REST API and requesting the opinion on the status of the target datacenter from other partner Voters. The status (up or down) of the datacenter is determined based on the majority of the replies from collaborating Voters and assuming that half of the requested Voters responded. The opinion collection and voting is retried up to a maximum number of attempts. If the targeted datacenter is down then the failure detection framework uses the DR-Orchestrator’s API to notify the backup datacenter of the disaster.

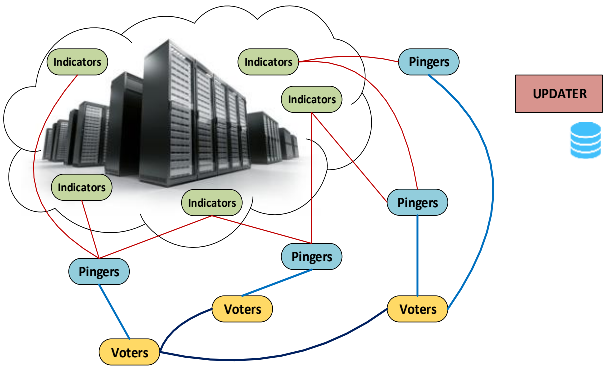


Figure 5: The Failure Detection framework.

The performance of the failure detection framework is characterized by the time it takes to identify a disaster in a datacenter and by the false positives. Disaster detection delay can be controlled by reducing the timeouts in both Pingers and Voters, increasing on the other hand the induced network and processing overhead. The interval of updating the available Pingers and Voters lists is also important, since a large timing interval can give an erroneous picture of the available failure detection entities. In any case, one has to experiment with the various interval values, so as to determine the "proper" ones for each setting (multi-site scenario). For this purpose, the failure detection framework is fully configurable. Also, the external false positives count the number of calls made to an Indicator to check whether a datacenter is down (under the caller’s belief that it is actually down), when the response provides a different indication (datacenter is up). The internal false positives count the number of votings initiated by failed pings, when the majority of the Voters declare the datacenter as a working one. Other metrics of interest are the number of retries until a datacenter is determined as up (during

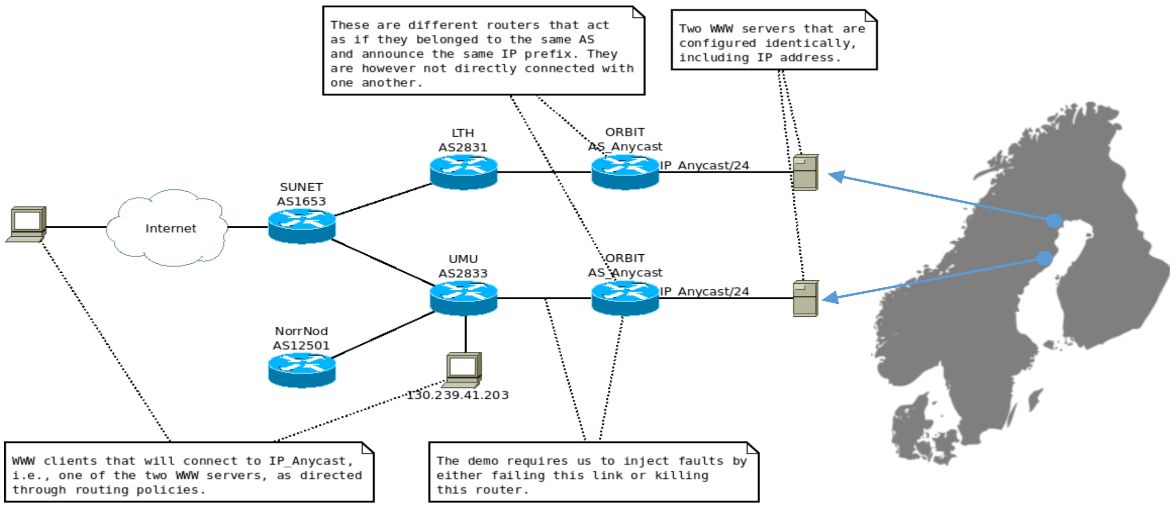

These are different routers that act as if they belonged to the same AS and announce the same IP prefix. They are however not directly connected with one another.
Two WWW servers that are configured identically, including IP address.
LTH AS2831
ORBIT AS_Anycast
IP_Anycast/24
SUNET AS1653
Internet
UMU AS2833
ORBIT AS_Anycast
IP_Anycast/24
NorrNod AS12501
130.239.41.203
WWW clients that will connect to IP_Anycast, i.e., one of the two WWW servers, as directed through routing policies.
The demo requires us to inject faults by either failing this link or killing this router.

