

VIRTUALIZATION AS A MEANS TO ENSURE THE RESILIENCE OF PEER-TO-PEER COMMUNICATION SYSTEMS

Valentin PAU¹
Dorina Luminița COPACI²

Rezumat. *Resilience to failures and deliberate attacks is becoming an essential requirement in most point-to-point communication networks today. The present paper presents a survey of strategies to ensure resilience in peer-to-peer communication networks by means of operating systems virtualization. Virtual machines can ensure the resiliency of the peer-to-peer communication network both by their very use and by providing service isolation. For hands-on research, OpenVZ as a Linux implementation of OS-level virtualization has been used.*

Keywords: virtualization, resilience, point-to-point communication systems

DOI <https://doi.org/10.56082/annalsarscieng.2022.2.97>

1. Introduction

P2p networks benefit from resilience-enhancing strategies in the underlying communication infrastructure, apart from the fact that their specific properties require sophisticated mechanisms. The dynamic nature of nodes requires taking certain precautions such as estimating node reliability, storing redundant information as well as provisioning reliable routing.

In this context, network resilience – the ability to provide and maintain an acceptable service level in the presence of failures – becomes increasingly important. A resilient network should be able to cope with a specific number of failures by remaining completely functional, providing connectivity to all of its components as well as providing enough capacity to fulfil its task.

Peer-to-peer networks enable such functionalities as the distributed searches. Each node in a point-to-point network performs both client and server functions, unlike client-server systems with asymmetric roles. The architecture of p2p networks is decentralized. They are built to handle choices, in the sense of adding or removing nodes. At the same time, the data stored in a p2p network are replicated

¹Professor, PhD Romanian Academy of Scientists;

² PhD, Titu Maiorescu University, Bucharest

on multiple nodes in the network. At the same time, the building of services using p2p networks is complex. The service, because it is provided by the nodes, depends on the trust among them, i.e. if the nodes perform their tasks- especially the tasks of storing and routing correctly-, as well as on the cooperation among them to provide a resilient service. If a peer-to-peer network is open to the public, as it is the case with file sharing networks, attacks are possible. An example of such attacks is Sybil [8]. In the case of this particular attack, the attacker emulates a big number of peers in the network both by hiding the contents or the nodes and by preventing their accessibility. These attacks can negate the resilience benefits of p2p networks.

2. Operating systems virtualization in a p2p communication network

Virtualization is a concept that involves running an operating system on virtual machines (computers) which are simulated with the help of dedicated software applications. These applications can emulate the operation of all components of a real computer system (CD-ROM, HDD, memory, CPU, USB devices, network card).

Basically, a virtual machine consists of a file of variable sizes that simulates the hard disk and several other files that contain the configurations of the respective machine (memory size, etc.). Computing power and RAM are shared from the resources of the real machine on which the emulation application is installed.

Hardware virtualization means creating a virtual machine that behaves like a real computer with an operating system installed. The applications running on these virtual machines are separated from the hardware resources existing on the host machine. In other words, if one wants to see how the new operating system behaves, it can be installed and tested on a virtual machine without formatting the hard disk of the real computer. At the same time, testing the behaviour of a software application on a certain operating system can be done using a virtual machine without worrying about the fact that the real operating system could be affected (by viruses or because of abnormal behaviour of the application, etc.).

At the same time, if we take into account the studies that have been carried out, it can be concluded that, in the case of server machines, on average, 16% of the CPU resources, 50% of memory resources; 5-6% of I/O resources are used.

Virtualization is a concept that aims at optimizing the use of these resources. The concept of consolidation appeared in the context of virtualization, too. It

represents the migration of physical machines into virtual machines, running on the same physical machine, in order to make the most of the existing resources.

