Dependability Evaluation and Benchmarking of Network Function Virtualization Infrastructures

Domenico Cotroneo, Luigi De Simone, Antonio Ken Iannillo, Anna Lanzaro, Roberto Natella
Critiware s.r.l. / Federico II University of Naples, Italy
{cotroneo, luigi.desimone, antonioken.iannillo, anna.lanzaro, roberto.natella}@unina.it

***Abstract*—Network Function Virtualization (NFV) is an emerging solution that aims at improving the flexibility, the efficiency and the manageability of networks, by leveraging virtualization and cloud computing technologies to run network appliances in software. However, the "softwarization" of network functions raises reliability concerns, as they will be exposed to faults in commodity hardware and software components. In this paper, we propose a methodology for the dependability evaluation and benchmarking of NFV Infrastructures (NFVIs), based on fault injection. We discuss the application of the methodology in the context of a virtualized IP Multimedia Subsystem (IMS), and the pitfalls in the design of a reliable NFVI.**

Keywords*—*NFV; NFVI; Fault Injection; Cloud Computing; Virtualization; Dependability Benchmarking; Certification

I. Introduction

Network Function Virtualization (NFV) [1], [2] is an emerging solution to supersede traditional network equipment to reduce costs, improve manageability, reduce time-to-market, and provide more advanced services [3]. NFV will exploit IT virtualization technologies to turn network equipment into *Virtualized Network Functions* (VNFs) that will be implemented in software, and will run on commodity hardware, virtualization and cloud computing technologies located in high-performance data centers, namely *Network Function Virtualization Infrastructures* (NFVIs).

This scenario imposes on NFVIs stringent performance and reliability requirements inherited from telecom applications, that are even more demanding than existing IT cloud systems: telecom workloads will require extremely low packet processing overheads, controlled latency, and efficient virtual switching, along with automatic recovery from faults and extremely high availability (99.99% or higher).

It can be easily seen that the "softwarization" of network functions raises performance and reliability concerns. NFVIs should be able to achieve resiliency in spite of *faults* occurring within them, such as hardware, software and configuration faults. The incidence of these faults is expected to be high, due to the large scale and complexity of data centers hosting the NFVI, and due to the massive adoption of several off-the-shelf hardware and software components: while these components are easily procured and replaceable, NFVIs will need to recover from faulty components in a timely way while preserving high network performance.

In this paper, we propose an experimental methodology for the dependability evaluation and benchmarking of NFVIs, based on *fault injection*. Characterizing and certifying the reliability of cloud computing systems, including NFV, is a high-priority issue for telecom operators, service providers and the user community, as demonstrated by recent initiatives encouraging the development of proof-of-concepts, best practices, test suites and benchmarks to assure cloud resiliency [4]. The proposed methodology includes both measures for characterizing performance and dependability, and the procedure and conditions under which these measures can be obtained. It is aimed to build confidence in the reliability of NFVIs, to highlight its weak points, and to provide practical guidance for designers. We apply the methodology in the context of a virtualized IP Multimedia Subsystem (IMS), deployed using commodity hardware and VMware virtualization technologies. In this case study, we evaluate the impact of faults on performance and reliability, analyze the sensitivity to different faulty components and fault types, and point out the pitfalls that can be incurred in the design of a reliable NFVI.

The paper is organized as follows. In section II, we provide background on dependability issues and prospective solutions for NFVIs. In section III, we describe in detail the proposed methodology for dependability evaluation and benchmarking. Section IV presents the IMS case study. Section V discusses related work, and section VI closes the paper.

II. Background on NFVI reliability

The NFV ISG identified use cases, requirements and architectures that will serve as a reference for the emerging NFV technologies. In particular, strict reliability requirements are demanded by customers and government regulations in the telecom domain [5], [6]. It is expected that virtualized network function will be able to assure comparable, or even superior reliability than traditional networks.

The prospective NFVI requirements and architecture, currently being defined by the ETSI [6], and presented in this section, includes fault tolerance mechanisms that will be adopted in the emerging NFVIs. According to these design principles, NFVI fault tolerance mechanisms will include *fault detection*, *fault localization*, and *fault recovery* (Fig. 1).

Fault Detection mechanisms of the NFVI are aimed at noticing the failure of a component (such as a VM or a node) as soon as the failure occurs, in order to timely start the fault treatment process. Fault detection mechanisms will be running in NFVI components (including hardware, hypervisors, guest OSes, and VNFs), and will interact with the *NFV Management and Orchestration* (NFV-MANO) of the NFVI during the fault treatment process. Fault detection involves (Fig. 2):

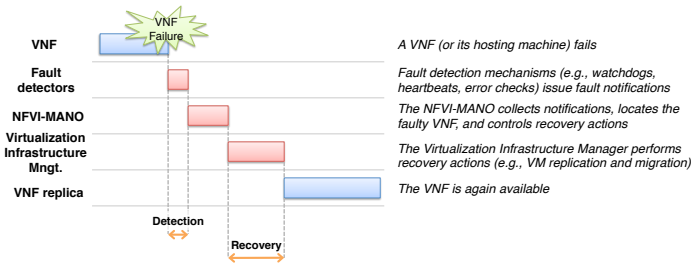


Fig. 1. Overview of fault tolerance in NFVIs.