Figure 6: The DRL testbed in Umeå and Luleå Universities and the network setup for traffic redirection

a ping) and the number of undecided votes until the majority declares the datacenter as down in case of a correct indication. The time that elapses between an initial indication of a datacenter down until the final decision from the Voters is made, is also important. Typical failure detection delays for the implemented testbed, have been measured to fall within the 500 msec range, since 5*100 msec is the resulting detection delay for 5 ping retries, of 100 msec period.

The failure detection framework is available as open source software [37]. In our implementation, we chose to use the Go programming language that is efficient and suitable for server tasks.

6 Testbed

The DRL testbed is a multi-site deployment, consisting of two, an active and a backup, installations/datacenters located in Umeå and Luleå, respectively (Figure 6). The two datacenters were 265km apart, connected over a two-hop network between each end-point and the core/common router/node. The core network of SUNET, the Swedish national research and education network, was utilized for interconnecting the datacenters.

6.1 Hardware and Software setup

The primary testbed installation, located in Umeå, is configured to host the applications and provide the protection mechanisms described in previous sections, in order to allow applications' (and their data) recovery in the secondary datacenter, located in Luleå, after a failure. This primary testbed, at Umeå University, consists of 6 identical servers, each with 2x6 core Intel Xeon E5-2620 (Sandy Bridge) at 2.1GHz, 64GB DDR3-1600MHz, 2x2TB HGST SATA HDD, 2x120GB Samsung SM843T Enterprise SSD. On the other hand, the secondary testbed, at Luleå University, consists of 3 identical servers (HP DL320e), each with 4 cores Intel Xeon E3-1220-v2 at 3.1GHz, 32GB RAM and 2x2TB HGST SATA HDD.

Both testbed installations run CentOS 7.0 and a complete installation of the OpenStack software stack, along with the respective DRL components and OpenStack extensions. Also, in each testbed one of the servers is configured as the controller (including all the components, such as Neutron, Cinder, Keystone, Glance, etc) and the rest are configured just as workers.

6.2 Traffic Redirection

Traffic redirection mechanisms are necessary so as to steer traffic from the original datacenter to the backup after a failure. Several traffic redirection mechanisms are available today [11]. In our testbed, we applied the BGP Anycast scheme (IETF RFC 4786), as shown in Figure 6. Anycast has in recent years become increasingly popular for adding redundancy and load balancing to the provided services (e.g., DNS and Content Distribution Networks [31][32]). In [12], the authors apply BGP Anycast for network recovery in case of a VM failure.

Anycast actually performs traffic redirection when a server fails, simply by letting the routing system do its work, i.e. routing. Anycast is a communication paradigm where a group of distinct nodes, probably in different locations, share the same "Service Address" and the responsible routing system forwards packets with that destination "Service Address" to the "nearest" node of the group. IP anycast is an anycast service implemented at the network layer, where the "Service Address" is the IP address and the routing protocol is one of the Interior Gateway Protocol (IGP) or the Border Gateway Protocol (BGP), depending on the scale through which IP is anycasted (IETF RFC 4786, RFC 7094). BGP anycast requires that both domains belong to the same Autonomous System (AS) and the active and backup VMs or servers have the same IP address. The respective routers advertise the same IP prefix to their neighbors. As the routing information flows from two different directions, it ends up at a common node (e.g. core router). In this way redundant information about the same subnet is available, gathered in the router's Routing Information Base (RIB). According to BGP's best path selection algorithm, the most preferable route is used to forward traffic according to Forwarding Information Base (FIB), while the alternative path is also stored. If the primary path fails then the alternative path is preferred, giving redundancy to our network.

Regarding, the efficiency of BGP Anycast, according to [33], the BGP convergence delay increases rapidly with the magnitude of failures before levelling off and going down. This means that multiple failures can lead to much longer periods of instability when compared to single failures, considerably affecting the applications' connectivity and the user experience. Furthermore, the convergence delay scales with the inter-AS distance between the client and the cloud/datacenter provider.

