Virtualization: Past and Present Challenges

Bruno Rodrigues[1], Frederico Cerveira[2], Raul Barbosa[2] and Jorge Bernardino[1,2]
[1]*Polytechnic Institute of Coimbra, Superior Institute of Engineering of Coimbra, Coimbra, Portugal*
[2]*CISUC, Department of Informatics Engineering, University of Coimbra, P-3030 290, Coimbra, Portugal*

Keywords: Virtualization, Security, Dependability, Performance, Cloud Computing.

Abstract: Virtualization plays an important role in cloud computing by providing the capability of running multiple operating systems and applications on top of the same underlying hardware. However, there are limitations in current virtualization technologies, which are inherited by cloud computing. For example, performance, security and dependability. While these challenges remain, the adoption of virtualization in the different fields will be limited. This paper presents a survey in the past and open virtualization challenges, such as nested virtualization, reliability and security.

1 INTRODUCTION

Virtualization plays an ever-widening role in today's society as can be seen from its applications, ranging from critical embedded systems (Heiser, 2008) (such as automotive (Wolf, n.d.) (Lee et al., 2016) and aeronautical applications (Joe et al., 2012)) to big private and public datacenters (e.g., cloud computing (Xing and Zhan, 2012)).

Despite virtualization being extensively used, there are still open challenges (Adams and Agesen, 2006) that present a barrier to adoption by customers that require strict security and dependability guarantees.

Ultimately, virtualization adds another layer of software (i.e., the hypervisor and other smaller components), which inevitably carries its own bugs (Lacoste and Debar, n.d.). This layer is of utmost importance as it is located right above the hardware, meaning that it has direct and unchecked control over the hardware and is a single point-of-failure (SPOF) of the system.

Co-location of virtual machines over the same hardware also exposes the clients of virtualization to dependability and security threats. On the one hand, transient and permanent hardware faults will now be amplified and propagated across the clients of the system. On the other hand, malicious clients can launch rogue virtual machines to carry denial of service (Yan et al., 2016), information disclosure or privilege escalation attacks (Lombardi and Di Pietro, 2011) on co-located virtual machines (e.g., by exploiting shared hardware resources, such as the CPU cache).

Research in recent years has proposed approaches to solve some of the previously mentioned problems, particularly with regards to hardware fault-tolerance (Cully et al., 2008), hypervisor rejuvenation (Bagdi et al., 2017a), and security (Wang and Jiang, 2010).

In this paper, we survey the most important challenges that virtualization has overcome in the past and present the open challenges that are being extensively researched, along with the solutions available in literature. Throughout this paper, abbreviations are used such as OS which stands for operating system, VM which stands for virtual machine, ISA which stands for instruction set architecture and CPU which stands for central processing unit.

The structure of this paper is as follows. Section 2 provides the main concepts about virtualization. Section 3 presents the past challenges faced by virtualization, i.e., the challenges already overcome by virtualization, while in Section 4 we present the open challenges to virtualization, i.e., the challenges being currently faced by virtualization. Finally, Section 5 presents the main conclusions and some future work.

2 CONCEPTS OF VIRTUALIZATION

Virtualization can be defined as a layer of indirection

Rodrigues, B., Cerveira, F., Barbosa, R. and Bernardino, J.
Virtualization: Past and Present Challenges.
DOI: 10.5220/0006910707550761
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 755-761
ISBN: 978-989-758-320-9

between abstract view and implementation of resources (Aboulnaga, n.d.).

The advantages brought by virtualization can be summarized as cost reductions in hardware and power consumption, better hardware utilization (Colsani et. al., 2008) (i.e., less idle time) and good resource elasticity (i.e., request and pay for the resources required if operating over a private or public cloud) (Herbst et al., n.d.). In summary, it allows a single computer to host several clients and their jobs.