- *Redundant checks over data and control flow* within software components. For instance, a hypervisor can check the status of CPU, memory, and I/O devices (e.g., I/O errors, parity errors, temperature warnings), and notify any error status that it detects. In the same way, VNFs can also perform end-to-end checks over protocol data, and collect and forward fault notifications produced by the underlying layers (e.g., the hypervisor or the guest OS).
- *Watchdog* components within the hypervisor and within VMs, that will check the "liveness" of components running on virtual and physical CPUs. The watchdog embeds a timer (either physical or virtual) that is periodically reset by a software routine; if a fault affects a virtual or physical CPU (thus, stopping its execution), then the timer will eventually trigger a fault notification and a recovery routine.
- *Heartbeat* components that check the health of other components, by periodically polling their status. A failure is detected if, for instance, a monitored component does not respond to status requests from the heartbeat component.
- *Performance monitors*, which analyze performance metrics such as resource consumption and system throughput and latency. For instance, a fault is detected if the throughput falls below a threshold.

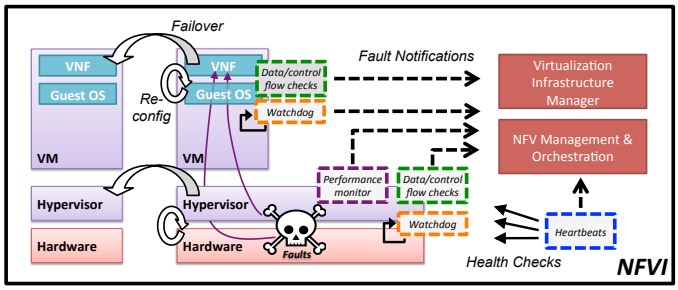


Fig. 2. Fault tolerance mechanisms in NFVIs.

Fault Localization mechanisms identify which components, among all components in the NFVI, have failed. It is important to note that even a single fault within the NFVI can cause cascading failures in one or more components, causing several fault notifications across the NFVIs (for example, a low memory condition in a physical machine could lead to several faults at the application level). Therefore, fault localization has to go back to the root cause of failures, in order to avoid unnecessary and/or incorrect recovery actions that would slow down or hamper recovery.

To locate faults, *fault correlators* are deployed across the NFVI (either locally on the hosts, or remotely in the NFVI-MANO) to collect and analyze failure information from NFVI components. Fault correlators will adopt *correlation rules* and *fault precedence graphs* defined by system administrators. By taking advantage of information collected from the NFVI (such as, guest OS logs or hypervisor logs collected at the time of a crash), fault correlators will be able to identify which kind of failure occurred. In turn, fault localization is followed by the selection and activation of a recovery action appropriate for the faulty component.

Fault Recovery mechanisms of the NFVI will perform a recovery action to remediate to the faulty component. Recovery actions for NFVIs (see Fig. 2) include the activation of VNFs replicas and of their VMs, and their migration to different hosts, by leveraging a *Virtualization Infrastructure Manager* to implement these actions. Moreover, VNFs and physical hosts can be reconfigured to mask a fault (for instance, by updating a virtual network configuration, by deactivating a faulty network interface card, or by retrying a failed operation). The recovery action can succeed or not, depending on the ability of the VNF and of the hypervisor to maintain a consistent state after the recovery action (i.e., the VNF is able to work correctly after recovery). A fault is successfully recovered if the time-to-recovery is below a maximum allowed time, which depends on the type and criticality of a VNF, ranging from few seconds (e.g., 5 seconds) in the most critical scenarios, to tenths of seconds (e.g., 25 seconds) in the less critical scenarios. Moreover, it is required that VNF performance after recovery should be comparable to the performance of VNFs before the occurrence of a fault.

Given the complexity of this fault management process, it becomes important to get confidence that NFVIs can achieve its strict performance and reliability requirements, which is the goal of the experimental approach proposed in this work.

III. Methodology

We propose an experimental methodology for evaluating and benchmarking performance and reliability of NFVIs *in the presence of faults*. The proposed methodology is based on *fault injection*, that is, the deliberate introduction of faults in a system during its execution [7]. The methodology includes three parts, that are summarized in Fig. 3.

