2015 IEEE International Conference on Cloud Engineering

Comparing Containers versus Virtual Machines for Achieving High Availability

Wubin Li and Ali Kanso

Ericsson Research, Ericsson, Montréal, Canada
{wubin.li, ali.kanso}@ericsson.com

***Abstract*—In recent decades, virtualization as an abstraction from physical hardware has become a popular solution to resource isolation and server consolidation. With the surge in adoption of virtualization technologies, ensuring High Availability (HA) for applications hosted in virtualized environments emerges as an important problem and has garnered substantial attention. In this paper, we present a brief comparison of virtualization technologies from a HA perspective. The state-of-the-art HA solutions in two mainstream types of virtualized platforms (i.e., hypervisor-based platform and container-based platform) are respectively investigated in terms of limitations and features such as live migration, failure detection, and checkpoint/restore. One of our key findings is that, compared with hypervisor-based platforms, HA features in container-based platforms are far from enough. From a HA perspective, extensions on top of container technologies are required.**

Keywords high availability, fault tolerance, virtualization, hypervisor, container, cloud computing.

I. Introduction

Recent development and optimization in virtualization technologies [2], [6], [15], [17], [22], [29] in the past decades have led to their wingspread adoption and a growing trend toward hosting workload in virtualized platforms. While virtualization technologies promise a reduction in cost and complexity through abstraction of physical resources, they also raise the concerns on the availability of applications hosted in virtualized platforms. In particular, high VM density on a single host may have a negative impact on the availability of applications encapsulated, since all VMs as well as the services provided by these VMs will fail upon an event of host failure.

To guarantee applications' high availability (HA) despite and in presence of a disastrous and disruptive event, HA features such as VM monitoring , live migration and checkpoint/restore, are integrated into the virtualization technologies. In this article we conduct a comparison of virtualization technologies from a HA perspective. The state-of-the-art HA solutions in two mainstream types of virtualized platforms (i.e., hypervisor-based platform and container-based platform) are respectively investigated in terms of limitations and HA features.

The remainder of the article is organized as follows: Section II briefly gives a general background introduction on HA and Fault Tolerance (FT). Section III surveys HA solutions in hypervisor-based platforms using VMware platform and XenServer platform as examples. Section IV presents HA solutions in container-based platforms and specifically elaborates a state-of-the-art implementation (i.e., CRIU) targeting HA features for containers. Section V discusses some on-going efforts on clustering containers which can potentially help to achieve high availability. Finally, some conclusions are presented in Section VI, followed by a presentation of acknowledgments, and a list of references.

II. Preliminaries

High availability (HA) refers to the measurement of the ability of a system (or application, component) to remain continuously functional with 99.999% of the time (aka. five nines) [12]. Some systems such as air-traffic-control system and emergency-response (9-1-1) system have even more severe demands on availability, namely fault tolerance (FT), or service continuity.

Stateless applications with the ability to store their state in a replicated distributed storage can be deployed behind load balancers where their failures are masked by redirecting the traffic to other healthy replicas. Nevertheless for statefull applications where the state of the application and the network stack needs to be maintained, a different approach is needed.

In order to achieve HA in a virtualized environment, enabling features including 1) the ability to retrieve and save the state of a VM, and 2) the ability to perform VM live migration, are required by virtualization platforms. In addition to these, a complete HA solution should also include mandatory features, including 1) continuous failure detection, 2) automatic state synchronization between the active and standby, and 3) recovery management by dynamically failing over to the standby when a failure is detected and terminating the active.

Virtualization vendors offer different HA/FT solutions to their customers. Commonly among those solutions, HA is implemented by building multiple levels of failover capabilities into a system. A typical HA solution consists of a pool of loosely coupled servers which are self-contained and continuously health monitor (e.g., via heartbeat) each other. In an event of host failure, VMs can failover from one server to others. For FT purpose however, the workload failover is not enough. To guarantee service continuity, a secondary replica (hot standby) is required to be tightly coupled and consistent with the primary, such that in case of failure the replica

978-1-4799-8218-9/15 $31.00 © 2015 IEEE
DOI 10.1109/IC2E.2015.79

is always ready to take-over without service interruption and data loss.

While HA/FT solutions may significantly differ in implementation, they share the same principle, that is, duplicating critical components via redundancy in an attempt to eliminate single points of failure.