Figures 1 and 2 show the difference between a virtualized and a non-virtualized system. Figure 1 shows hardware utilization of 10-20% without virtualization, due to various services being spread over multiple physical machines that are over-provisioned and under-used. Whereas Figure 2 shows that virtualization can increase hardware utilization (to around 70%) by consolidating those services over a single physical machine.

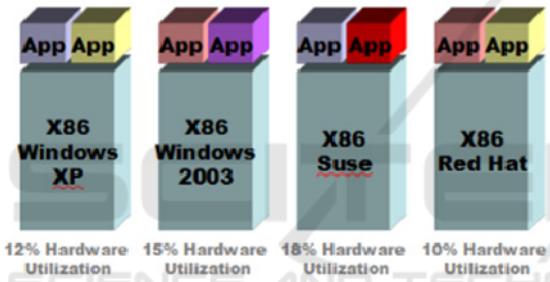

Figure 1: Hardware usage without virtualization (Colsani et. al., 2008).

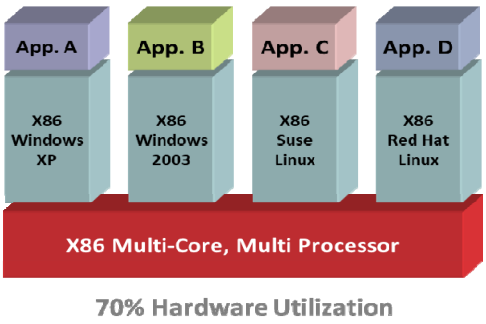

Figure 2: Hardware usage with virtualization (Colsani et. al., 2008).

Virtual machines can be divided into two main categories, namely process virtual machines and system virtual machines (Berghmans, 2010). On the one hand, process virtual machines can only run a single program at a time and for example, a process of an operating system, is a process VM, because one process can only run one program at a time. On the other hand, system virtual machines provide a complete system environment by virtualizing the ISA layer (Berghmans, 2010). System virtual machines allow a physical hardware system to be shared among multiple, isolated guest operating systems simultaneously (Berghmans, 2010). These system virtual machines are mainly hypervisors such as Xen, VirtualBox and VMWare.

Just like virtual machines, virtualization per se can also be divided into different types, namely full virtualization, paravirtualization and hybrid-virtualization.

Full virtualization is the most common virtualization mode and implies that guest operating systems are not aware that they are being virtualized; therefore, there is no direct communication between the VMs and the hypervisor (Rose, 2004). This mode has the benefit of allowing unmodified operating systems to be virtualized (Barham et al., 2003), so it works even with closed-source OSs, such as Windows.

The second type of virtualization – paravirtualization – requires the kernel of guest operating systems to be modified in order to support virtualization (Rose, 2004). This is usually done by removing privileged instructions and replacing them with hypercalls (Barham et al., 2003) that ask the hypervisor directly for resources.

This virtualization mode has the benefit of having better performance (Barham et al., 2003), result of the direct communication path between hypervisor and VM.

The last type of virtualization – hybrid virtualization – combines the two aforementioned types of virtualization and takes the best of them into one single mode of virtualization. It employs para-virtualization for I/O, interrupt controllers, and timer to simplify the system and optimize performance. (Nakajima and Mallick, 2007). For CPU virtualization, it allows one to use the same code as in the original kernel (Nakajima and Mallick, 2007).

3 PAST CHALLENGES

In this section, we present past challenges to virtualization. While past obstacles once proved to be serious limitations to widespread adoption and merited efforts from researchers and industry to be overcome, they are now a solved question, whereas present obstacles are still a subject of active research and pose a risk for the remaining users that have not yet migrated to virtualization, such as, the owners of

highly-critical, dependable or secure systems.

One of the most central obstacles faced by virtualization since its ascension to widespread popularity, was the fact that the x86 processor architecture was not fully virtualizable using the common approaches at the time (Barham et al., 2003) (e.g., trap-and-emulate). In its origin, virtualization was used in the scope of massive mainframes, and its application to commodity hardware was not a main goal.