The first part consists in the definition of *key performance indicators* (KPIs), the *faultload* (i.e., a set of faults to inject in the NFVI) and the *workload* (i.e., inputs to submit to the NFVI) that will support the experimental evaluation of an NFVI. Based on these elements, the second part of the methodology consists in the execution of a sequence of fault injection experiments. In each fault injection experiment, the NFVI under evaluation is first configured, by deploying a set of VNFs to exercise the NFVI; then, the workload is submitted to the VNFs running on the NFVI and, during their execution, faults are injected; at the end of the execution, performance and failure data are collected from the target NFVI; then, the experimental testbed is cleaned-up (e.g., by un-deploying VNFs) before starting the next experiment. This process is

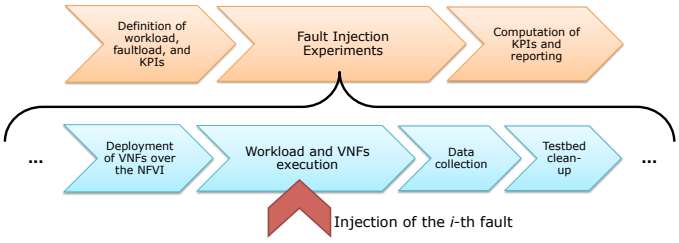


Fig. 3. Overview of dependability evaluation methodology.

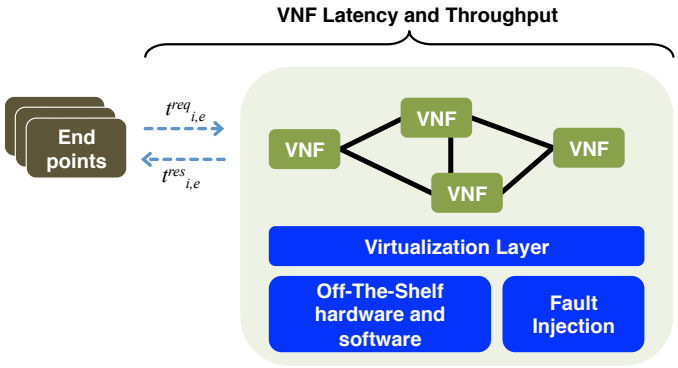


Fig. 4. VNF Latency and Throughput.

repeated several times, by injecting a different fault at each fault injection experiment (while using the same workload and collecting the same performance and failure metrics). The execution of fault injection experiments can be supported by automated tools for configuring virtualization infrastructures, for generating network workloads, and for injecting faults. Finally, performance and failure data from all experiments are processed to compute KPIs, and to support the identification of performance/dependability bottlenecks in the target NFVI.

In the following, we first present KPIs for performance and dependability of NFVIs, and then discuss the faultload and workload of fault injection experiments in NFVIs.

A. Key Performance Indicators

To evaluate performance and dependability of an NFVI, we consider the quality of service as perceived by its users. First, we define metrics for evaluating performance of an NFVI, which will be based on the responsiveness of VNFs running on the NFVI (referred to as *VNF latency* and *VNF throughput*). It is important to note that, while latency and throughput are widely adopted for characterizing performance of several types of systems, we specifically consider latency and throughput *in the presence of faults*. In fact, it can be expected that performance will degrade in the presence of faults, in terms of higher latency and/or lower throughput, since less resources will be available (due to the failure of components in the NFVI) and since the fault treatment process requires time (at least few seconds in the case of automated recovery, and up to several hours in the case of manual recovery), as discussed in section II. Thus, we introduce latency and throughput KPIs for NFVIs to quantify the *impact* of faults on performance, and evaluate whether the impact is too strong to be neglected.

Later in this section, we also discuss additional metrics related to the availability of NFVIs from the perspective of end-users. In fact, another likely impact of faults is the unavailability of VNFs, leading to the loss or the rejection of network traffic, in terms of incoming packets or requests that will not processed by VNFs. To analyze these effects in fault injection experiments, we include the *experimental availability* among the KPIs. Finally, at the end of this subsection we define the *risk score* of an NFVI, which provides a concise evaluation of NFVIs based on performance and dependability KPIs previously mentioned.

1) VNF Latency and Throughput: In general terms, network latency is the delay that a message "takes to travel from one end of a network to another" [8]. A similar notion can also be applied to network traffic passing through a VNF, or, more generally, through a network of interconnected VNFs (represented by a *VNF graph* [9], see Fig. 4). The *VNF latency* is the time required by a network of VNFs to process incoming traffic, which can be evaluated by measuring the time between a unit of traffic (such as a packet or a service request) enters the network of VNFs, and the time at which the processing of that unit of traffic is completed (e.g., a packet is routed to a destination after inspection, and leaves the VNFs; or, a response is provided to the source of a request).

Latency will be characterized by the empirical cumulative distribution function (CDF) of traffic processing times, and by considering the percentiles from this distribution. We denote the CDF by $F_{l_e}(x) = P(l_e < x)$, where l_e is the latency of a traffic unit in the fault injection experiment e. In turn, $l_e = t_e^{\mathtt{res}} - t_e^{\mathtt{req}}$, where $t_e^{\mathtt{req}}$ and $t_e^{\mathtt{res}}$ refer to the time of a request and of its response, respectively.

