Towards Fault Propagation Analysis in Cloud Computing Ecosystems

Luigi De Simone
DIETI, Federico II University of
Naples, Italy
luigi.desimone@unina.it

***Abstract*—Nowadays, *Cloud Computing* is a fundamental paradigm that provides computational resources as a service, on which users heavily rely. Cloud computing infrastructures behave as an *ecosystem*, where several actors play a crucial role. Unfortunately Cloud Computing Ecosystems (CCEs) are often affected by outages, such as those experienced by Amazon Web Service in the last years, that result from component faults that propagate through the whole CCE. Thus, there is still a need for approaches to improve CCEs' reliability. This paper discusses both existing approaches and open challenges for the dependability evaluation of CCEs, and the need for novel techniques and methodologies to prevent fault propagation within CCEs as a whole.**

Keywords*—*Cloud Computing; Virtualization; Hypervisors; Containers; Fault Injection Testing; Fault Propagation

I. Introduction

Cloud Computing is strongly reshaping the IT industry panorama, actually making new research topics and challenges for the near future. Such impact was shown in a recent IDC market analysis [1], which predicted that the cloud software market will surpass $75B by 2017, attaining a five year compound annual growth rate of 22% in the forecast period. To support such view, starting from 2012, we are witnessing the proliferation of new Infrastructures as a Service (IaaS), such as *Google Compute Engine*, *HP Cloud*, and *Microsoft Windows Azure*, along with the well-known *Amazon Web Services (AWS)* IaaS. Cloud computing was born to avoid the need for many corporations to build and manage their own IT data centers: it is a new paradigm meant to provide computational resources, in such a way to allocate and deallocate them on demand, based on a pay-per-use business model, avoiding expensive hardware platforms and big initial capital costs. Such computing resources include Internet services, storage facilities, compute power, all provided just like a service. In this panorama, several research projects have been pursuing novel methodologies, architectures and tools to achieve trustworthy cloud services [2] [3].

A cloud computing infrastructure is a complex *ecosystem*, in which several actors play a crucial role (see Fig. 1). These ecosystems are highly distributed, consist of heterogeneous hardware and software components, and are expected to provide highly-available services requested by millions of user in parallel. Failures in such a complex system are inevitable, because of too many factors are outside of our control. They include not only hardware faults (e.g., CPU, memory, disk, network faults) but also software faults (i.e., bugs) and operator faults (e.g., a software misconfiguration). As the reader might guess, these problems are exacerbated by the fact that *Cloud Computing Ecosystems* (CCEs) are mostly driven by software, thus the likelihood of incurring in software and operator faults is very high, leading cloud services to outages, unresponsiveness and data losses.

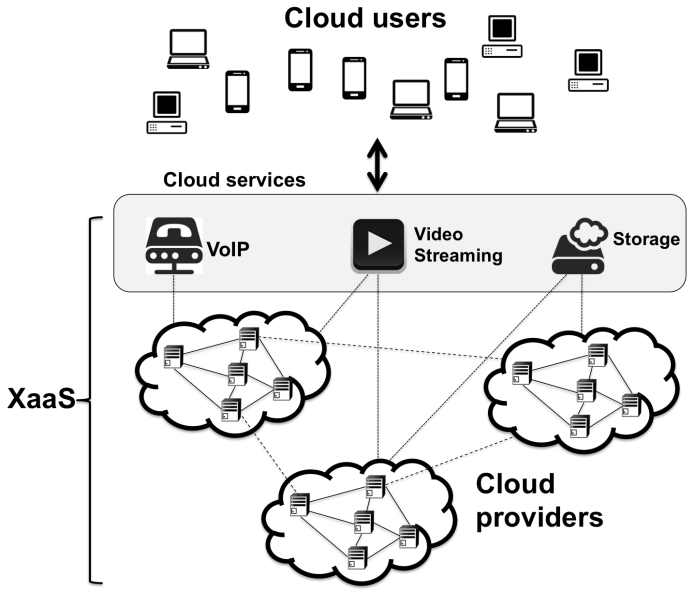

Fig. 1. Cloud Computing Ecosystem (CCE)

In the last years, both industry and researchers are debating about how much we can trust in cloud computing [2]. Recently, several cases of cloud failures such as Amazon Web Services (AWS) [4] (a race condition in the code on the servers which manage data storage), Microsoft Azure [5] (a network device misconfiguration) and Google Docs [6] (software fault in the memory management component), have raised concerns on cloud computing. From the cloud provider perspective, failures can lead to billion of money losses, just for an hour of offline time [7]. These outages result from faults that occur in a component of the cloud system, and that propagate through the entire ecosystem.