The active and backup testbed installations, located in Umeå and Luleå, connect to two different routers that have been setup as if they belong to the same Autonomous System and announce the same IP prefix; however, they are not directly connected to each other (Figure 6). Also, in our DRL testbed the BGP anycast scheme redirects traffic for the original IP address to the new VM location. For this reason, we extended OpenStack in order to restore the original public (floating) IP addresses to the VMs when they run in the secondary cloud. Note that the private IPs are also restored as in the primary datacenter so that the VMs are able to communicate internally, in the same way they were doing so in the primary datacenter before the disaster occurred.

6.3 Failure initiation

In order to emulate a datacenter failure, without actually having to wait for a real one to occur, we have created a simple script that:

- Stops the BGP daemon at Umeå testbed so that the users cannot get access to the VMs located in the primary datacenters.
- Blocks the network traffic between Umeå and Luleå through IPTables rules, so that no data can be replicated/backed up in the secondary datacenter. This, besides blocking any ongoing transfer between the two datacenters, including possible ongoing VM/Volume snapshots and/or VM image copies, it also forces the DRBD volumes in the secondary datacenter to lose connection with the primary ones, so that they do not update their data from that point in time.

7 Results

The testbed described in Section 6 was used to evaluate the performance of the final integration of all DRL components, while also comparing the implemented protection and recovery policies.

7.1 Protection performance for various volume sizes

Initially, we evaluated the protection performance, in terms of the time it takes (duration) to update the data that the secondary datacenter holds for the protected VMs and volumes running in the primary datacenter. Figure 7 presents a comparison of this duration, for the protection policies implemented and integrated in DRL, namely, snapshotting (for both VM images and volumes) and replication (i.e., image-copy and DRBD-based volume-replication). In the experiments performed the VM image size was less than 200MB, while we considered various volume sizes, ranging from 1GB to 10 GB. The protection actions were triggered by the DR-Orchestrator every 5 minutes and the total duration of each experiment was 30 minutes, while obtaining the average time required for each action.

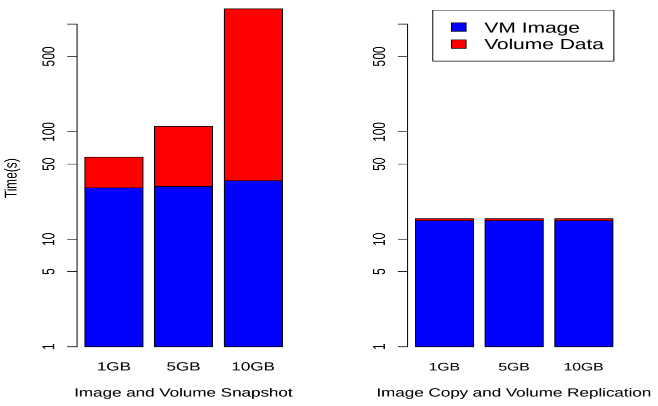


Figure 7: Protection time for snapshot-based and for replication-based protection policies, considering various volume sizes (1GB, 5GB, 10GB).

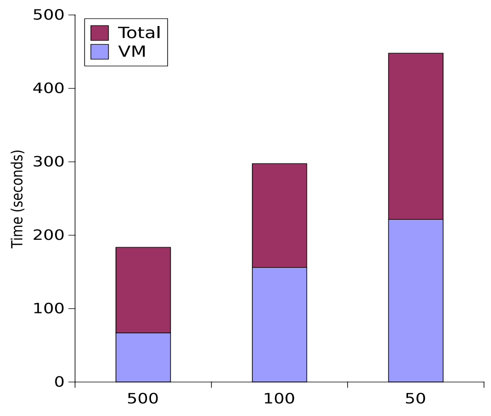
As can be seen in Figure 7, the VM image-copy and the volume replication protection policies always perform better than the respective ones that are based on snapshots. In particular, VM image copy presents a smaller transfer time compared to VM snapshot, as it just needs to send the current base image, and thus saves time by not having to take periodic snapshots that also need to be sent over the wire. This performance benefit is enhanced as the size of the VM increases, due to the overhead of taking the snapshot, which also impacts the performance of the VM as a whole and consequently of the applications running inside.