Fig. 5 shows an example of latency distribution. In particular, we consider the 50th and the 90th percentiles of the CDF (i.e., $F_{l_e}(50)$ and $F_{l_e}(90)$), which are adopted to characterize the average and the worst-case performance of telecommunication systems [10]. In the example, three scenarios are shown, with three cumulative distributions: (i) latency in fault-free conditions, (ii) latency in faulty conditions, in which the network is still providing good performance, and (iii) latency in faulty conditions, in which performance is severely degraded. The 50th and the 90th percentiles are compared to *reference values* for these percentiles, which specify the maximum allowed value of the percentile for an acceptable quality of service (for instance, reference values can be imposed by service level agreements). In Fig. 5, the maximum allowed values are $150ms$ for the 50th percentile, and $250ms$ for the 90th percentile. Both values are exceeded in the faulty scenario with performance degradation; in such a case, the NFVI is not able to properly mask faults to its users.

In a similar way, the VNF throughput considers the rate at which traffic units are successfully processed, e.g., processed packets or requests per second, in the presence of faults. VNF throughput represents the average throughput of VNFs along an experiment: it can be computed by dividing the total number N of traffic units (i.e., all traffic processed during an experiment e) by the total time that the system spent to process all the requests, that is, $N/(\max_i(t_{e,i}^{\mathtt{res}}) - \min_i(t_{e,i}^{\mathtt{req}}))$. Again, the VNF throughput is evaluated in the presence of injected faults. The measured VNF throughput can be compared to a reference

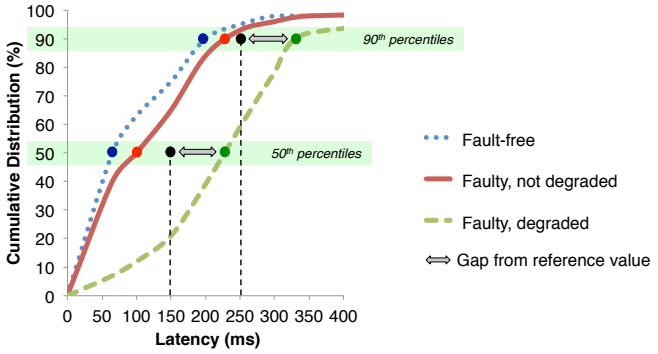


Fig. 5. Examples of VNF Latency distributions

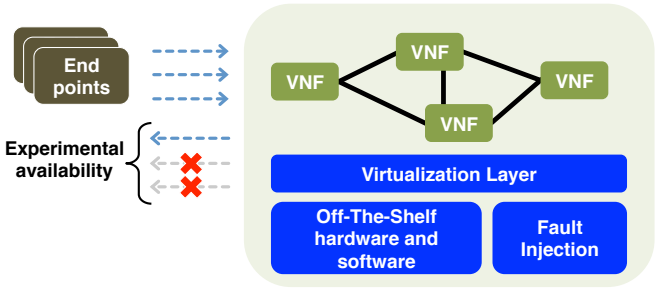


Fig. 6. Experimental availability

value, such as the VNF throughput in fault-free conditions, or the expected VNF throughput imposed by service level agreements.

To compute VNF latency and throughput, the end-points (Fig. 4) should record, for each unit of traffic i, its $t_{e,i}^{\texttt{req}}$ and $t_{e,i}^{\texttt{res}}$. The role of end-points is taken on by a *workload generator*, that is, a tool acting as user of the VNFs by submitting traffic to them, listening for replies, and computing performance measures based on these data. This aspect is further discussed later in this section.

2) Experimental Availability: Availability is a key aspect of quality of service. According to the TL 9000 definition for telecommunication systems [10], [11], availability is "the ability of a unit to be in a state ready to perform a required function at a given instant in time". The NFVI and its VNFs can become unavailable because of faulty components, causing service disruptions such as user-perceived outages, data losses and corruptions. The impact of faults on the NFVI can be mitigated through fault tolerance mechanisms and algorithms. Our dependability evaluation methodology deliberately injects faults into the NFVI, in order to evaluate whether the NFVI is able to maintain or to quickly restore availability.

It must be noted that, in general, availability cannot be predicted in probabilistic terms by the sole application of fault injection. Fault injection specifically focuses on evaluating the reaction of a system *given that a fault already occurred*. The availability also depends on the probability of occurrence of faults, which relies on other factors beyond the possibilities of fault injection and of our evaluation methodology, such as the reliability of individual components. For this reason, we evaluate the *experimental availability*, that is, the ability of an NFVI to be available when a fault is present in the NFVI. The availability of NFVI can be predicted by other means through the combination of the experimental availability along with other parameters, such as the probability of faults [7].

Experimental availability is defined as the percentage of traffic units that are successfully processed during a fault injection experiment (see Fig. 6), such as the percentage of packets or requests neither lost nor corrupted. It is obtained by dividing the number of successful requests $r_e^{\texttt{success}}$ (e.g., requests followed by a correct reply) over the total number of requests r_e during an experiment e, that is, $|r_e^{\texttt{success}}|/|r_e|$. To compute experimental availability, end-points need to track VNF request failures, which are typically denoted by error notifications sent to users, and by the lack of responses within a maximum allowed time (i.e., a *timeout*).