Trap-and-emulate technique requires the CPU generates a trap whenever an attempt to execute a privileged instruction in user mode is made. The biggest problem is that x86 CPUs did not generate those traps, so the hypervisor did not have a way to know when the guest OS tried to execute a privileged instruction. Instead, the execution of that instruction simply failed silently, and the guest OS was mistakenly led to think its privileged instruction execution was successful thus reacting accordingly.

To solve this issue, researchers have come up with a virtualization technique called binary translation. This technique uses a new piece of software to detect the privileged instructions and replace at run-time that code with a less privileged one but which tries to do the same thing (Adams and Agesen, 2006).

With this new technique the x86 architecture becomes fully virtualizable, because the hypervisor is not dependent on the traps generated by the CPU to know when the guest OS tries to execute a privileged instruction. Instead, the hypervisor will spy the execution flow of the guest OS. In fact, VMWare ESXi (a well-known bare-metal/type 1 hypervisor from VMWare) uses binary translation

("Virtualization system including a virtual machine monitor for a computer with a segmented architecture," 1998).

To summarize, researchers have used several techniques, such as binary translation, paravirtualization and more recently, hardware-assisted virtualization, to make both x86 and x86_64 architectures fully virtualizable.

4 OPEN VIRTUALIZATION CHALLENGES

In this section we explore the open challenges faced by virtualization, that represent the most active research areas under the main topic of virtualization. We group the challenges in the following topics: nested virtualization, reliability and security.

4.1 Nested Virtualization

Nested virtualization refers to the ability of running a virtual machine within another, having this general concept extendable to an arbitrary depth (Tan et al., 2012) ("Virtualization," 2018).

In other words, nested virtualization refers to running one or more hypervisors inside another hypervisor ("Virtualization," 2018).

It poses an open challenge because most hypervisors still do not support this feature, and those who do often have performance and stability problems. Nested virtualization has a broad range of applications, from testing, to security (Greamo and Ghosh, 2011) and fault tolerance (Cully et al., 2008) (Tan et al., 2012).

In the case of fault tolerance, it is used by several tools, like TinyChecker (Tan et al., 2012) and HyperFresh, (Bagdi et al., 2017b) that protect against both software (e.g., hypervisor) and hardware failure.

In security, nested virtualization can be used by antivirus software (Greamo and Ghosh, 2011) to execute malicious software in a sandbox that is isolated from the original system. We can have our antivirus running in a virtualized environment and it can use another level of virtualization (nested virtualization) to test whether a software is malicious or not. Figure 3 shows a classical setup which is using nested virtualization. Right above the hardware we have our first hypervisor (which is known as a Layer 0 hypervisor), then above this hypervisor we have VMs (which are called Layer 1 VMs) and inside those VMs we have another hypervisor (which does not need to be the same as the first one and which is called Layer 1 hypervisor) and finally above the second hypervisor we have our nested VMs (which are called Layer 2 VMs).

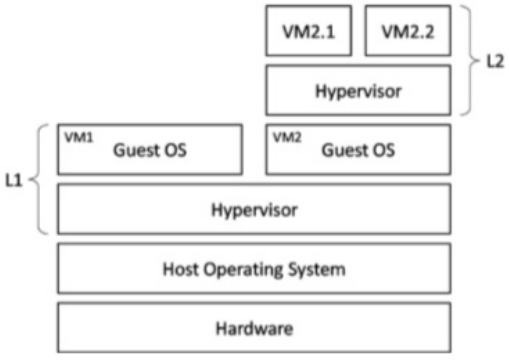

Figure 3: Nested virtualization setup (Berghmans, 2010).