Delivering a *trustworthy* cloud computing service (ranging from IaaS to SaaS [8]) is a priority. Indeed, if we think that a lot of organizations and companies rely on cloud computing services (e.g., in traffic management system for decision support [9], in finance [10], in healthcare [11], etc.), cloud services will become more critical in the near future!

Failures are random in time and it is really difficult to predict them. In addition, many cloud services depend on third-party services over the Internet, exposing cloud services to cascading failures. Given such an unpredictable and dynamic scenario, it is extremely important to know the nature of faults and their propagation within CCE, in order to deliver dependable cloud services. Actually, addressing dependability issues early in design and making decisions to reduce the impact of failures of a specific service, are important benefits that cloud developers have to consider. Thus, it is necessary to conduct research and develop techniques and methodologies that allow us to build countermeasures against faults, with the purpose of preventing fault propagation within a CCE and, ultimately, of avoiding failures of the CCE as a whole.

In this context, my PhD thesis proposal is to:

> *Analyze and understand fault propagation within Cloud Computing Ecosystems (CCEs), in order to prevent and mitigate failures of CCEs as a whole.*

The main goal of my work will be to provide techniques and methodologies for understanding how faulty components in the CCE can affect other components and the overall CCE services, and for predicting and quantifying the impact of fault propagation on the CCE as a whole. This will enable CCE architects to identify dependability bottlenecks in their infrastructures, will lead them at introducing fault-tolerance mechanisms and at making dependability-oriented design choices. Furthermore, knowledge from fault propagation analysis can help cloud developers both to minimize the amount of time necessary to recover cloud services from failures and to make the time to failure of cloud services as long as possible. My work will leverage tools and techniques for both fault injection and monitoring of cloud systems, and will enhance them to provide an effective and easy-to-use experimental framework for evaluating CCEs dependability.

This paper will discuss open challenges behind the evaluation of CCE dependability. It is organized as follows: Section II overviews CCEs, describing each component and how they can be source of failures; Section III presents open issues in testing and validation of CCEs, focusing on the definition of fault models (III-A), on tools, techniques and methodologies needed to analyze fault propagation (III-B), and on how to test the cloud management software (III-C); finally, Section IV presents the conclusion and future directions.

II. Cloud Computing Ecosystems

As mentioned in the introduction, the CCEs is a complex infrastructures. According to NIST service models [8], we can see them as consisting of a stack of several layers, which are supplied by a cloud provider depending by the specific model (see Fig. 2).

Usually, applications in the cloud are deployed in a distributed way, that is, on several interconnected computers, which belong to data centers that are geographically distributed and managed by third parties. Furthermore, each application entity communicates with others entities leveraging middleware technologies such as Web Services and Message Queues. Finally, application entities rely on several core services such as storage, database, load balancing, etc., provided by the cloud infrastructure where application is deployed.

In particular, the *virtualization layer* is the cornerstone of such infrastructures. Indeed, it allows to abstract the specific details of physical resources (e.g., CPUs, network and storage devices, etc.), providing and sharing them for applications at higher levels. This perspective completely changes the way we see a physical machine (or a pool of physical machines), making it a simply soft component to use and manage at the push of a button. A main concept within virtualization is the *Virtual Machine* (VM), which provides an isolated execution context running on top of the underlying physical resources. The *Hypervisor* actually runs VMs and it is also responsible of coordinating multiple VMs in order to discipline the access to the underlying CPU, memory and I/O resources. Furthermore, in CCE a central role is given to cloud managements tools (e.g., Openstack [12]) that orchestrate and control compute, storage and networking resources in order to achieve the essential characteristic of a cloud computing environment (see [8]).