III. HA IN HYPERVISOR-BASED PLATFORMS

A. Overview of HA Solutions

Table I presents a brief overview of HA features by three of the main vendors, i.e., VMware, Citrix XenServer, and Marathon everRun MX. In general, HA is supported by all of them. In fact, according to our study, most of the vendors (e.g., Microsoft Azure [19], HP Serviceguard [13], and Red Hat Enterprise Linux OpenStack Platform [26]) in the market provide integrated HA using failover clustering strategies mentioned in Section II.

As illustrated in Table I, VM live migration is supported both by VMware and XenServer via *vMotion* and *XenMotion* respectively. However, CPU compatibility is required to ensure that a VM can perform normally on the destination host after migration. The CPUs on the source and destination hosts are expected to provide the same set of features to the VM so that the live VM and accompanying applications do not crash. Meanwhile, Checkpoint/Restore is also supported by all of them, enabling the capability of VM snapshotting.

FT: Checkpointing vs. Record-and-Replay: In contrast to HA that can be commonly achieved by failover clustering, VM FT is more challenging in virtualized platforms as efficiently synchronizing a secondary VM with a primary VM is a complicated task. Currently, there are two main strategies addressing this problem.

The first one is ***record-and-replay***, which records all input data in the primary VM, sends it over a dedicated link to the secondary replica and then replay it in the replica. Implementing this strategy for a uniprocessor VM is comparatively simple as all instructions executed by the vCPU in the primary VM are replayed deterministically on the vCPU in the secondary VM. However, for concerns on performance, modern CPU architectures usually consist of multiple processors and employ techniques such as branch prediction, speculation, and out-of-order execution, introducing non-deterministic behavior across program executions. This non-determinacy increases the complexity of synchronizing a replica with a primary execution. This challenge is often referred to as symmetric multiprocessing fault tolerance (SMP FT). Pereira et. al from Intel argue that an efficient record-and-replay system targeting this challenge must be assisted by hardware support [24]. Because of this, VMware who applies this strategy can currently only manage to provide FT for uniprocessor VMs. At the time of this writing, only Marathon everRun MX [27] announced that they had solved this problem. However, the information about it is pretty scarce.

Instead of recording the inputs in the primary VM, another alternative strategy (i.e., ***checkpointing***) checkpoints the state of the VM after the inputs happen, sends it to the replica, and keeps the replica VM consistently synchronized with the primary. In contrast to record-and-replay, checkpointing has its advantages in its simplicity and SMP support. However, its performance heavily relies the checkpointing frequency and the amount of data that needs to checkpoint and transfer to the replica side. Remus [4] and Kemari [21] are implementations of this kind. However, despite the fact that Remus has been a part of XEN hypervisor (since version 4.0), it is not included in XenServer for commercial use (see Table I). The same applies to Kemari which is integrated in KVM hypervisor but not product-ready yet.

Finally, while Marathon everRun MX provides all HA/FT features, it only supports Microsoft Windows as the guest OS, strictly limiting customers to a single option.

B. HA in VMware

VMware platform utilizes VMware Distributed Resource Scheduler (DRS), VMware HA, VMware FT and vMotion for virtual environment high availability.

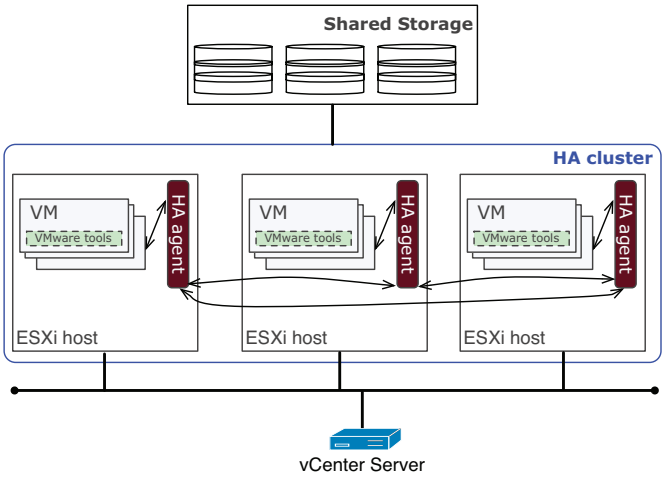

Fig. 1. VMware HA: High-Level Architecture.