However, nested virtualization has drawbacks in terms of performance, which arise due to the multiple layers of virtualization and the increase in *vmentry* (flow of execution goes from the hypervisor to the VM) and *vmexit* (flow of execution goes from the VM to the hypervisor) events (Michael et al., 2013). If nested virtualization is implemented poorly, we can have a huge decrease in performance compared to only one degree of virtualization, because of the huge number of *vmentry* and *vmexit* (Michael et al., 2013). Modern techniques that promise to increase this aspect, try to decrease the number of *vmentry* and *vmexit* (Michael et al., 2013). Figure 4 shows the execution flow of a nested virtualization setup and it shows the *vmentry* and *vmexit* in a nested virtualization environment.

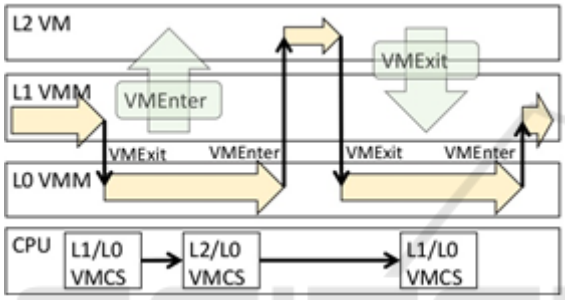

Figure 4: Flow of execution in a nested virtualization setup ("Unsafe Nested Virtualization on Intel CPU", 2018).

4.2 Reliability

An obstacle to reliability that is inherent to virtualization is related with the fact that multiple clients become consolidated over the same physical hardware and virtualization software (e.g., hypervisor). This means that, not only the physical hardware itself but also the virtualization software, becomes a single point-of-failure (SPOF). When one of these components fails (e.g., due to a transient hardware fault affecting the CPU, or a software bug in the hypervisor that is activated) there is the possibility that multiple (or even all) of the clients are affected, thereby amplifying the impact of fault.

To reduce a part of the problem (i.e., the fact that the hypervisor becomes a SPOF), systems administrators can use tools like TinyChecker (Tan et al., 2012) or HyperFresh (Bagdi et al., 2017b), which have been specifically designed to solve this issue. It follows the principle that, as the upper hypervisor is a complex piece of software it is more likely to have software bugs, TinyChecker adds another layer below the original hypervisor, which monitors the state of the hypervisor and takes corrective measures when required. Since TinyChecker is significantly smaller (with respect to lines of code) than any other hypervisor, it is expected to have fewer software bugs and therefore becomes a smaller SPOF.

TinyChecker interposes the control and data exchange between the hypervisor and the guest VMs, so that it can transparently detect hypervisor's failure and recover the system without losing the on-going work in per-exit level (Tan et al., 2012) (per-exit level means a VM does not lose its data when TinyChecker reboots the buggy hypervisor, because the state of the VM is held in the memory of TinyChecker). When a failure is detected, TinyChecker will reboot the hypervisor while preserving the state of the VMs in memory (Tan et al., 2012). This can be achieved by the three parts of TinyChecker: access recorder, memory protector and failure detector.

The access recorder responds for recording the entire communication context between VMs and hypervisor (Tan et al., 2012). The duty of the memory protector is to guarantee the integrity of the critical data (Tan et al., 2012). During the TinyChecker initialization stage, the critical area in memory will be set to readable but non-writable to the hypervisor (Tan et al., 2012). The critical areas in hypervisor are the metadata for managing VMs, such as p2m table or running domain list. VM's memory is another critical area which has also been protected by TinyChecker after its allocation(Tan et al., 2012). In short, the memory protector has protected the critical memory from hypervisor's arbitrary modification (Tan et al., 2012).

The failure detector is the core part of TinyChecker which is going to detect and confirm the hypervisor failure's occurrence (Tan et al., 2012). This module works with information from both the access recorder and the memory protector. Figure 5 shows in a graphical way, the working mode of TinyChecker described above.