Currently, in the virtualization panorama there is a spectrum of approaches, but mainly *Hypervisor-based* and *Container-based* (or Operating system-level) virtualization are actually in use in CCEs.

Hypervisor-based virtualization technologies provide an environment that allows for a guest operating system to run on top of a so-called Hypervisor. As mentioned above, hypervisor is a piece of software that runs multiple virtual machines, each of which run a guest operating system. There are two main types of Hypervisor-based virtualization:

- *Full virtualization*, in which the hypervisor completely isolates the guest OS and completely abstracts hardware resources to guest. The task of the hypervisor is to emulate privileged CPU instructions and I/O operations, and to multiplex resources between concurrent VMs. Examples of hypervisors with full virtualization are VMware ESXi [13], KVM [14], and Microsoft Hyper-V [15].
- *Paravirtualization* (partial virtualization), in which the guest OS is aware that it is running in a VM, that is, it is modified in order to communicate directly with the hypervisor (via a system call mechanism called *hypercalls*) to achieve better performance. An example of hypervisor with paravirtualization is Xen [16].

On the other side of spectrum, **Container-based** virtualization, also called *Operating System-level* virtualization, allows to runs multiple guest OSes without hardware virtualization. OSes have been already designed to provide resource isolation, but among processes. Thus, the idea is to use a traditional OS to run virtual appliances, by enhancing the (host) kernel OS to provide isolation between guest applications that run in so-called *containers*. A container is not a Virtual Machine in the traditional sense, but it is an environment that allows to runs a guest application with its own filesystem, memory, devices, processes, etc., leveraging kernel OS host process isolation (e.g., *namespace* in Linux [17]) and resource management capabilities (e.g., *cgroups* in Linux [18]). Examples of container-based virtualization technologies are LXC (LinuX Container [19]), Docker [20], OpenVZ [21].

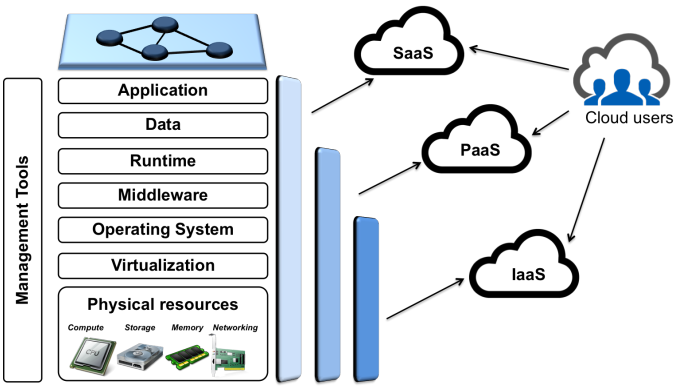

Fig. 2. Layers in a Cloud Computing Ecosystem

Virtualization is a major, but not the sole, ingredient for CCEs. Among cloud computing technologies, **cloud management tools** are a fundamental component. Management tools allow to perform and automate many tasks in CCEs, over hundreds or even thousands of services, resources and VMs. Nowadays, there is a pletora of tools that help to enhance the reliability, availability and, more in general, efficiency of cloud service management. Such tools allow to manage:

- VMs and resource allocation, for example how many CPUs are assigned to a VM, or how much memory is reserved for.
- Virtual-physical resource mapping, in order to track resource usage;
- Monitoring and fault diagnosis, that allows to gather basic information (e.g., system and network information) during an issue, and supports system administrators in the troubleshooting process;
- Fault detection and recovery, in order to react as soon as possible to faults using policies defined by system administrators;
- Reconfiguration, in order to perform resource allocation in response to changes in workload increase/decrease;
- Checkpointing and rollback, that are techniques to enhance the VM availability, saving the state of a VM and then migrate such VM to a remote node within the cloud; finally, restore the VM state.
- Migration, in order to move data, applications (often within a VM), from a cloud environment to another;
- Update/upgrade, in order to upgrade cloud software without service availability loss.