3) Risk Score: The *Risk Score* (RS) provides a brief and concise measure of the impact of faults within the NFVI, such as the risk of experiencing service unavailability and performance failures. We take into account several factors in the evaluation of risk, including: (i) the type of service and its criticality (in terms of number of users and importance of the service for the users), (ii) the impact of faults on the service as perceived by the end-users (e.g., faults can turn, in the best case, into negligible performance degradation or, in the worst case, into service unavailability), and (iii) the relative frequency of occurrence of faults. The Risk Score summarizes these factors, to provide an indication of risk for system designers, and to guide further analysis and improvements. In particular, the higher is the RS, the higher is the risk of service failures and, consequently, the worse is the capability of the underlying NFVI infrastructure to tolerate faults and assure service availability.

The Risk Score is a weighted sum of the number of service failures in fault injection experiments. It is defined as:

$$RS = \sum_{i=1}^{N} P_i \sum_{j=1}^{M} C_j \frac{F_{i,j}}{E_i}$$

where N different types of faults are injected, and M different types of service failures can be observed. For NFVIs, it is important to consider both *performance degradation* (section III-A1) and *service unavailability* (section III-A2) failures. Therefore, in the following, we will assume $M = 2$, and:

- $F_{i,1}$ = number of *performance degradation* failures ($j = 1$) under fault type i;
- $F_{i,2}$ = number of *service unavailability* failures ($j = 2$) under fault type i.

Moreover, E_i represents the number of fault injection experiments in which the fault type i has been injected. For instance, consider a hypothetical case where $N = 2$, and:

- $E_1 = 10$ (i.e., 10 experiments are performed using fault type $i = 1$), where $F_{1,1} = 2$ experiments experienced a performance degradation, $F_{1,2} = 3$ experiments experienced a service unavailability failure, and 5 experiments did not experience any failure;
- $E_2 = 10$ (i.e., 10 experiments are performed using fault type $i = 2$), where $F_{2,1} = 3$ experiments

experienced a performance degradation, $F_{2,2} = 4$ experiments experienced a service unavailability failure, and 3 experiments did not experience any failure.

In the weighted sum, $0 \leq C_j \leq 1$ is a weight that represents the *severity* of the failure type j, which depends on the impact of the failure in terms of business loss, number of affected users, and cost of recovery. We assume:

- $C_1 = 0.2$ for *performance degradation* failures;
- $C_2 = 1$ for *service unavailability* failures.

P_i is the relative importance of the fault type i, with $0 \leq P_j \leq 1$ and $\sum_j P_j = 1$. When all faults have a low probability of occurrence, and when no apriori information is available about their relative frequency, then their weights can be set to the same value (i.e., $P_i = 1/N$ for each i). These weights can be tuned using failure data for the NFVI if available (i.e., data obtained by analyzing failures occurring in production), for instance, by assigning a higher weight to the most frequent fault types.

In this hypothetical example, we have:

$$\begin{aligned} RS &= 0.5 \cdot \left(0.2 \cdot \frac{2}{10} + 1 \cdot \frac{3}{10}\right) + 0.5 \cdot \left(0.2 \cdot \frac{3}{10} + 1 \cdot \frac{4}{10}\right) \\ &= 0.5 \cdot 0.34 + 0.5 \cdot 0.46 = 0.4 = 40\% \end{aligned}$$

that is, there is a 40% risk of experiencing a service failure (either a performance degradation or unavailability) in the presence of a fault. In such a case, the exposure of VNFs to NFVI failures could not be neglected!

To compute $F_{i,j}$, we need to count the number of failures $F_{i,j}$ for each fault type i and for each failure type j. Failures should be identified by end-points that generate service requests and collect responses during the experiments:

- *Performance Degradation* failure: both the 50th and the 90th percentiles of VNF latency are respectively more than two thresholds T_{50} and T_{90}. In such a case, faults have a significant impact both on the average and worst case duration of network processing.
- *Service Unavailability* failure: the experimental availability is lower than a threshold T_{out}, that is, faults affect an unacceptably high number of requests.

In this case, T_{50} and T_{90} are latency thresholds (with $T_{50} < T_{90}$), and T_{out} is a percentage threshold on requests. Alternatively, VNF throughput can be considered instead of, or along with, VNF latency. The thresholds depend on the type of service, service level agreements, and users' expectations.

Finally, given the envisioned scenarios for NFVI [6], we must consider the case in which the NFVI hosts more than one service at a time. For instance, the NFVI could be used for the deployment of three services, such as video call, voice call, and emergency communication services. In this case, the Risk Score for the NFVI can be obtained by first computing the Risk Score for each individual service, and then by aggregating the three Risk Scores of the services with the formula:

$$RS_{NFVI} = \sum_{s=1}^{S} W_s RS_s \Big/ \sum_{s=1}^{S} W_s$$

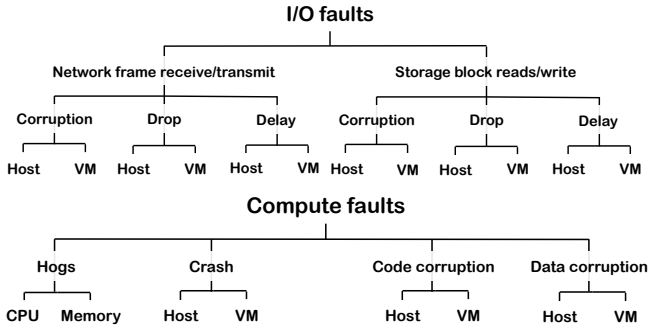


Fig. 7. Faultload for the dependability evaluation of NFVIs.