Figure 1 presents an architectural overview of VMware HA, showing that VMware HA is also built on failover clustering strategy [28]. All VM disk images are required to locate on shared storage. The *HA agents* installed on all ESXi hosts are responsible for maintaining heartbeats between hosts in the cluster, heartbeats between VMs and the vCenter server, and heartbeats between applications and the vCenter server.

TABLE II
VMWARE HA ACTIONS ON DIFFERENT FAILURE SCENARIOS.

Failure	HA action	Extra requirement
Host failure	restart VMs on other hosts	VMware DRS
Guest OS failure	reset VM on the same host	VMware tools
Application failure	restart VM on the same host	VMware tools

As listed in Table II, VMware HA protects against three types of failures. If an ESXi host fails, the heartbeat signal is no longer sent out from that host. VMware HA restarts affected

TABLE I
COMPARISON OF VMWARE, XEN, AND MARATHON EVERRUN MX.

Features	VMware	Citrix XenServer	Marathon everRun MX
Fault tolerance	Yes but no SMP support	Feasible with Remus [4] integration but not product-ready yet	Yes SMP supported, product-ready
Live migration	Yes vMotion CPU compatibility required	Yes XenMotion CPU compatibility required	Yes built on top of XenServer environment.
Checkpoint/Restore	Yes	Yes	Yes
Failure detection	Automatic	Automatic	Automatic
Failover management	Yes	Yes	Yes
Guest OS	Any	Any	Microsoft Windows Only

VMs on other surviving hosts. In case of guest OS failure, the heartbeat signal is sent between the VM and the vCenter server. If a VM fails or the guest OS within the VM freezes, the VM tools installed inside the VM also freezes, which results in no longer sending the heartbeat to the vCenter server. In response, vCenter server resets the VM on the same host. Similar to the previous two cases, failure on application level is detected by monitoring the heartbeat between the application and the vCenter server. In case of application failure, vCenter server restarts not just the application, but the entire VM on the same host followed by the application.

For planed or predictable scenarios (e.g., host maintenance/upgrade), HA for VMs can be achieved via live migration by vMotion. In case of VM migration or restarting a VM on a different host, the decision of the new location for a VM is made by VMware DRS according to information such as the state of the hosts in the cluster, the resource consumption of a VM over time, and (anti-)affinity rules.

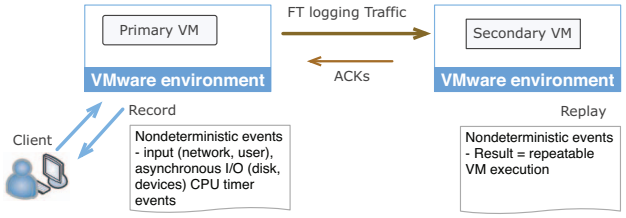

Fig. 2. VMware FT: vLockStep overview.

Figure 2 shows the current released implementation of VMware FT. The execution of the primary and the secondary replica of a FT-protected VM are synchronized through the *vLockStep* protocol by sending all input data over a dedicated FT logging link to the secondary copy and replay it there. vLockstep technology is supported by physical processor extensions which were developed in collaboration with Intel and AMD. All currently shipping x86 server processors are vLockstep capable. To use vLockStep, a fast link network (at least 1 Gbit) between the primary and the secondary is required. In addition, the secondary VM must reside on the same datastore[+] as the primary. Keeping the primary and secondary VMs on the same host is possible if they are both in a powered-off state. But when the primary VM is power-on, the host-level anti-affinity check is performed and the secondary VM has to be started on a different host at that time. When vLockStep is initiated, the hypervisor on the two hosts establish a system of heartbeat signals and mutual monitoring. In case of failure on either host, the other host can take over and continue running the protected VM seamlessly via transparent failover.

However as aforementioned, VMware has FT capabilities based on the current implementation, but only a single logical processor on the VM is supported. To protect multiprocessor VMs, VMware is developing a new protocol, namely, the *SMP FT protocol*, which has even higher network requirements (at least 10 Gbit link). The increased bandwidth is used to not only synchronize multiple vCPUs, but also to eliminate the requirement on shared storage between the primary and secondary VM. While technical previews on SMP FT were presented in VMworld 2012 and 2013 [14], there is no guarantee on the time of its product release. One major reason is that performance overhead introduced by SMP FT is too heavy. According to the evaluation presented [14], depending on the workload, the performance of a VM with SMP FT enabled is only between 55% and 75% of the same VM without FT protection.