As regards resilience, the fact that virtual machines are completely isolated from the host machine or other virtual machines, is extremely important. If one virtual machine freezes or crashes, the rest of the machines are not affected. At the same time, by creating a virtualization cluster, when a hardware component fails, the behaviour of virtual machines is not affected, as they share the available hardware resources. By means of virtualization, the entire virtual machine environment is saved as a single file, so backing up, moving or copying is easy.

Multiple virtual machines are installed on a physical machine to maximize CPU and hardware resources over the lifetime of the physical machine.

For hands-on research, OpenVZ, as a Linux implementation of OS-level virtualization, has been used. The most important aspects related to OpenVZ and to the operating system virtualization are: the physical machine is called **hardware node**; virtual machines are known as **containers**; the core is **shared** between node hardware and containers. Each container has: isolated file hierarchy; isolated process space; configuration file (in which memory, "disk" space, etc. are specified).

2.1. OpenVZ resource management

OpenVZ subsystem resource management consists of the following components:

Level Two Disk Quota – OpenVZ server administrator can configure the disk quotas for the virtual environment in terms of disk space and number of nodes. This is the first level of disk partitioning. The second level of partitioning allows the virtual environment administrator to use standard partitioning tools to configure users and groups.

CPU listing for OpenVZ is also two-level. At the first level, the virtualization environment is established, taking into account the priority of the processor which is set at the level of the virtual environment, and the limit settings. At the second level, standard Linux scheduler decides which processes in the virtualization environment reserve time for sharing, using standard process priorities.

User to start the search - this is a set of counters, limits and guarantees of the virtual environment. Some parameters are set to cover all the aspects of virtual environment functioning. Thus, a limited resource for the system, taken as a

whole, cannot be used by only one virtualization environment at the expense of others. Basically, the controlled resources are the memory and various objects in the kernel, such as IPC shared memory segments, network buffers, etc.

2.2. The facilities provided by OpenVZ to ensure resilience

Among the many facilities provided by OpenVZ, the following ones have been taken into account to demonstrate resilience: the possibility of creating containers; of customizing containers; network configuration; data migration.

Installing OpenVZ support on Debian required installing a kernel that had the appropriate support for running the OpenVZ application: **Hint:** `uname -a`.

For the global configuration of OpenVZ it was necessary to edit the file `/etc/vz/vz.conf` accordingly. Each container had its own configuration file, which was located in `/etc/vz/conf/$VEID.conf` (for example, for the container with network ID 10, the configuration file is `/etc/vz/conf/10.conf`). The configuration file of a container could be edited directly or through the utility `vzctl`, by means of such commands as:

```
vzctl set $VEID <param>=<value> --save.
```

The archived container, which is called a *template*, is important because others can be created based on it (by simply copying). The advantage of using templates for creating containers is the ease of use. They can be created by the user (using the `debootstrap` command) or the existing templates, available on the official website, can be used.

2.3. Simulating virtualization in order to ensure resiliency

OpenVZ containers based on the Debian distribution (fig. 2.3.a.) were used for the simulation.