III. Challenges in testing Cloud Computing Ecosystems

As mentioned before, the problem of testing and validating the CCE is a big challenge. In fact, since a CCE is consisting of many elements, it is hard to understand how individual elements impact on the reliability of the cloud service/infrastructure, and how to prevent failures. Recent studies have been done in testing of cloud-based applications [22], using cloud platforms to perform testing of application [23], and studies related to verification of cloud services [24]. Furthermore, other studies [2] addressed the problem of reliability of cloud infrastructure.

Nevertheless, there is still a need for approaches specifically focused on the reliability of cloud services and infrastructures against faults. We foresee the following challenges and open problems that needs to be tackled in testing CCEs as a whole.

A. Define Fault Models and Reliability measures for CCEs

In recent years, several studies and tools have proved that Fault Injection Testing is a valuable approach for assessing fault-tolerant systems [25]. Fault injection is an approach in which we deliberately introduce faults in a system. As mentioned before, this approach can assess the robustness and performance of a system in the presence of faults, and to state if fault tolerance algorithms and mechanisms are effective.

In the CCE context, some studies have faced with testing of components that constitute such ecosystem, mainly focusing on network applications (*D-Cloud* [26]; *DS-Bench Toolset* [27], *Chaos Monkey* [28]), Hypervisors (*CloudVal* [29]), and cloud management stack (*Openstack resilience study* [30], *PreFail* [31]). Table I shows a comparison between fault injection approaches and the related tools mentioned above. All these tools are mostly focused on injection of hardware faults, for example CPU (e.g., corrupt registers), memory (e.g., bit error), network controller (e.g., bit error in packet), and hard disk (e.g., fault in a specific sector). Moreover, these tools are not meant for the evaluation of CCE architecture as a whole.

In addition to hardware faults, a system can be affected also by software faults and configuration faults. A ***Fault Model*** is a description of the types of fault that the system is expected to experience during runtime. This description model is useful to help the design of fault-tolerant and robust systems against the faults within the model. Furthermore, a fault model is fundamental to drive fault injection tests which aim to verify fault-tolerance and robustness properties. The fault model should describe which entities, in terms of environment, components and services, can be affected by faults, and how the faulty component behaves. Finally, a fault model must be state *what* to inject, *when* to inject and *where* to inject. It is very challenging to define realistic fault models that takes into account all the specifics of each CCE elements, given the complexity of these systems. Furthermore, in the CCE context, software and operator faults have not yet been studied deeply, thus there is another big question to answer.

Another important aspect is ***Measures*** that aim to express quantitative indicators (e.g., performance and the responsiveness) of the cloud system. For instance, when testing a fault tolerance mechanism or algorithm (e.g., a system with redundant components, or a fault-tolerant network protocol), measures can express the probability that the system recovers from a fault (e.g., injected faults do not lead to system unavailability), and performance levels in the presence of faults (e.g., a server can sustain a high rate of served requests or messages even in the presence of injected faults).

TABLE I. COMPARISON OF FAULT INJECTION APPROACHES.

Approach	Tool	Target	Faultload	Injection technique	Examples of results
Fault Injection Testing of Virtual Machines	D-Cloud [26] and DS-Bench Toolset [27]	Server software (e.g., web applications)	Network, disk, memory faults	Emulation of faulty devices; VM memory corruption	Validation of performance levels under faults
	Chaos Monkey [28]	Virtual instances during runtime	CPU, disk, network faults	Executing scripts that simulates a fault on target machine	Terminates over 65,000 instances running in Netflix production and testing environments, detecting many failure scenarios
Fault Injection Testing of Cloud Management Software	PreFail [31]	Distributed filesystems and algorithms (e.g., HDFS, ZooKeeper)	Network and disk faults; Process crashes	API exception injection	Robustness of recovery protocols
	Openstack resilience framework [30]	OpenStack	Service crash and Network partition	API exception injection	Improvement of robustness
Fault Injection Testing of Hypervisors	CloudVal [29]	Hypervisors (e.g., Xen, KVM)	CPU, memory, VM faults	Memory corruption	Improvement of VM isolation