C. HA in XenServer platform

Similar to VMware, XenServer platform also offers HA protection for VMs, but it can only handle failures on host level. XenServer can be augmented with third-party products (e.g., HA-Lizard [25]) to deliver HA capabilities to address failures on VM level and application level. Moreover, built-in support for FT is not available in XenServer platform. One potential direction to enable FT in XenServer is to integrate Remus [4], which has been a part of XEN hypervisor.

Remus uses an active-passive configuration where the state of the VM is continuously replicated from the primary host to the secondary host. As illustrated in Figure 3, Remus allows speculative execution to concurrently run the active VM slightly ahead of the replicated system state. This allows the primary server to remain productive, while synchronization with the replicated server is performed asynchronously, improving the performance of the primary VM. In the meanwhile, the output at the primary is buffered (not released to the external client) until the complete checkpoint is ac-

[+]In the context of VMware environment, a datastore is a storage location within a datacenter for VM files. It can be a directory on network attached storage, or a local file system path.

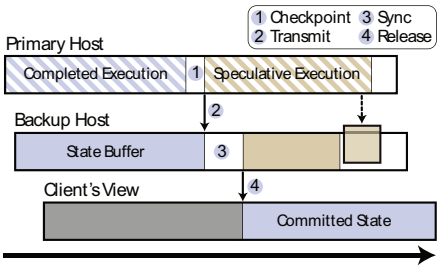

Fig. 3. Speculative execution and asynchronous replication in Remus. Illustration from [4].

knowledged, keeping the consistence between the primary and the secondary. Once again as aforementioned, while XEN hypervisor is with Remus support, it is not included with XCP, XenServer.

IV. HA IN CONTAINER-BASED PLATFORMS

As opposed to hypervisor-based virtualization, container-based virtualization (aka. operating system-level virtualization) is not aimed at emulating an entire hardware environment, but rather enabling the modern Linux kernel to manage isolation between applications. With OS-level virtualization, multiple isolated Linux systems (containers) can run on a control host, sharing a single kernel instance. Each container has its own process and network space. Systems such as LXC (LinuX Containers) [17], Docker [6], and OpenVZ [22] are implementation examples of this kind.

In essence, a container is a set of processes (usually with storage associated) completely isolated from other containers. To ensure high availability of a container, the capability of process checkpoint/restore is a must. There has been a large amount of effort devoted to targeting this challenge, including BLCR [9], DMTCP [1], and ZAP [23]. However, these systems were not specifically designed for containers. They either lack one feature or another, or usually only support a limited set of applications. Moreover, none of them has been a part of the mainstream Linux kernel due to the complexity of implementations.

By adding high-level features such as *versioning* and *sharing* on top of LXC[+] Docker is becoming an attractive platform for container hosting. However, HA features in Docker/LXC are not available at the time of this writing. For example, the capability of checkpointing in Docker relies on the *lxc-checkpoint* [18] command by LXC, but it is not implemented yet. As an alternative, one can snapshot a running container using the *commit* [5] command, which however, only saves a container's file changes and settings into a new image, with no concern on the state of the running processes. While another container based on the snapshot image can be restarted on other hosts, the state of running processes in the previous container is not preserved.

[+]With the release of version 0.9 (and thereafter) Docker has replaced LXC with libcontainer [7], an implementation of LXC-like control over cgroups and namespaces, as the default execution environment.

TABLE III
HA FEATURES IN DOCKER/LXC AND OPENVZ.