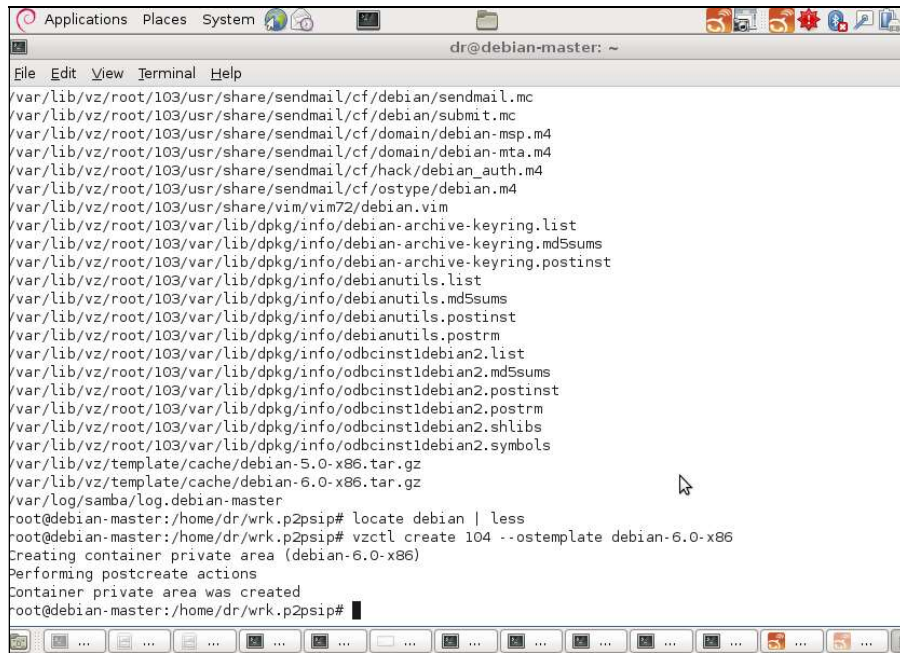


Fig. 2.3.a. Creating a container

The created containers had a standard configuration: 512 MB RAM and 2 GB HDD. OpenVZ, by means of available parameters, provides a fine control of the resources allocated to the respective container. Most of these parameters are in the form of *limita_soft:limit_hard*, and, when the hard limit is reached, the behaviour for applications is non-deterministic.

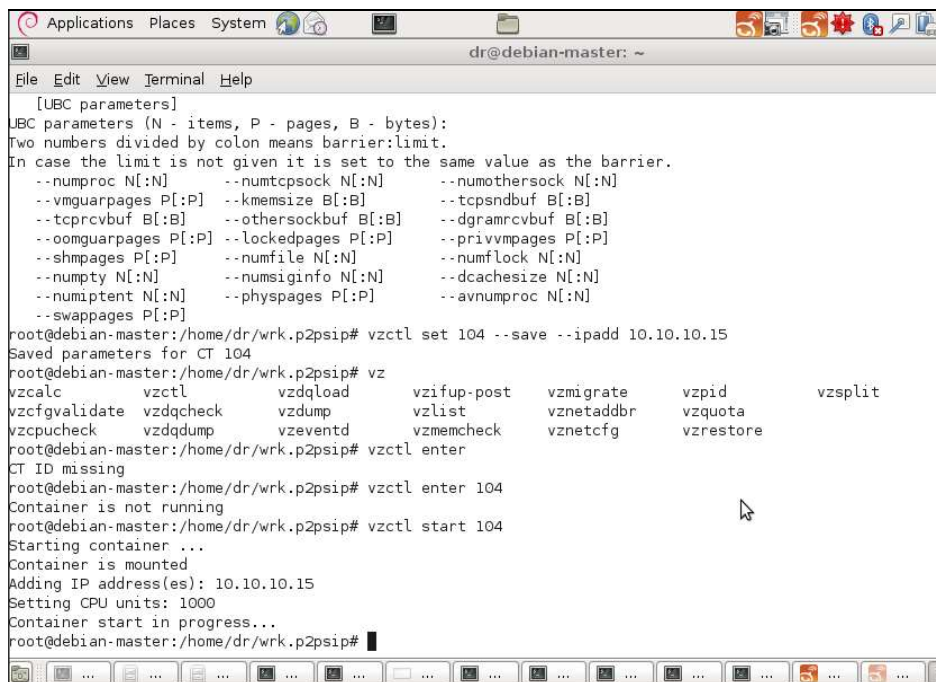
Modifying these parameters in order to adjust the container so that it might meet the resilience conditions, consisted in meeting the following conditions:

- it was necessary that all machines have a `vm$NODi` hostname:
 - `vzctl set $NOD1 --hostname vm$NOD1 --save`
 - `vzctl set $NOD2 --hostname vm$NOD2 --save`
 - `vzctl set $NOD3 --hostname vm$NOD3 --save`
- it was necessary that the machine having VEID `$NOD1` hostname have 128MB RAM:
 - `vzctl set $NOD1 --privvmpages 128M:129M --save`
- it is necessary that all machines have a 2GB storage space on the virtual hard-disk-ul:

```
for i in $NOD1 $ NOD2 $ NOD3; do vzctl set $i --diskspace 2G:2G --
save; done
```

This command has changed only the size of the available space but not the number of nodes that could be used.