where S is the number of services (for instance, $S = 3$ represents three services), and W_s represents the relative importance of service s. For instance, if emergency communication ($s = 1$) is ten times more important of voice and video calls ($s = 2$ and $s = 3$), we may have $W_1 = 10$, and $W_2 = W_3 = 1$.

B. Fault Model

As discussed later in section V, fault injection in distributed systems encompasses two main fault categories: faults affecting I/O components (e.g., virtual network and storage), and faults affecting computational components (e.g., virtual CPUs and virtual memory). Faults in virtualized infrastructures (including hardware faults in OTS equipment, and software and configuration faults in the virtualization layer) mostly manifest as disruptions in I/O traffic (e.g., the transient loss or corruption of network packets, or the permanent unavailability of a network interface) and erratic behavior of the CPU and memory subsystems (in particular, corruption of instructions and data in memory and registers, crashes of VMs and physical nodes, and resource leaks).

These types of faults can be injected by emulating their effects on the virtualization layer. In particular, I/O and Compute faults (Fig. 7) can be emulated, respectively, by deliberately injecting I/O losses, corruptions and delays, and by injecting code and data corruptions, by forcing the termination of VMs and of their hosting nodes, and by introducing CPU and memory "hogs" (i.e., tasks that deliberately consume CPU cycles and allocate memory areas in order to cause resource exhaustion). Faults can be injected either in a specific VM (e.g., traffic from/to a VM), or in an NFVI node (affecting the hypervisor and all VMs deployed on the node).

These types of faults can be injected in a transient, intermittent, and permanent way to emulate different scenarios. The injection of *transient* faults (e.g., affecting an individual I/O transfer) can emulate temporary faults, such as failed reads/writes due to bad disk sectors or electromagnetic interferences. The injection of *intermittent* (i.e., periodical) faults can emulate temporary, but recurrent, faults, such as I/O errors due to worn-out connectors and/or partially damaged hardware interfaces. The injection of *permanent* faults can emulate faults that persist for a long period of time, such as unavailable hardware interfaces. We have implemented these faults in a prototype fault injection tool for virtualization infrastructures (currently supporting VMware ESXi and Linux containers),

by using loadable kernel modules to inject losses, delays, corruptions, and leaks.

C. Workload

During fault injection tests, the NFVI has to be exercised using a workload. In order to obtain reasonable and realistic results from fault injection, these workloads should reflect the workloads that VNFs will face in production: in this way, the experiments will provide a realistic picture of performance and dependability of the NFVI. Realistic workloads are typically generated using load generators and performance benchmarking tools. Our dependability benchmarking methodology is not tied to a specific choice of workload. Moreover, the selection of a workload mostly depends on the kind of VNFs that are hosted on the NFVI. For these reason, we refer the reader to existing network performance benchmarks and network load generators. Suitable examples of workloads for NFVIs are represented by performance benchmarks specifically designed for cloud computing systems [12], [13], [14], and by network load testing tools such as *Netperf* [15].

IV. Case Study

To show the application of the dependability evaluation methodology, we perform an experimental analysis of a virtualized IP Multimedia Subsystem (IMS) deployed over an NFVI. The goal of this analysis is to provide examples of results that can be obtained from fault injection. We consider a commercial virtualization platform (the VMware ESXi hypervisor) running real-world, open-source NFV software. In these experiments, we adopt fault injection to analyze:

- whether degradations/outages are **more frequent or more severe** than reasonable limits;
- the impact of different types of faults, to identify **the faults to which the NFVI is most vulnerable**;
- the impact of different faulty component, to find **the components to which the NFVI is most sensitive**.

A. NFVI Testbed

The experimental setup consists in an NFVI, whose fault tolerance is going to be evaluated, along with the VNFs that will be deployed on the NFVI. The NFVI testbed is depicted in Figure 8, and consists of:

- *Host 1* (Fault Injection Target): a workstation equipped with an Intel Xeon 4-core 3.70GHz CPU, 8 GB of RAM, and the VMware ESXi hypervisor. It hosts VMs running the VNFs of our case study (see section IV-B). It is instrumented with the fault injection tool.
- *Host 2*: a workstation with the same hardware and hypervisor of Host 1, and hosting VM replicas of the VNFs.
- *Tester Host*: a Linux-based computer that hosts a workload generator, and tools for managing the experiments by orchestrating the deployment of VNFs, by controlling the fault injection tool, and by collecting performance and failure data from the workload generator and from the NFVI.