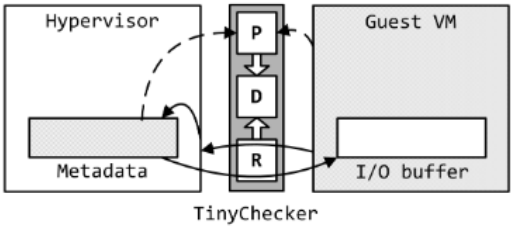

Figure 5: TinyChecker setup (Tan et al., 2012).

To solve the second and last part of the problem (i.e., the fact that the hardware infrastructure

becomes a SPOF), systems administrators can use tools like Remus (Cully et al., 2008).

Remus was created primarily with the intention of providing generic fault tolerance against hardware faults to non-virtualized workloads. Soon its performance caught the attention from users of virtualization, which integrated Remus into their existing systems. Remus takes advantage of virtualization to encapsulate the workload into a VM and easily migrate it across different physical hosts.

It provides an extremely high degree of fault tolerance, to the point that a running system can transparently continue execution on an alternate physical host in the face of failure with only seconds of downtime, while completely preserving guest state such as active network connections (Cully et al., 2008).

Remus' approach encapsulates protected software in a virtual machine, asynchronously propagates changed state to a backup host and uses speculative execution to concurrently run the active VM slightly ahead of the replicated system state (Cully et al., 2008). Remus provides OS- and application-agnostic high availability on commodity hardware (Cully et al., 2008), thus being a very cheap solution to deploy hardware high availability. Remus' three main goals are: *generality*, *transparency* and last but not the least, *seamless failure recovery* (Cully et al., 2008).

The first one aims at giving high availability regardless of the application being protected or the hardware on which it runs (Cully et al., 2008). The second goal aims at giving high availability without having to either modify the application or the OS source code (Cully et al., 2008), because it might not even be available for modification. The last one states that no externally visible state should ever be lost in the case of single-host failure (Cully et al., 2008). Figure 6 explains, in a graphical manner, the working mode of Remus described above.

The conjunction of the aforementioned tools enables an improvement in reliability by taking advantage of virtualization in varied manners. Nevertheless, the reliability of virtualized systems is not a closed topic and new approaches are being investigated that aim to provide better protection at a lower performance cost.

4.3 Security

With the consolidation of multiple clients in the same hardware, the possibility for side-channel attacks between virtual machines is not negligible (Bazm et al., 2017).

In fact, several papers have already demonstrated attacks that successfully extract sensitive information (e.g., cryptographic keys) through timing attacks (Ristenpart et al., 2009), or that cause performance degradation through resource exhaustion (Zhang et al., 2017). Adopted solutions to these problems can be applied at various levels, ranging from hardware to the application (Bazm et al., 2017), however the hypervisor-level is perhaps the most adequate, as it can provide mitigation that covers all the clients but does not need their interaction (i.e., the client does not need to change his application to protect against these attacks). Examples of solutions that have been applied to hypervisors are locking cache lines (Costan and Devadas, 2016) or page coloring (Wang and Lee, 2008) (Shi et al., 2011) (Jin et al., 2009).

Another security problem which is characteristic to cloud computing, is the possibility for a rogue cloud provider to eavesdrop or manipulate a client's virtual machine without his knowledge (Claycomb and Nicoll, 2012). To enable computation with some integrity and confidentiality guarantees, even in presence of an untrusted hypervisor, secure enclaves, powered by hardware extensions (Intel SGX (Costan and Devadas, 2016) and AMD SEV (AMDSEV,

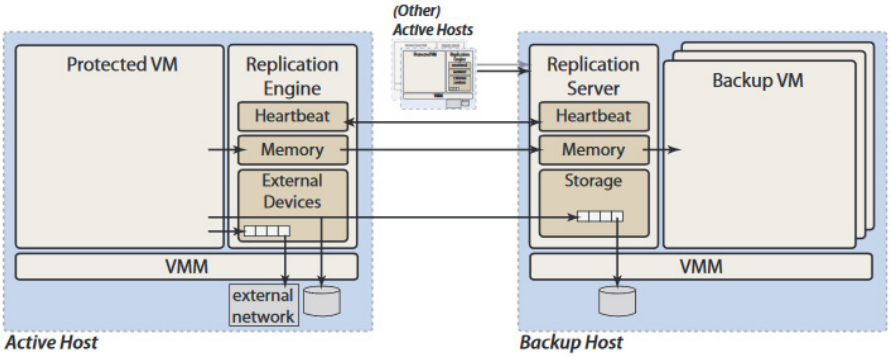

Figure 6: Remus high-level architecture (Cully et al., 2008).