For the interaction with the OpenVZ containers, the *vzctl* utility, with the related commands (*start*, *stop*, *restart*, *enter*, *exec*, etc.), was used. After the containers had been configured, they were started (*vzctl start*) (fig. 2.3.b and fig. 2.3.c).



```

Applications Places System dr@debian-master: ~
File Edit View Terminal Help
[UBC parameters]
UBC parameters (N - items, P - pages, B - bytes):
Two numbers divided by colon means barrier:limit.
In case the limit is not given it is set to the same value as the barrier.
--numproc N[:N] --numtcpsock N[:N] --numothersock N[:N]
--vmguarpages P[:P] --kmemsize B[:B] --tcpsndbuf B[:B]
--tcprcvbuf B[:B] --othersockbuf B[:B] --dgramrcvbuf B[:B]
--oomguarpages P[:P] --lockedpages P[:P] --privvmpages P[:P]
--shmpages P[:P] --numfile N[:N] --numflock N[:N]
--numpty N[:N] --numsignifo N[:N] --dcachesize N[:N]
--numtptent N[:N] --physpages P[:P] --avnumproc N[:N]
--swappages P[:P]
root@debian-master:/home/dr/wrk.p2psip# vzctl set 104 --save --ipadd 10.10.10.15
Saved parameters for CT 104
root@debian-master:/home/dr/wrk.p2psip# vz
vzcalc vzctl vdzload vzifup-post vzmigrate vzpid vzsplitt
vzcfgvalidate vdzqcheck vdzdump vzlist vznetaddrbr vzquota
vzcpucheck vdzqdump vzeventd vzmcheck vznetcfg vzrestore
root@debian-master:/home/dr/wrk.p2psip# vzctl enter
CT ID missing
root@debian-master:/home/dr/wrk.p2psip# vzctl enter 104
Container is not running
root@debian-master:/home/dr/wrk.p2psip# vzctl start 104
Starting container ...
Container is mounted
Adding IP address(es): 10.10.10.15
Setting CPU units: 1000
Container start in progress...
root@debian-master:/home/dr/wrk.p2psip#

```

Fig. 2.3.b. Start container

```

dr@debian-master: ~
File Edit View Terminal Help
collisions:0 txqueuelen:0
RX bytes:133927 (130.7 KiB) TX bytes:10939999 (10.4 MiB)

root@debian-master:/home/dr/wrk.p2psip# vzctl enter 104
entered into CT 104
root@debian-master:/#
root@debian-master:/# ifconfig
lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:78 errors:0 dropped:0 overruns:0 frame:0
        TX packets:78 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:5644 (5.5 KiB)  TX bytes:5644 (5.5 KiB)

venet0  Link encap:UNSPEC  Hwaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
        inet addr:127.0.0.1  P-t-P:127.0.0.1  Bcast:0.0.0.0  Mask:255.255.255.255
        UP BROADCAST POINTOPOINT RUNNING NOARP  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

venet0:0 Link encap:UNSPEC  Hwaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
        inet addr:10.10.10.15 P-t-P:10.10.10.15  Bcast:0.0.0.0  Mask:255.255.255.255
        UP BROADCAST POINTOPOINT RUNNING NOARP  MTU:1500  Metric:1

root@debian-master:/#

```

Fig. 2.3.c. Enter container

By default, OpenVZ connects all containers in its own bridge called `venet0`. The network parameters have been configured by means of `vzctl`. Thus, the configured containers have been added in a communication network in view of interacting with them remotely.