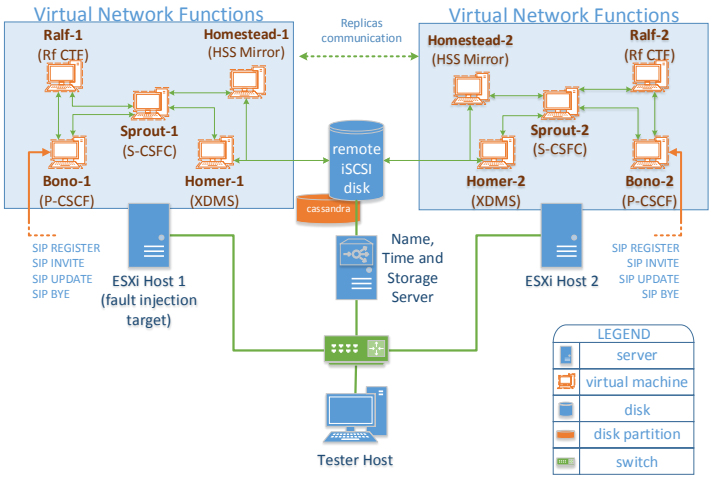


Fig. 8. The NFVI testbed, running an IP Multimedia Subsystem.

- *Name, Time and Storage Server*: a workstation hosting services (DNS, NTP, iSCSI) to support the execution of VNFs.
- A Gigabit Ethernet LAN connecting all the machines.

B. Virtual Network Functions

The VNFs running on the NFVI under evaluation are from the *Clearwater* project [16], [17], which is an open-source implementation of an IMS for cloud computing platforms. Figure 8 shows the components of the Clearwater IMS that are deployed on the NFVI testbed. They are:

- *Bono*: the SIP edge proxy, which provides both SIP IMS Gm and WebRTC interfaces to clients.
- *Sprout*: the SIP registrar and authoritative routing proxy, and handles client authentication.
- *Homestead*: component for retrieving authentication credentials and user profile information.
- *Homer*: XML Document Management Server that stores MMTEL service settings for each user.
- *Ralf*: component that provides billing services.

The workload consists of the set-up of several SIP sessions (calls) between end-users of the IMS. A SIP session includes requests for registering, inviting other users, updating the session and terminating the session. This workload is generated by *SIPp* [18], an open-source tool for load testing of SIP systems. A single experiment exercises the IMS by simulating 200 users and 100 calls.

We will consider a high-availability set-up, in which each VNF is actively replicated across the hosts (Fig. 8). The VNFs in Clearwater are designed to be stateless and to be horizontally scalable, in order to load-balance the SIP messages between the replicas using round-robin DNS. Later on, we extend the testbed with additional fault-tolerance capability provided by VMware vSphere (namely, HA cluster [19]), which automatically migrates and/or restarts VMs to recover from overload and crash failures.

C. Fault Injection Test Plan

Faults will be injected in the Host 1, and on VNF replicas running on that node. As discussed in section III-B, we consider both *I/O* and *compute* faults, and both *intermittent* and *permanent* faults. Network frame corruptions, drops, and delays will be injected in the host, and on the *Sprout* VNF. The only VNFs that use remote storage (iSCSI) are Homer and Homestead; to emulate storage faults, we will inject network faults in the iSCSI traffic generated by Homestead. Moreover, experiments will include compute faults such as CPU/memory hogs, host/VM crashes, and code/data corruptions at the VM and at the host level. Three repeated experiments will be performed for each type of fault, for a total of 93 fault injection experiments.

D. Experimental analysis

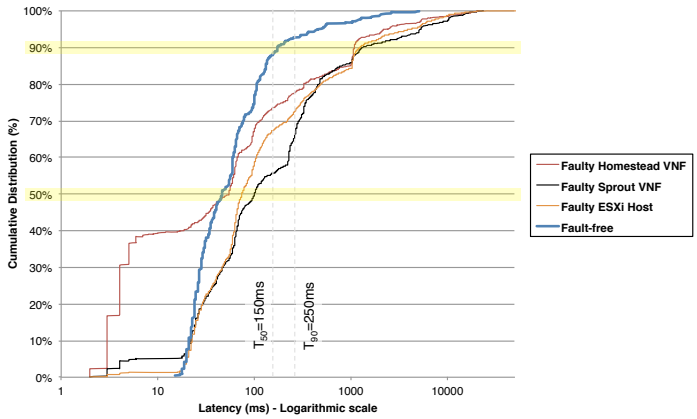
Using performance and failure data from the experiments (in particular, the logs of the workload generator), we first analyze the experimental availability in the presence of faults, which is obtained from the percentage of SIP requests successfully processed by the IMS. Table I provides the experimental availability for different subsets of experiments, and the average for all fault injection experiments.

TABLE I. EXPERIMENTAL AVAILABILITY, FOR DIFFERENT GROUPS OF FAULT INJECTION EXPERIMENTS.