2018)), have been proposed and are an active area of research by the industry.

These enclaves shift the trust from the cloud operator to the hardware manufacturer and use cryptographic algorithms to attest to the user that the code and data that he provided is being executed in a secure enclave that cannot be probed by external agents without being noticed. As disadvantages they usually bring significant performance overheads and are nonetheless susceptible to some attacks (Weichbrodt et al., 2016).

5 CONCLUSIONS AND FUTURE WORK

Virtualization is at an advanced stage now, however there is still much to do in order to make this technology safer, faster, easier and better.

In this paper, we surveyed virtualization, with a main emphasis on its past and current challenges as well as solutions, to help in the improvement of this technology, giving a view of what already exists, what is being currently done in the field and what can be done in the future.

Virtualization will continue to increase in security, performance and popularity in the nearby future. The future work in the area will surely fall under one of the previous mentioned topics: VM protection against failure (either by improving the already existing tools or by creating new ones); VM performance (mainly through the improvement of the existing virtualization techniques); cloud virtualization and data security (either by improving the existing tools or by implementing new ones).

REFERENCES

Aboulnaga, A., n.d. Virtualization and Databases: State of the Art and Research Challenges 45.

Adams, K., Agesen, O., 2006. A comparison of software and hardware techniques for x86 virtualization. *ACM SIGARCH Comput. Archit. News 34, 2–13*.

AMDSEV - https://github.com/AMDESE/AMDSEV (accessed 26/04/2018)

Bagdi, H., Kugve, R., Gopalan, K., 2017a. HyperFresh: Live Refresh of Hypervisors Using Nested Virtualization, in: *Proceedings of the 8th Asia-Pacific Workshop on Systems. ACM, p. 18*.

Bagdi, H., Kugve, R., Gopalan, K., 2017b. HyperFresh: Live Refresh of Hypervisors Using Nested Virtualization, in: *Proceedings of the 8th Asia-Pacific Workshop on Systems. ACM, p. 18*.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., 2003. Xen and the art of virtualization, in: *ACM SIGOPS Operating Systems Review. ACM, pp. 164–177*.

Bazm, M.-M., Lacoste, M., Südholt, M., Menaud, J.-M., 2017. Side Channels in the Cloud: Isolation Challenges, Attacks, and Countermeasures.

Berghmans, O., 2010. Nesting virtual machines in virtualization test frameworks (PhD Thesis). Master's thesis, *University of Antwerp.*

Claycomb, W.R., Nicoll, A., 2012. Insider threats to cloud computing: Directions for new research challenges, in: Computer Software and Applications Conference (COMPSAC), *2012 IEEE 36th Annual. IEEE, pp. 387–394*.

Colsani, G., Giusti, G., Pássera, P., Protti, D., (2008) 'Virtualization Technology Introduction'.

Costan, V., Devadas, S., 2016. Intel SGX Explained. IACR Cryptol. *EPrint Arch. 2016, 86*.

Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., Warfield, A., 2008. Remus: High availability via asynchronous virtual machine replication, in: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. San Francisco, pp. 161–174*.

Greamo, C., Ghosh, A., 2011. Sandboxing and Virtualization: Modern Tools for Combating Malware. IEEE Secur. Priv. 9, 79–82. https://doi.org/10.1109/MSP.2011.36

Heiser, G., 2008. The role of virtualization in embedded systems, in: Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems. *ACM, pp. 11–16*.