For the volume replication policy, as the data replication is happening continuously, its resulting performance is a lot better than that of the volume snapshot policy (note the log scale of the plots). Also, the volume replication method, unlike the snapshot policy, is not affected by the size of the Volume (1, 5, or 10 GB), as it can be seen from the constant protection time (negligible time – red bar). Again, the reason is that for volume replication, the replication is happening continuously, before the actual "protection" action is invoked and it only replicates data when VMs are writing into the volumes, regardless of the size of the volumes. Thus, if no data is written into the volume during a period of time, no data is sent at all, unlike the snapshot technique that needs to take a new snapshot of the volume and send it. This means that at any point in time, the secondary testbed site already has the most up-to-date data, and therefore, there is no waiting time (i.e., transfer time) when the protection actions are triggered. This also minimizes the amount of data that is lost in case of a sudden failure. In contrast, for the snapshot case, as the size of the volume increases, the time needed to create and ship the snapshot data also increases. If a failure occurs during the transfer period, all the data between protection actions will be lost. In fact, as the snapshot protection action is triggered automatically and periodically, it is possible that the previous actions are not even completed before the new one starts, and so they may overlap with it. This in turn increases the amount of data loss, since an older snapshot needs to be used for the recovery.

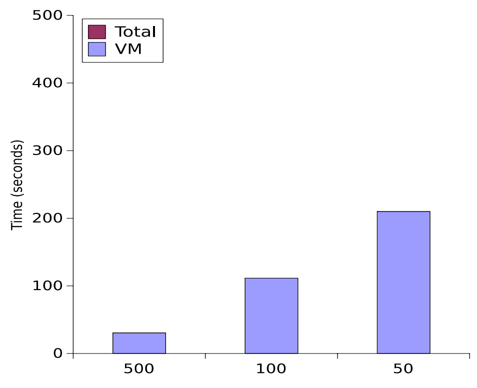
7.2 Protection performance for various network bandwidth scenarios

Because of the benefits that image-copy and volume replication drivers provide, the protection mechanisms can better deal with network capacity shortages, providing higher performance when the available bandwidth is limited. To test this scenario, we have made use of a video transcoder application, running as a VM with a volume (virtual disk) attached. This application is a media application that fetches videos (in this case, 50 videos of size between 20 and 100 Mbytes each) from a remote repository, stores them in the volume, performs some transcoding actions to adapt it to different formats/sizes, and stores the transcoded videos back in the volume. In this scenario, volume data are continuously modified, resulting in large amounts of data to be replicated, both continuously for the replication, and periodically for the snapshot protection policies. To limit the (inter-datacenter) bandwidth between the two sites, we used the Linux Traffic Control tools (TC) to create a QDISC through which all the traffic from primary to secondary datacenter will be sent, and then limit the maximum bandwidth for the QDISC itself.

As Figure 8 shows, we have tested the protection performance by limiting the bandwidth perceived by the primary testbed to 500Mbps, 100Mbps, and 50Mbps.



(a)



(b)

Figure 8: Protection time (a) for snapshot-based protection policies and (b) for replication-based protection, in various network bandwidth scenarios (500Mbps, 100Mbps, 50 Mbps).

As expected, when the bandwidth decreases, the time it takes to perform the protection actions increases, while the impact of the limited bandwidth on the image copy protection policy is lower than the corresponding impact on the image snapshot policy. The figure also highlights that there is no extra overhead for the volume protection when using the replication driver, since volume replication is performed continously. This is the reason why there is no difference between VM protection and total time protection in Figure 8. In contrast, when volume snapshot is used, then the protection actions lead to the volume being copied, and this takes longer as the bandwidth between the primary and the backup datacenters decreases, making the performance differences between protection actions more profound. In particular, in the 50 Mbps scenario the replication based policies require half the time compared to the snapshot based ones.