In order to evaluate the degree of trustworthiness of the CCE we need to consider that there are many metrics already adopted for monitoring, administration, and accounting purposes. Given a Service Level Agreement (SLA) requirements to be met, there are Key Performance Indicators (KPIs) that tell you how the service is performing according to certain parameters. Moreover, there are several detailed metrics collected from network services (e.g., round trip time, response time, packet loss rate), storage services (e.g., throughput, free disk space, average read/write speed), compute services (e.g., CPU utilization, memory consumption). Which metrics to use for analyzing dependability? For sure, some of the metrics above will still play a role, to provide an user- and administrator-oriented evaluation of CCEs. Nevertheless, new measures will be necessary at a more fine-grained level, to monitor the behavior of each individual component, in order to spot the effects of faults during the experiments and metrics related to the fault propagation. Furthermore, we will need to consider measures related to fault tolerance, such as fault detection coverage and latency, migration and checkpointing time, etc.

In such scenario we have to get two kind of measures:

- **System-level measures** to evaluate the CCE as a whole. This is important in the perspective of *validation* and also to meet SLA specific requirements [32]. For example, let's consider a scenario in which we want to "cloudify" the network elements that constitute an *IP Multimedia Subsystem* (IMS) [33], that is, adopting virtualization technologies to virtualize the network elements within an IMS. This scenario is well described in NFV Use Cases document [34], leveraging *Network Function Virtualization* (NFV), an emerging solution that exploits virtualization technologies to turn network equipments (i.e., middleboxes) into virtual entities, in order to reduce costs, improve efficiency and scalability, reduce time-to-market [34]. IMS is one of the most important element in 3G network and essentially delivers IP multimedia services, such as video conferencing, IPTV, VoIP, etc., with the main goal of achieving a specific Quality of Service (QoS). In this scenario i would want to get measures related to specific IMS KPI [35] (e.g., Call Drop Rate of IMS Sessions, Session Setup Time, Mean Session Utilization). Such metrics are important to conduct empirical analysis;
- **Internal measures**, that is, those related to individual cloud stack components (see Fig. 2). Such measures are necessary to test and analyze in depth CCE, in order to learn how to design and configure them optimally. For example, related to virtualization layer, we need to assure that resources (e.g., number of CPUs assigned to a VM) are efficiently used, to achieve SLA assurance, and that faults in individual components are readily detected and isolated.

Experimenting and making measurements in CCEs can be challenging, due to the quantity and the heterogeneity of hardware and software technologies that are involved. For instance, in the IP Multimedia Subsystem scenario mentioned above, the CCE can include proprietary components that could not be supported by existing monitoring tools. Moreover, we would need to collect measures with very low intrusiveness, since these systems are performance-sensitive and any instrumentation can potentially perturb their behavior. Finally, measures often depend on the specific cloud scenario, and need to be customizable by the experimenter.

B. Fault Propagation analysis in CCEs

It is clear that having knowledge about how faults propagate in a system is fundamental. In the past, a lot of fault propagation studies were done in complex system, much more related to hardware systems and embedded systems [36]. Also in complex software systems, like OSes [37] [38] and real-time OSes [39] a lot of work have been done.

In the context of CCE, in order to perform a reliability evaluation there is a need to study, possibly with experimental methods, how faults in individual components propagate and impact on CCE as a whole. In particular, we need:

- **Comprehensive Tools/Framework for execution monitoring**. As mentioned in the previous section, measures are fundamental to conduct an empirical evaluation of CCE. Moreover, given the high number of components and of hardware/software layers in cloud systems, it is important to monitor the reliability at run-time (i.e., during the operational phase), in order to quickly pinpoint the effects and the causes of performance and fault tolerance issues, and to enable a quick recovery. This goal is achieved through reliability monitoring, by leveraging resource and performance data collected from VMs, the OS, and from the hardware.

- **Tools/Framework for data analysis and to define/propose analyses that would be useful for CCE designer/architect**. Analysis of monitoring data allows a variety of objectives, which range from resources and SLA management to troubleshooting [40]. Measurement of workload and quality of service (QoS) parameters collected during the system execution enables resource planning and management. For example, monitored data can be used to establish the optimal number of nodes, CPUs, storage and memory size, that allow copying with a given SLA, to apply corrective measures to the system, or to characterize the state of the system as affected by the workload.