Herbst, N.R., Kounev, S., Reussner, R., n.d. Elasticity in Cloud Computing: What It Is, and What It Is Not 6.

Jin, X., Chen, H., Wang, X., Wang, Z., Wen, X., Luo, Y., Li, X., 2009. A simple cache partitioning approach in a virtualized environment, in: Parallel and Distributed Processing with Applications, *2009 IEEE International Symposium On. IEEE, pp. 519–524*.

Joe, H., Jeong, H., Yoon, Y., Kim, H., Han, S., Jin, H.W., 2012. Full virtualizing micro hypervisor for spacecraft flight computer, in: 2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC). Presented at *the 2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC), pp. 6C5-1-6C5-9*. https://doi.org/10.1109/DASC.2012.6382393

Lacoste, A.W.M., Debar, H., n.d. KungFuVisor: Enabling Hypervisor Self-Defense.

Lee, C., Kim, S.-W., Yoo, C., 2016. VADI: GPU virtualization for an automotive platform. *IEEE Trans. Ind. Inform. 12, 277–290*.

Lombardi, F., Di Pietro, R., 2011. Secure virtualization for cloud computing. *J. Netw. Comput. Appl., Advanced Topics in Cloud Computing 34*, 1113–1122. https://doi.org/10.1016/j.jnca.2010.06.008

Michael, D. D. I., Harper, R. A., Liguori, A. N., 2013. Nested virtualization performance in a computer system.

Nakajima, J., Mallick, A.K., 2007. Hybrid-virtualization—enhanced virtualization for Linux, in: *Proceedings of*

the Linux Symposium. Citeseer, pp. 87–96.

Ristenpart, T., Tromer, E., Shacham, H., Savage, S., 2009. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds, in: *Proceedings of the 16th ACM Conference on Computer and Communications Security. ACM, pp. 199–212*.

Rose, R., 2004. Survey of system virtualization techniques.

Shi, J., Song, X., Chen, H., Zang, B., 2011. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring, in: Dependable Systems and Networks Workshops (DSN-W), *2011 IEEE/IFIP 41st International Conference On. IEEE, pp. 194–199*.

Tan, C., Xia, Y., Chen, H., Zang, B., 2012. Tinychecker: Transparent protection of vms against hypervisor failures with nested virtualization, in: *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference On. IEEE, pp. 1–6*.

Virtualization system including a virtual machine monitor for a computer with a segmented architecture, 1998.

Virtualization- https://en.wikipedia.org/wiki/Virtualization (accessed 16/04/18).

Wang, Z., Jiang, X., 2010. HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity, in: 2010 IEEE Symposium on Security and Privacy. *Presented at the 2010 IEEE Symposium on Security and Privacy, pp. 380–395. https://doi.org/10.1109/SP.2010.30*

Wang, Z., Lee, R. B., 2008. A novel cache architecture with enhanced performance and security, in: *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, pp. 83–93*.

Weichbrodt, N., Kurmus, A., Pietzuch, P., Kapitza, R., 2016. AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves, in: *European Symposium on Research in Computer Security. Springer, pp. 440–457*.

Wolf, M., n.d. Virtualization Technologies for Cars 10.

Xing, Y., Zhan, Y., 2012. Virtualization and cloud computing, in: Future Wireless Networks and Information Systems. *Springer, pp. 305–312*.

Yan, Q., Yu, F. R., Gong, Q., Li, J., 2016. Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Commun. Surv. Tutor. 18, 602–622*.

Unsafe Nested Virtualization on Intel CPU - https://www.slideshare.net/DeepTokikane/unsafenested-virtualization-on-intel-cpu-83409391 (accessed 25/04/2018).

Zhang, T., Zhang, Y., Lee, R. B., 2017. DoS Attacks on Your Memory in Cloud, in: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. ACM, pp. 253–265*.