Figure 9 shows the cumulative outgoing network traffic between the primary and backup datacenters, for the various protection policies (replication and snapshot) considered and different values for the available network bandwidth (500Mbps, 100Mbps, 50 Mbps). We observe that by using the volume replication driver, data are continuously transferred from the primary to the backup datacenters, and not only at protection actions periods. Therefore, the available bandwidth is better utilized, and more data are sent over the same time period. The graphs also illustrate large jumps coinciding with the protection actions, indicating times where suddenly there are more data to be sent. This is due to the image-copy, for the replication policy, and due to both image and volume snapshot, for the snapshotting policy. However, as the available bandwidth decreases, the slope observed during the protection action makespan becomes smaller and smaller, meaning that it takes more time to send the same amount of data. Moreover, in the lowest bandwidth scenario (50 Mbps) and when the replication protection policy is used, we can observe flat periods of time, which indicate that the volume has already replicated its updated data and there is still some time left between replication actions where no data have to be sent. By contrast, when using snapshotting (and without considering the initial part, before the first protection action), there is always some data that need to be sent, as protection actions are overlapping due to the larger amount of data replicated and the lower available bandwidth, both of which increase the transfer time.

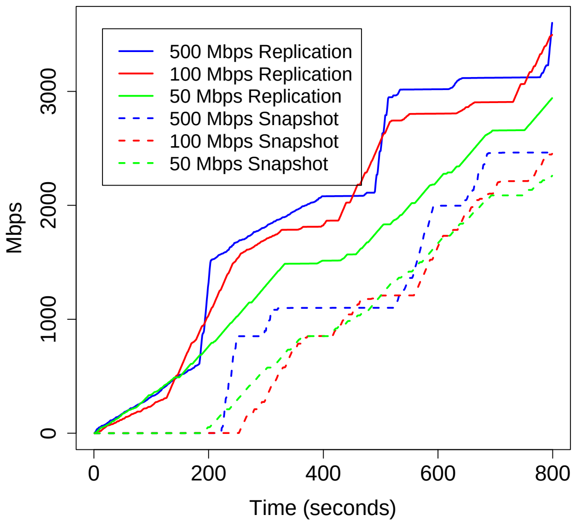


Figure 9: The cumulative outgoing network traffic between primary and backup datacenters, for the different protection policies (replication vs snapshot) and available network bandwidth scenarios (500Mbps, 100Mbps, 50 Mbps).

Figure 10 shows the same information as Figure 9 but in almost continuous time, by averaging data for every 10 seconds. For the replication policy, we observe that there is some background traffic in addition to that generated by the protection action (in this case due to the image-copy data). Then, at the protection action less data needs to be sent, resulting in the protection action being completed in a shorter period of time. In addition, the average bandwidth required for the replication policy is higher since the VM image needs to be copied. For the snapshot policy, more data needs to be sent as no protection/update data is sent between protection actions. Moreover, two peaks are observed for each protection action, one for the VM snapshot and the other for the volume snapshot. As described before, taking snapshot requires time, making the average throughput for that period not as high as the one achieved by the replication, due to the extra work that needs to be performed before sending the data. Figure 10.b illustrates the same trend when a lower inter-datacenter bandwidth of 100 Mbps is available (instead of 500 Mbps as in Figure 10.a), in which case the protection action takes longer for both policies. Finally, Figure 10.c presents the case where network bandwidth is only 50Mbps, thus becoming the bottleneck of the protection operation, with the protection actions taking longer and longer. To better highlight what is illustrated in that figure, we have annotated it with information about the times at which protection actions start and finish for each policy (blue for replication and red for snapshot). Generally, when the available bandwidth is limited and/or the amount of data to be transferred is very large, the snapshot technique starts exhibiting problems in completing the protection operations before the start of the next protection action, leading to heavier network contention, even longer protection action times, and consequently to a higher risk of losing valuable data in case of a sudden datacenter-site failure. On the other hand, the volume replication policy does not have any protection action overlapping issues, even at low available inter-datacenter bandwidth, as it sends data continuously instead of doing so at fixed periodic intervals.