All containers have been set for the `10.0.0.0/24` network, each having such IP addresses as `10.0.0.$VEIDi/24`. `10.38.0.1` has been used as nameserver.

```

vzctl set $NOD1 --ipadd 10.0.0.$NOD1 --nameserver 10.38.0.1 --save
vzctl set $NOD2 --ipadd 10.0.0.$NOD2 --nameserver 10.38.0.1 --save
vzctl set $NOD3 --ipadd 10.0.0.$NOD3 --nameserver 10.38.0.1 --save

```

where `$NOD` represents the workstation number.

An IP address has been added on HN's `venet0` interface: `10.0.0.1/24`. Because a client needs to have a root account on that machine, unlike in the case of the classic model, a new range of Virtual Private Servers (VPS) services has been used. Actually, we have provided customers with VPS services using OpenVZ containers. In order to do this, a `veth` interface has been added to each of the previously created containers. At the same time, a bridge that included the virtual

interfaces of the containers and *eth0*, has been added on HN. Thus, the VEs were visible from the outside, that is operations such as to connect via SSH, install and configure services, as well as other operations have become possible.

In order to ease our work, the containers needed virtual interfaces. Thus, by adding interfaces to containers,

```
vzctl set $NOD1 --netif_add eth0 --save
```

```
vzctl set $NOD2 --netif_add eth0 --save
```

```
vzctl set $NOD3 --netif_add eth0 --save
```

a bridge on HN has been created:

```
brctl addbr br0
```

ip link set br0 up (by default the bridge is down)

and interfaces in the bridge have been added:

```
ip flush dev eth0
```

```
brctl addif br0 eth0
```

```
brctl addif br0 veth$NOD1.0
```

```
brctl addif br0 veth$NOD2.0
```

```
brctl addif br0 veth$NOD3.0
```

In order to function in the network, these interfaces have been assigned IP addresses:

- on bridge: `dhclient br0`
- on *eth0* of each container `vzctl exec $NODi dhclient eth0, i=1,2,3`
- Internet connectivity will be tested from each container

Virtual machines were able to ensure the resilience of the p2p communication network by ensuring the isolation of services. In order to show the importance of service isolation for resiliency, a server on which an LDAP service is running, a DNS service, and an APACHE2 have been taken into account. In the usual version, they would run within the same operating system. It becomes problematic when it is intended to move the services to other machines with better performance (APACHE2) or to replace the machine on which they are running. One solution is to use virtual machines (containers, in this case) for an easy migration of services from one machine to another with almost 0 configuration. This migration concept is based on the idea of checkpointing: saving the current state and restore: restoring after a checkpoint. The machine with VEID \$NOD1, on which the DNS service had been installed, was moved to machine \$NOD-1. The migration was carried out by taking the following the steps:

1. I suspended the current machine from execution using the command:
`vzctl chkpnt $NOD1 --suspend`
2. I created the checkpointing:
`vzctl chkpnt $NOD1 --dump --dumpfile dump.$NOD1`
3. I shut down the machine in order to save the information from the file system:
`vzctl chkpnt $NOD1 --kill`

At this point, the checkpointing information was stored and the migration process could begin:

- a. I archived the contents of `/var/lib/vz/private/$NOD1`
- b. I copied the previously created archive together with the checkpoint and the configuration file located in `/etc/vz/conf/$NOD1.conf` to the destination machine (\$NOD-1)
- c. I unzipped the machine in `/var/lib/vz/private/$NOD1`
- d. I recreated the checkpoint
`vzctl restore $NOD1 --undump --dumpfile dump.$NOD1`
`vzctl restore $NOD1 --summary`

I did the reconfiguration taking the following steps:

- the imported machine will stop
`vzctl stop $NOD1`
- the suspended machine will be resumed
`vzctl restore $NOD1 --undump --dumpfile dump.$NOD1`
`vzctl restore $NOD1 --resume`

Increasing the resilience of p2p communication networks generates many algorithmic challenges. In particular, creating alternative routes to protect communication often requires hardware troubleshooting and can often be done entirely heuristically or by means of special topologies.