Features	Docker/LXC	OpenVZ
Live Migration	No	Yes
Checkpoint/Restore	No	Yes
Automatic state synchronization	No	No
Failure detection	No	No
Failover management	No	No
Kernel patches	N/A	Yes

Conversely, as presented in Table III, features of live migration and checkpoint/restore in OpenVZ are complete and ready-to-use. They are implemented as loadable kernel modules plus a set of user space utilities [20]. However, like other existing work aforementioned, a major shortcoming of OpenVZ is the lack of official integration with upstream Linux kernel. While these features are present in OpenVZ kernel since April 2006, they were never accepted to upstream Linux kernel. OpenVZ with full features are now only available on 2.6.32 kernel with customization. There will be issues (e.g., security and compatibility) for users when using old kernels. To tackle this problem, a new working direction (namely, CRIU [3]) is being explored, that is, moving most of the checkpointing complexity out of the kernel and into user space, thus minimizing the amount of the in-kernel changes needed. At the time of this paper, more than 150 patches from CRIU (e.g., memory tracking patches) have been merged into mainline kernel for checkpoint/restore functionality and can be potentially integrated with LXC [10]. Also note that, there is no other container-based platform that supports features of failure detection, automatic state synchronization between the active and standby, and failover management.

A. Checkpoint/Restore and Live Migration in CRIU

Figure 4 illustrates the sequences for live migration of a container. The current implementation in OpenVZ and CRIU is in line with these sequences.

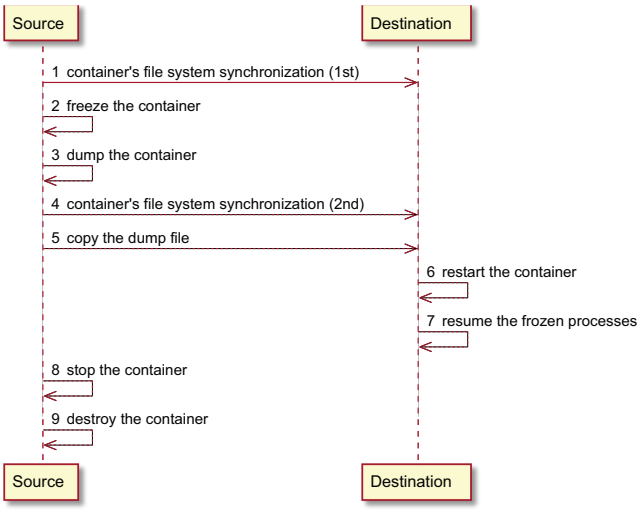

Fig. 4. Sequence diagram for live migration in CRIU.

An advantage of this implementation is that the migration process can rollback to source and resume the container on the source when there is a failure (e.g., in case of network disconnectivity during the transfer of memory pages or the synchronization of file system).

Service availability mostly depends on the time consumed in stages $3-5$. Three optimizations, i.e., file system changes tracking, lazy migration, and iterative migration, are introduced to decrease the service downtime.

1) File System Changes Tracking: The basic motivation is to decrease the time for file system synchronization at stage 4 in Figure 4. This is done by tracking the file system changes and continuously synchronizing the changes only. The current optimization in CRIU is implemented on top of ploop, a block device similar to Linux loop device but specifically designed for containers. A major feature of ploop is *write tracker*, which enables the kernel memorizes a list of modified data blocks. This list then can be used to efficiently migrate a ploop device to another host, with minimal container downtime. User-space support for this is implemented in ploop copy tool and is used by *vzmigrate* utility.

2) Lazy Migration: As highlighted in Figure 5, the basic idea of lazy migration is to only transfer a minor subset of memory pages to the destination, resume the container on the destination, and then pull the remaining pages from the source on demand. With lazy migration, the container can be resumed on the destination without waiting for full memory copy from the source. On a page miss, the container sends a request to the page-in swap device who then redirects the request to the page-out daemon resides on the source to pull the missing page. In this way, the requested page is transferred and loaded into memory on the destination (see steps 1-5 in Figure 5). Finally, whenever the container is idle for a certain timeslot (which is configurable), a last swap-out action is forced and all remaining pages are transferred from the source to the destination.

A major drawback of lazy migration is that neither the source nor the destination holds an integral state of the container, meaning that the source and the destination as well as the network connection in-between must be reliable until the full migration process is completed. In particular, the container restored on the destination may become malfunctioning due to incomplete pulling of memory pages in case of network failure between the source and the destination. Moreover, rollback is infeasible under such a circumstance.

3) Iterative Migration: Another optimization (iterative migration) is to perform memory pages transfer and file system synchronization prior to freezing the container, striving to reduce the amount of data needed to transmit after the freeze of the container. However, as pages are being dirtied and file systems might also be changing during the data transfer, this process needs to be executed iteratively. Taking this strategy, the file system is iteratively synchronized, simultaneously with dirtied memory pages migration in each iteration.