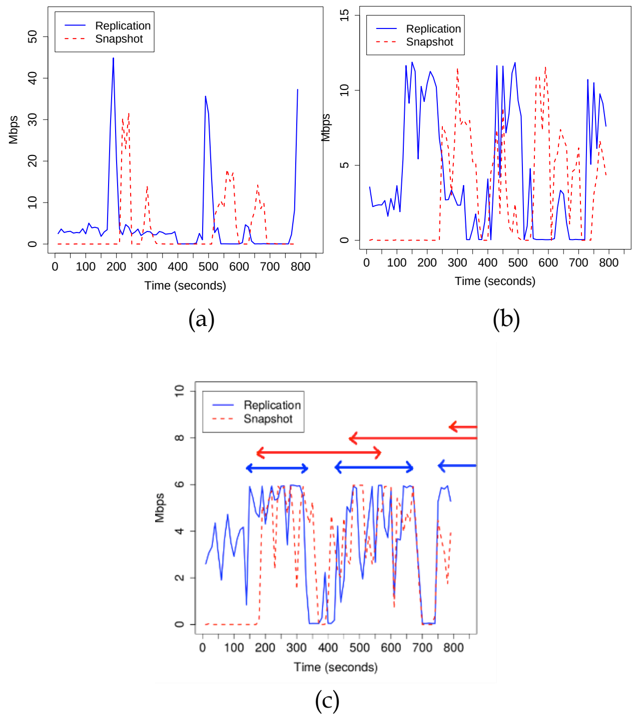


Figure 10: The continuous time outgoing network traffic between primary and backup datacenters, for the different protection policies (replication vs snapshot) and available network bandwidth scenarios: (a) 500Mbps, (b) 100Mbps, (c) 50 Mbps.

Generally, we should note that when the available outgoing bandwidth is limited, there is an inherent trade-off between the network resources used for data consistency through the replication of changes to the backup datacenter and the network resources used by the applications hosted in the VMs, e.g., for responding to user requests. This is primarily a matter of cost and introduces inherent conflicting goals, since the Quality of Service (QoS) of an application is typically directly related to the rate at which it can serve requests from clients. At the same time, the degree to which the application remains disaster tolerant is subject to what rate application write operations can be mirrored at the remote replica and how expediently the transfer of VM images and related meta-data can be completed. By not considering this tradeoff, high congestion situations can lead to unacceptable service degradation or disaster tolerance.

7.3 Recovery performance

We also evaluated and compared the time needed to recover an application and its associated data (volume) when a failure occurs, considering different volume sizes.

In particular, Figure 11 illustrates for various scenarios the failure detection time, the time required till the recovery operations start (the DR-Orhcestrator is informed by the failure detection mechanism) and the time need for the recovery operations to complete and the respective VM to become responsive in the backup-site. As it can be seen in Figure 11, image-copy and volume-replication policies improve remarkably the recovery times in comparison to those exhibited by the respective snapshot policies. Also, for all cases the performance of the failure detection mechanism and the trigger of the recovery actions remain roughly the same, while the effect of traffic redirection is negligible. The recovery times are much longer for the snapshotting policies, as the volume needs to be recreated as the backup site, while for DRBD volumes (in which volume replication is applied) it just needs to be imported into OpenStack, namely into Cinder. This leads to a significant reduction in the overall recovery times. Also, for the replication policy (i.e., image-copy and volume replication), we observe that the recovery times are around half a minute, regardless of the volume size. In contrast, for the snapshot policy, the recovery time increases notably with the size of the volume, leading to downtimes of more than 200 seconds for just 5 GB of recovery data.

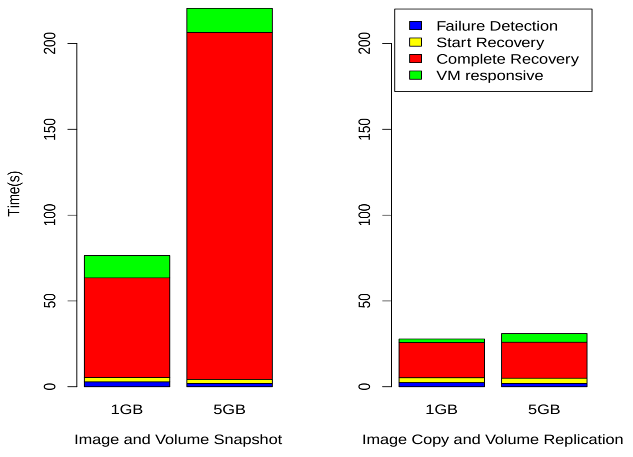


Figure 11: Recovery time performance for snapshot-based and for replication-based protection policies, considering various volume sizes (1GB, 5GB).