Conclusions

Conclusion (1) The present paper has focused on the study of peer-to-peer communication networks. They are networks that evolve together with the development of computing systems and communication environment. Such networks can evolve with potential vulnerabilities that cannot be predicted and that can be exploited. These vulnerabilities can be the result of an attack or a network failure, following a confluence of unexpected circumstances.

Conclusion (2) The present research has focused on the concept of virtualization to show that the systems using virtualization are resilient systems with a high degree of operational safety. As concerns resilience, the fact that virtual machines are completely isolated from the host machine or other virtual machines

is of particular importance. If one virtual machine freezes or crashes, the others are not affected. At the same time, the creation of a virtualization cluster is a guarantee of the fact that, when a hardware component fails, the operation of the virtual machines is not affected due to the fact that virtual machines can share the available hardware resources. By means of virtualization, the entire virtual machine environment is saved as a single file, so backing up, moving or copying is easy.

Conclusion (3) Resource management is very important for OS-level virtualization solutions because there is a finite set of resources in a single core. These resources are shared between multiple virtual environments. At the same time, all these resources must be controlled in a way that allows multiple virtual environments on a single system, and that do not influence each other.

Notes and abbreviations

| | |
|--------|---------------------------------------|
| DNS | Domain Name System |
| IP | Internet Protocol |
| IPC | Mecanisme de comunicare intre procese |
| LDAP | Lightweight Directory Access Protocol |
| NAT | Network Address Translator |
| p2p | Peer to Peer |
| RAM | Random-Access Memory |
| SIP | Session Initiation Protocol |
| VPN | Virtual Private Network |
| VoIP | Voice over Internet Protocol |
| OpenVZ | Open Virtualization |

REFERENCES

- [1] A. Bacivarov, "Fault tolerant techniques for integrated circuits in submicron and nanotechnologies", Proc. SPIE, Vol. 6635, 66350B (2007); SPIE Digital Library doi:10.1117/12.741873.
 - [2] A.-L. Barabasi, R. Albert, and H. Jeong, "Scale-free Characteristics of Random Networks: The Topology of the World Wide Web," *Physica A* 281, 2000.
 - [3] A. Fessi, "Resilient Application Layer Signaling based on Supervised Peer-to-Peer (p2p) Networks", Technische University at Munchen, 2010.
 - [4] C. Baransel, W. Doboseiwicz, and P. Gburzynski, "Routing in Multi-hop Packet Switching Networks: Gbps Challenge," *IEEE Network Magazine*, 1995.
 - [5] F. Auder, C. Jennings, Network Address Translation (NAT), January 2007
 - [6] M. Fisher, S. Grau, G. Schafer, T. Strufe, "Methods for Improving Resilience in Communication Networks and P2P Overlays".
 - [7] J. Aspnes, Z. Diamadi, and G. Shah, "Fault-Tolerant Routing in Peerto-Peer Systems," *ACM PODC*, July 2002.
 - [8] J. Douceur. The Sybil attack. In *Proc. of the IPTPS02 Workshop*, Cambridge, MA (USA), March 2002.
 - [9] R. Albert, H. Jeong, and A.-L. Barabasi. "Error and attack tolerance of complex networks" *Nature*, 406(6794):378-382, July 2000.
 - [10] V. Pau, D.L. Copaci, "Considerations on the resilience and security of Communication Networks", *Annals of the Academy of Romanian Scientists, Series on Engineering Sciences*, Vol. 11, nr. 1/2019, ISSN 2066-8570.
 - [11] Y. Azar, A. Broder, A. Karlin, and E. Upfal, "Balanced Allocations," *SIAM J. on Computing*, vol. 29, no. 1, 1999.
 - [12] <https://openvz.org/>.
-