As shown in Figure 6, the iterative process stops if there is no change or a configurable threshold condition is reached.

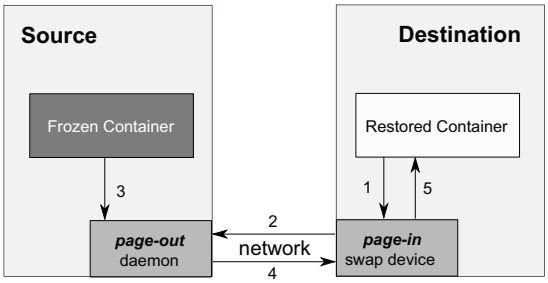

Fig. 5. Lazy Migration. Illustration from [20].

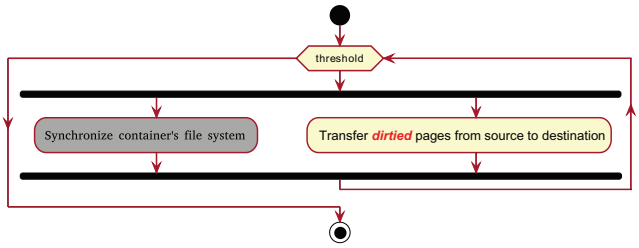

Fig. 6. Iterative Migration.

V. Clustering Efforts for Containers

Linux containers are still in their infancy when it comes to leveraging their potentials. It is well understood that the power of containers is not only in their individual value, but also in their collective functionality to build services with multiple building blocks. For this purpose, the effort of building cluster-level management for containers is ongoing. Docker Swarm [8] is a Docker-native clustering system that aims at exposing a cluster of Docker hosts as a single virtual host. However, to date, Docker-Swarm is still it its incubation phase and it does not natively offer support for the availability of the Docker instances. Kubernetes [11] is another promising clustering solution for containers, where tightly coupled containers are grouped into pods, and the loosely coupled cooperating pods are organized into key/value labels. The labels are considered metadata describing the semantic of the service composed of multiple pods. Kubernetes define the notion of a *ReplicaController* which ensures that a predefined number of replicas of a given pod is always running. In case of a pod failure, it can restart it on another healthy host. Although this solution can work for stateless services, it is still lacking the state preservation mechanisms which are essential for the service continuity. Moreover, at the time of this writing, Kubernetes is still in its pre-production beta version.

VI. Concluding Remarks and Notes

In this paper, virtualization technologies are compared from a HA perspective. As summarized in Table IV, we conclude that HA/FT solutions already exist in hypervisor-based platforms, and are commonly achieved by failover clustering. However, these solutions have limitations, e.g., either on SMP support or on guest OS. On the other hand, in container-based

TABLE IV
COMPARISON OF VIRTUALIZATION TECHNOLOGIES FROM A HA PERSPECTIVE.

Features	VMware	Citrix XenServer	Marathon everRun MX	Docker/LXC	OpenVZ
Fault tolerance	Yes but no SMP support	Feasible with Remus integration but not product-ready yet	Yes SMP supported, product-ready	No	No
Live migration	Yes vMotion CPU compatibility required	Yes XenMotion CPU compatibility required	Yes built on top of XenServer environment.	No	Yes
Checkpoint/Restore	Yes	Yes	Yes	No	Yes
Failure detection	Automatic	Automatic	Automatic	No	No
Failover management	Yes	Yes	Yes	No	No
Guest OS	Any	Any	Microsoft Windows Only	Linux	Linux kernel restriction

platforms, there are large amount of missing pieces. In particular, the checkpoint/restore features in container-based environments are far from complete. Moreover, despite the efforts of clustering containers, there is no mature feature available for continuous monitoring to detect failure of the container and automatic failover management, which are mandatory for a complete HA solution[+]. In conclusion, extensions on top of container technologies are required from a HA perspective.

On optimizing HA features in container-based platforms, it would be interesting to investigate the feasibility (as well as the overhead) of using compression technique such as *gzip*, *bzip2*, and *rar* to reduce the volume of data transmitted from the source host to the destination and thus speed up the live migration procedure. In addition, there exists no attempt to employ record-and-replay to realize FT on containers.