All in all, the next two behaviors are clearly appreciated for volume-replication:

- Recovery times remain constant, independently of the volume size.
- Recovery times are much smaller than when using snapshotting policies, while the difference in performance becomes larger as the volume size increases.

Note that for the purpose of fair and simple comparisons, we have used a small VM that does not require a lot of time to boot, allowing us to focus on the time required to restore the VM's data. In any case, the VM's boot up time is simply added to the overall recovery time and it is the same for all scenarios and policies considered.

8 Conclusions

In this work, we presented the Disaster Recovery Layer (DRL) for OpenStack-based datacenters, enabling the business continuity as a service concept [4][5]. The DRL scheme being implemented and integrated with OpenStack, is extensible by design so as to easily incorporate new protection, restoration and orchestration policies. Two protection policies for VM images and two protection policies for persistent storage (volumes) were implemented, using either image/volume snapshot or image copy and volume replication. Replications policies are based on the continuous transfer of update data between the primary and backup datacenters. A distributed failure detection mechanism was also implemented, based on the notion of voters that decide on the status of a datacenter using information received from multiple pingers. DRL was evaluated in a real-world testbed consisting of an active and a backup datacenter located 265km apart. The BGP anycast scheme was also setup in the testbed, for the transparent traffic redirection between the two datacenters. A number of experiments were performed, showing that DRL can efficiently protect the workloads (VMs and volumes) with minimum service disruption in case of failures and low overhead, even though the two sites were connected over a long distance public, inter-datacenter network. Our experiments show that the replication policies outperform the snapshot based methods, in terms of protection and recovery times, while also making efficient use of the available bandwidth.

Acknowledgment

This work has been supported by the European Commission through the FP7-ORBIT project under grant agreement n^{o} 609828.

References

[1] Amazon, "Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region", http://aws.amazon.com/message/65648

[2] L. Wang, H. Ramasamy, R. Harper, M. Viswanathan, E. Plattier, "Experiences with Building Disaster Recovery for Enterprise-Class Clouds", Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 231-238, 2015.

[3] OpenStack, https://www.openstack.org/

[4] ORBIT, EU FP7 project, http://www.orbitproject.eu/

[5] D. Kyriazis, V. Anagnostopoulos, A. Arcangeli, D. Gilbert, D. Kalogeras, R. Kat, C. Klein, P. Kokkinos, Y. Kuperman, J. Nider, P. Svärd, L. Tomas, E. Varvarigos, T. Varvarigou, "High performance fault-tolerance for clouds", IEEE Symposium on Computers and Communication (ISCC), pp. 251-257, 2015.

[6] C. Oberg, A. Whitt, R. Mills. "Disasters will happen-are you ready?", IEEE Communications Magazine, Vol 1, No. 49, pp. 36-42, 2011.

[7] T.Costello, "Business Continuity: Beyond Disaster Recovery", Journal IT Professional, Vol. 14, No. 5, 2012.

[8] EVault Cloud Disaster Recovery, http://www.seagate.com/files/www-content/services-software/cloud-resiliency-services/_shared/masters/docs/wp-cloud-disaster-recovery-ready-for-midmarket-2014-09-0019-w-us.pdf

[9] T. Wood, E. Cecchet, K. Ramakrishnan, P. Shenoy, J. Van Der Merwe, A. Venkataramani, "Disaster recovery as a cloud service:

Economic benefits & deployment challenges", 2nd USENIX workshop on hot topics in cloud computing, pp. 1-7, 2010.

[10] A. Khoshkholghi, A. Abdullah, R. Latip, S. Subramaniam, M. Othman, "Disaster recovery in cloud computing: a survey", Computer and Information Science, Vol. 7, No. 4, 2014.

[11] P. Kokkinos, D. Kalogeras, A.Levin, E. Varvarigos, "Live Migration and Disaster Recovery over Long Distance Networks", ACM Computing Surveys, 2016.

[12] S. Rajagopalan et al, "SecondSite: Disaster Tolerance As a Service", ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments, pp. 97—108, 2012.

[13] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: high availability via asynchronous virtual machine replication", USENIX Symposium on Networked Systems Design and Implementation, pp. 161-174, 2008.