These tools/framework should be developed taking into account key properties such as *usability* (i.e., tools/framework easily and quickly usable by cloud developers, with a high level of compatibility across heterogeneous cloud technologies), *non-intrusiveness* (i.e., tools/framework should not introduce significant perturbation to the system, in order to provide accurate measures), *efficiency* (i.e., tools/framework that achieve meaningful results with a reasonable computational time).

Failures in CCEs may involve fault propagation, and due to complex interactions between different layers, it is very challenging to predict and quantify which is the impact that such a propagation could have on the CCE as a whole.

The idea is to leverage fault injection techniques to conduct such a fault propagation analysis. Fig. 3 shows that we can inject faults (hardware, software and configuration faults) in each layer, to understand how these faults propagate through different components and layers within CCEs. For instance, fault injection can emulate application faults triggered by user data (e.g., customers' data submitted to a data-processing SaaS) [41] [42]. This analysis can give useful information about if there is (or not) a fault propagation path from less critical components/layers to more critical components/layers. Furthermore, we can discover new failure modes, and localize failures to the greatest extent possible. This work aims to develop framework, tools, mechanisms, and algorithms in order to detect faults and prevent their propagation within CCEs.

Important questions need to be answered by fault propagation analysis:

- We can inject faults in whatever layer within CCE stack, and in several components that constitute these layers (see Fig. 2). *What is the components/faults that have the major impact on the CCE? How to quantify that impact?*
- The CCEs should be *self-healing*, that is, an infrastructure that autonomically detects failures and takes proper decisions to react them. Most of self-healing is related to the *resilience* of a system, that is, the capability to adapt properly when facing changes in the environment (i.e., ecosystem) [43]. For example, self-healing actions are VMs migration and checkpointing. *How can we use monitored data to train algorithms and mechanisms for "smart-healing"?*
- A designer/architect can choose various technologies to build a cloud infrastructure. For example, at virtualization layer, he can choose different virtualization technologies, such as hypervisor-based (e.g., VMware ESXI) and container-based (e.g., LXC) virtualization technologies. *How to compare/benchmark alternative design choices in terms of reliability (e.g., degree of fault isolation provided) in order to improve the overall reliability of CCE?*

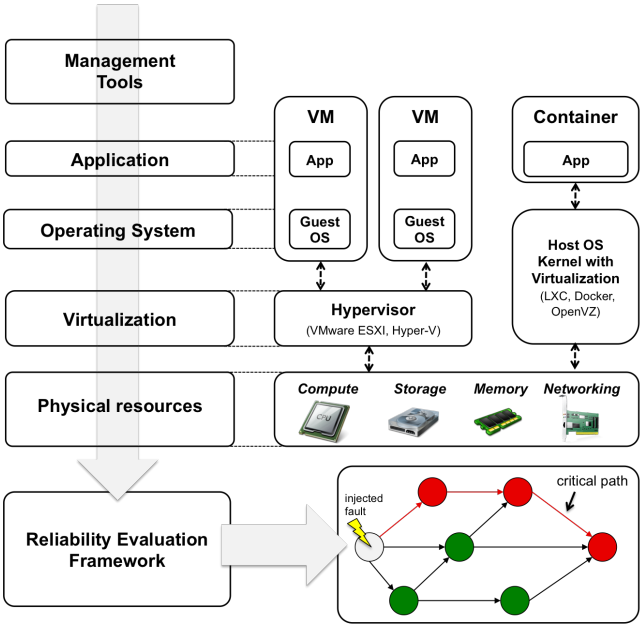

Fig. 3. Fault Propagation in CCE

C. Testing of Cloud Management Software

Nowadays, cloud management and provisioning tools are very popular. For instance, *Puppet* [44] and *Ansible* [45], are widespread open source tools today in the cloud panorama [46]. Cloud computing administrators heavily rely on these tools, whose faults may have critical consequences on the CCEs. The nature of cloud management software is very different from traditional software. Indeed, it is not an elementary procedure, module, or program with simple inputs (integer, strings, etc.), but it is software with several components, which takes in input complex entities like VMs to snapshot/migrate, policies for fault detection and recovery, software updates, network topologies and traffic flows. Given such complexity, we need to ask ourselves: *How we should define meaningful and thorough test scenarios for cloud management software? How we can automate this kind of testing?* Towards these goals, we can exploit fuzzing techniques and search-based algorithms. Furthermore, it is necessary to formalize the input space of this software, the properties to verify, and the measurements that need to be collected and analyzed.