ACKNOWLEDGMENTS

We are grateful to the CRIU team (especially Kir Kolyshkin and Andrew Vagin) for providing valuable information about OpenVZ and CRIU. We would also like to thank the anonymous reviewers for their constructive feedback.

REFERENCES

[1] Jason Ansel, Kapil Arya, and Gene Cooperman. DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2009)*, pages 1–12. IEEE, 2009.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[3] CRIU. Checkpoint/Restore In Userspace. http://criu.org/, visited Jan 2015.

[4] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. San Francisco, 2008.

[5] Docker. Docker Commit Command. https://docs.docker.com/reference/commandline/cli/#commit, visited Jan 2015.

[6] Docker. Docker: The Linux Container Engine. http://www.docker.io, visited Jan 2015.

[7] Docker. libcontainer source. https://github.com/docker/libcontainer, visited Jan 2015.

[+]As a follow-up of this study and to compensate the limitations of containers in achieving HA, we carried out our first attempt and built a middleware comprising of a set of HA agents which are capable of continuous monitoring the state of containers (as well as the applications encapsulated) and performing failover in case of failure [16].

[8] Docker Swarm. Swarm: a Docker-native clustering system. https://github.com/docker/swarm/, visited Jan 2015.

[9] Jason Duell. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. *Lawrence Berkeley National Laboratory*, 2005.

[10] Pavel Emelianov and Serge Hallyn. State of CRIU and Integration with LXC. Presented at Linux Plumbers Conference 2013.

[11] Google. kubernetes.io. http://kubernetes.io/, visited Jan 2015.

[12] Jim Gray and Daniel P. Siewiorek. High-Availability Computer Systems. *Computer*, 24(9):39–48, 1991.

[13] HP. HP Serviceguard Solutions. http://www.hp.com/go/serviceguardsolutions, visited Jan 2015.

[14] Jim Chow et. al. VMware vSphere Fault Tolerance for Multiprocessor Virtual Machines - Technical Preview. Presented at VMworld 2013.

[15] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. KVM: the Linux Virtual Machine Monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[16] Wubin Li, Ali Kanso, and Abdelouahed Gherbi. Leveraging Linux Containers to Achieve High Availability for Cloud Services. In *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E 2015)*. IEEE, 2015. accepted.

[17] LXC. LXC - LinuX Container. https://linuxcontainers.org/, visited Jan 2015.

[18] LXC. lxc-checkpoint – checkpoint a running container (not implemented yet). http://lxc.sourceforge.net/man/lxc-checkpoint.html, visited Jan 2015.

[19] Microsoft. Azure: Microsoft's Cloud Platform. https://azure.microsoft.com/, visited Jan 2015.

[20] Andrey Mirkin, Alexey Kuznetsov, and Kir Kolyshkin. Containers Checkpointing and Live Migration. In *Proceedings of the Linux Symposium*, pages 85–92, 2008.

[21] Nippon Telegraph and Telephone Corporation. Kemari Project. http://www.osrg.net/kemari/, visited Jan 2015.

[22] OpenVZ Project. OpenVZ Linux Containers. http://openvz.org, visited Jan 2015.

[23] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The Design and Implementation of Zap: A system for Migrating Computing Environments. *ACM SIGOPS Operating Systems Review*, 36(SI):361–376, 2002.

[24] Cristiano Pereira, Gilles Pokam, Klaus Danne, Ramesh Devarajan, and Ali-Reza Adl-Tabatabai. Virtues and Obstacles of Hardware-assisted Multi-processor Execution Replay. *HotPAR, June*, 2010.

[25] Pulse Supply. HA-Lizard - High Availability for XenServer. http://www.halizard.com/, visited Jan 2015.

[26] Red Hat. Red Hat Enterprise Linux OpenStack Platform. http://www.redhat.com/products/enterprise-linux/openstack-platform/, visited Jan 2015.

[27] Stratus Technologies. everRun MX. http://www.stratus.com/Products/Software/everRun/everRun-MX, visited Jan 2015.

[28] VMware Inc. VMware High Availability Concepts, Implementation and Best Practices. http://www.vmware.com/files/pdf/VMwareHA_twp.pdf. visited Jan 2015.

[29] VMware Inc. vSphere ESX and ESXi Info Center. http://www.vmware.com/products/esxi-and-esx/overview.html, visited Jan 2015.