[14] T. Wood, H. Lagar-Cavilla, K. Ramakrishnan, P. Shenoy and J. Van der Merwe, "PipeCloud: Using causality to overcome speed-of-light delays in cloud-based disaster recovery", SoCC, 2011.

[15] A. Lenk, S. Tai, "Cloud standby: disaster recovery of distributed systems in the cloud", In Service-Oriented and Cloud Computing, pp. 32-46, 2014.

[16] O. Alhazmi, Y. Malaiya, "Evaluating disaster recovery plans using the cloud", IEEE Reliability and Maintainability Symposium", 2013.

[17] R. Couto, S. Secci, M. Campista, L. Costa, "Network design requirements for disaster resilience in IaaS clouds", IEEE Communications Magazine, Vol. 52, No. 10, pp. 52-58, 2014.

[18] Forbes, "The Big Bang: How the Cloud Is Changing Resilience in the Expanding Universe of Digital Data", http://www.forbes.com/forbesinsights/ibm_big_bang/index.html, last seen October 2015.

[19] Backup and Disaster Recovery Buyers Guide, http://solutions-review.com/backup-disaster-recovery/get-a-free-backup-and-disaster-recovery-buyers-guide/, last seen September 2015.

[20] Magic Quadrant for Enterprise Backup Software and Integrated Appliances, https://www.gartner.com/doc/3074822/magic-quadrant-enterprise-backup-software, last seen September 2015.

[21] Gartner, Magic Quadrant Disaster Recovery as a Service – 2015, https://www.gartner.com/doc/3033519/magic-quadrant-disaster-recovery-service, last seen September 2015.

[22] J. B. Leners, H. Wu, W.-L. Hung, M. K. Aguilera, and M. Walfish. "Detecting failures in distributed systems with the falcon spy network", ACM Symposium on Operating Systems Principles, pp. 279-294. 2011.

[23] Q. Guan, S. Fu. "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures", IEEE 32nd International Symposium on Reliable Distributed Systems, pp. 205-214, 2013.

[24] Magic Quadrant for WAN Optimization, https://www.gartner.com/doc/3008618/magic-quadrant-wan-optimization, last seen September 2015.

[25] T. Kang, M. Tsugawa, J. Fortes, T. Hirofuchi, "Reducing the migration times of multiple VMs on WANs using a feedback controller", IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp. 1480-1489, 2013.

[26] T. S. Kang, M. Tsugawa, A. Matsunaga, T. Hirofuchi, J. A. Fortes, "Design and Implementation of Middleware for Cloud Disaster Recovery via Virtual Machine Migration Management", IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp. 166-175, 2014.

[27] R. Couto, S. Secci, M. Campista, L. Costa,"Server placement with shared backups for disaster-resilient clouds", Computer Networks, 2015.

[28] V. Gramoli, G. Jourjon, O. Mehani, "Disaster-Tolerant Storage with SDN", The International Conference on Networked Systems, 2015.

[29] V. Gramoli, G. Jourjon, O. Mehani, "Can SDN Mitigate Disasters?", arXiv:1410.4296

[30] DRBD, https://www.drbd.org/en/

[31] Anycast, CloudFlare blog, http://blog.cloudflare.com/a-brief-anycast-primer

[32] Load Balancing without Load Balancers, CloudFlare blog, http://blog.cloudflare.com/cloudflares-architecture-eliminating-single-p/

[33] A. Sahoo, K. Kant, P. Mohapatra, "BGP Convergence Delay under Large-Scale Failures: Characterization and Solutions", Computer Communications, Vol. 32, No 7, pp, 1207-1218, 2009.

[34] J. Dürango, W. Tärneberg, L. Tomás, J. Tordsson, M. Kihl, M. Maggio. "A control theoretical approach to non-intrusive geo-replication for cloud services". 55th IEEE Conference on Decision and Control (CDC), 2016

[35] DR-Orchestration github repository, https://github.com/orbitfp7/nova/tree/DR-Orchestration

[36] DR-Engine github repository: https://developer.ibm.com/open/disaster-recovery-for-cloud-dragon/

[37] Failure detection framework: https://github.com/fithisux/orbit-dc-protector

[38] Flame, https://github.com/openstack/flame

[39] Galaxy, https://www.openstack.org/videos/video/protecting-the-galaxy-multi-region-disaster-recovery-with-openstack-and-ceph