IV. Conclusion

As discussed in this paper, CCEs are very critical, complex and layered infrastructures, and the interactions between their components make testing and validation activities very challenging. Existing studies reveal the necessity of approaches focused on the reliability of cloud services and infrastructures against faults. Fault propagation analysis, leveraging fault

injection techniques, is a promising solution, thus techniques and tools have to be developed in the next future, in order to help validation and testing activities of CCEs as a whole. Due to complexity of layers within CCE, such evaluation may be performed initially by focusing on faults occurring at the lower levels of the cloud stack.

ACKNOWLEDGMENT

This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research.

REFERENCES

[1] IDC and Cisco. Midsize enterprises leading the way with cloud adoption. [Online]. Available: http://share.cisco.com/cloudadoption/

[2] A. Bessani, R. Kapitza, D. Petcu, P. Romano, S. V. Gogouvitis, D. Kyriazis, and R. G. Cascella, "A look to the old-world sky: EU-funded dependability cloud computing research," *SIGOPS Operating Systems Review*, vol. 46, no. 2, pp. 43–56, Jul. 2012.

[3] Microsoft Corporation. Trustworthy computing homepage. [Online]. Available: http://www.microsoft.com/en-us/twc/default.aspx

[4] Amazon.com, Inc. (2011, Apr.) Summary of the amazon ec2 and amazon rds service disruption in the us east region. [Online]. Available: http://aws.amazon.com/message/65648/

[5] Gigaom.com. (2012, Jul.) Windows azure outage hits europe. [Online]. Available: http://gigaom.com/cloud/windows-azure-outage-hits-europe/

[6] A. Warren. (2011, Sep.) What happened to google docs on wednesday. [Online]. Available: http://googleenterprise.blogspot.it/2011/09/what-happened-wednesday.html

[7] David Linthicum. Calculating the true cost of cloud outages. [Online]. Available: http://www.infoworld.com/d/cloud-computing/calculating-the-true-cost-of-cloud-outages-212253

[8] P. M. Mell and T. Grance, "SP 800-145. The NIST Definition of Cloud Computing," NIST, Gaithersburg, MD, United States, Tech. Rep., 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[9] Z. Li, C. Chen, and K. Wang, "Cloud computing for agent-based urban transportation systems," *Intelligent Systems, IEEE*, vol. 26, no. 1, pp. 73–79, Jan 2011.

[10] "Cloud computing for financial markets," White Paper, Cisco Systems, Inc., 2011. [Online]. Available: http://www.cisco.com/web/strategy/docs/finance/cloud_wp_c112D518876.pdf

[11] "Your cloud in healthcare," White Paper, VMware, Inc., 2011. [Online]. Available: http://www.vmware.com/files/pdf/VMware-Your-Cloud-in-Healthcare-Industry-Brief.pdf

[12] Openstack. Openstack. [Online]. Available: http://www.openstack.org/

[13] VMware Inc. VMware ESXi Overview. [Online]. Available: http://www.vmware.com/it/products/esxi-and-esx/overview.html

[14] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proc. Linux Symp.*, vol. 1, 2007, pp. 225–230.

[15] Microsoft Corporation. Hyper-V. [Online]. Available: http://technet.microsoft.com/en-us/windowsserver/dd448604.aspx

[16] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. SOSP*, 2003, pp. 164–177.

[17] LWN.net. Namespaces in operation, part 1: namespaces overview. [Online]. Available: http://lwn.net/Articles/531114/

[18] P. Menage. CGROUPS. [Online]. Available: https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[19] LXC. LXC - Linux Containers. [Online]. Available: https://linuxcontainers.org/

[20] Docker Inc. Docker HomePage. [Online]. Available: https://www.docker.com/

[21] OpenVZ. OpenVZ Main Page. [Online]. Available: http://openvz.org/Main_Page

[22] X. Bai, M. Li, B. Chen, W.-T. Tsai, and J. Gao, "Cloud testing tools," in *Proc. Intl. Symp. SOSE*, 2011, pp. 1–12.

[23] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea, "Cloud9: A software testing service," *SIGOPS Operating System Review*, vol. 43, no. 4, pp. 5–10, Jan. 2010.

[24] S. Bouchenak, G. Chockler, H. Chockler, G. Gheorghe, N. Santos, and A. Shraer, "Verifying cloud services: Present and future," *SIGOPS Operating System Review*, vol. 47, no. 2, pp. 6–19, Jul. 2013.

[25] R. Natella, D. Cotroneo, J. Duraes, and H. Madeira, "On fault representativeness of software fault injection," *Software Engineering, IEEE Transactions on*, vol. 39, no. 1, pp. 80–96, Jan 2013.

[26] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, and M. Sato, "D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology," in *Proc. Intl. Conf. CCGRID*, 2010, pp. 631–636.

[27] H. Fujita, Y. Matsuno, T. Hanawa, M. Sato, S. Kato, and Y. Ishikawa, "DS-Bench Toolset: Tools for dependability benchmarking with simulation and assurance," in *Proc. Intl. Conf. DSN*, 2012, pp. 1–8.

[28] Netflix. The Chaos Monkey. [Online]. Available: https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey

[29] C. Pham, D. Chen, Z. Kalbarczyk, and R. K. Iyer, "CloudVal: A framework for validation of virtualization environment in cloud infrastructure," in *Proc. Intl. Conf. DSN*, 2011, pp. 189–196.

[30] X. Ju, L. Soares, K. G. Shin, K. D. Ryu, and D. Da Silva, "On fault resilience of OpenStack," in *Proc. SOCC*, 2013, pp. 1–16.

[31] P. Joshi, H. S. Gunawi, and K. Sen, "Prefail: A programmable tool for multiple-failure injection," in *Proc. Intl. Conf. OOPSLA*, 2011, pp. 171–188.

[32] L. Sun, J. Singh, and O. K. Hussain, "Service level agreement (sla) assurance for cloud services: A survey from a transactional risk perspective," in *Proc. Intl. Conf. MoMM*, 2012, pp. 263–266.

[33] ETSI 3GPP, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; IP Multimedia Subsystem (IMS); Stage 2," ETSI, Tech. Rep., 2013.

[34] ETSI ISG. Network Functions Virtualisation. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

[35] ETSI 3GPP, "Key Performance Indicators (KPI) for the IP Multimedia Subsystem (IMS)," ETSI, Tech. Rep., 2011.

[36] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, Apr. 1997.

[37] W. Gu, Z. Kalbarczyk, Ravishankar, K. Iyer, and Z. Yang, "Characterization of linux kernel behavior under errors," in *Proc. Intl. Conf. DSN*, 2003, pp. 459–468.

[38] A. Johansson and N. Suri, "Error propagation profiling of operating systems," in *Proc. Intl. Conf. DSN*, 2005, pp. 86–95.

[39] N. Ignat, B. Nicolescu, Y. Savaria, and G. Nicolescu, "Soft-error classification and impact analysis on real-time operating systems," in *Proc. Conf. DATE*, 2006, pp. 182–187.

[40] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, 2013.

[41] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, G. Goel, S. Sarkar, and R. Ganesan, "Characterization of operational failures from a business data processing saas platform," in *Proc. ICSE*, 2014, pp. 195–204.

[42] S. Li, H. Zhou, H. Lin, T. Xiao, H. Lin, W. Lin, and T. Xie, "A characteristic study on failures of production distributed data-parallel programs," in *Proc. ICSE*, 2013, pp. 963–972.

[43] Laprie, J.-C., "From dependability to resilience," in *Proc. Intl. Conf. DSN, Supplemental Volume, Fast Abstracts*, 2008.

[44] Puppet Labs. Puppet Homepage. [Online]. Available: http://puppetlabs.com/

[45] Ansible, Inc. Ansible Homepage. [Online]. Available: http://www.ansible.com/home

[46] Alexander Williams. The Top Open Source Cloud Projects of 2014. [Online]. Available: www.linux.com/news/enterprise/cloud-computing/784573-the-top-open-source-cloud-projects-of-2014