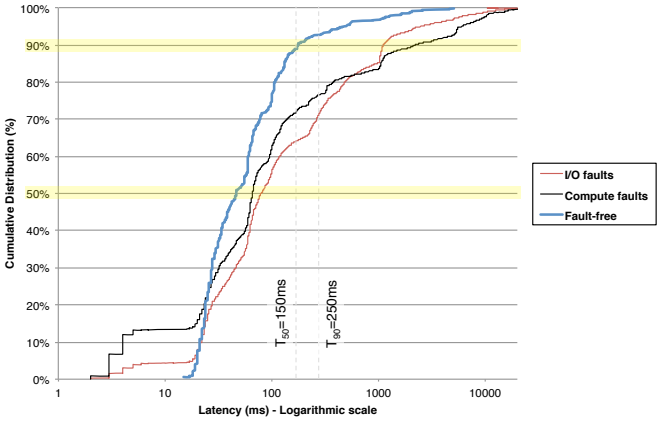
Fault Type \ Fault Target	Sprout	Homestead	ESXi host	Average
Compute faults	8.01%	39.67%	59.19%	35.62%
I/O faults	48.29%	82.40%	70.67%	67.12%
Average	28.15%	61.03%	64.93%	51.37%

The table shows that the average request success rate is 51.37% in the presence of faults. By looking more in detail at the fault types (i.e., by dividing the data between I/O faults and Compute faults, and separately analyzing the two sets), we observe that Compute faults have a stronger impact on availability (35.62%, lower than the average) than I/O faults (67.12%, higher than the average). This points out that, while it is important to have redundant and reliable devices to prevent I/O faults, it is even more important to introduce additional resources to mitigate CPU and memory faults, including more VM instances and physical CPUs to compensate for faulty ones, and to perform real-time monitoring of CPU and memory usage for the timely detection of faults.

We also analyze how the experimental availability of the NFVI is influenced by the location of faults. To this aim, we separately analyzed availability by dividing the data between faults injected in *Sprout*, faults injected in *Homestead*, and faults injected in the physical host. We found that the injected VNF has even more impact than the type of faults. In the case of *Sprout* faults, the success rate significantly decreases (28.15%, much lower than the average); the success rate is influenced to a lower degree by faults in *Homestead* and in the host (61.03% and 64.93%, higher than the average). This can be explained by the pivotal role of *Sprout* in the architecture of Clearwater, since this component acts as registrar, router, and handles client authentication. It is advisable to introduce special fault tolerance mechanisms and policies for this VM, for instance by providing more replicas, by transparently balancing the load among replicas, and automatic recovery (such as VM restarts).



(a) By targeted NFVI component



(b) By type of injected faults

Fig. 9. Cumulative distribution of latency.

We then analyze the SIP request latency in the presence of faults. In fact, SIP request failures were not the only effect of faults. We also found that, even if some requests succeed, they can take much more time than normal to complete. Fig. 9 shows this behavior, in which we report the cumulative distribution of service latency, by dividing the data respectively by target component and by type of injected faults. In both cases, we observe that the latency increases only by a moderate amount in the average case, which is represented by the 50th percentile of the CDFs, and remains below 150ms. Instead, the latency significantly increases in the worst case, which is represented by the 90th percentile of the CDF: The latency increases by an order of magnitude, as 10% of the SIP requests take several seconds to be processed. This is a results of the reduced amount of resources that is caused by the injection of faults. This result means that, on the one hand, that faults in the NFVIs can also turn into performance degradation, and that this kind of behavior needs to be studied and prevented; on the other hand, this result suggests that performance monitoring (e.g., using internal and/or external heartbeat mechanisms, and analyzing performance logs) at the service level is critical to assure a high coverage of fault detection, localization, and recovery.

From data on experimental availability and latency, we identified performance degradation and service unavailability failures, and computed the risk score for the NFVI, which is shown in the TABLE II. The overall risk score (55%) is quite high and reflects the strong impact that faults have on experimental availability (TABLE I): as before, there is a high risk of service outages, especially in the case of Compute faults. This result points out that the NFVI under evaluation is not sufficiently fault-tolerant, and that fault tolerance mechanisms need to be carefully improved to lower the risk of failures. We remark that this result confirms the strong need for fault injection when dealing with complex architectures such as NFVIs. In fact, the effectiveness of fault tolerance mechanisms is very dependent on the actual configuration chosen by NFVI designers and administrators. Unfortunately, modern virtualization platforms are quite complex technologies, requiring many design choices concerning the placement of VMs across physical nodes, the topology of virtual networks and storage, the allocation of virtual CPU and memory for VMs, and so on. The problem is exacerbated by the issues behind development and testing of fault-tolerant distributed applications, such as the IMS that we have considered. In our specific case, after a detailed analysis of experiment logs, we found that the low experimental availability was due to a capacity planning issue: once a VNF on Host 1 fails (because of fault injection), the SIP traffic is forwarded to a replica of the VNF on Host 2, but the capacity of the VNF was not enough to handle all SIP traffic, causing the failure of many SIP requests.

TABLE II. RISK SCORE.

Fault Type \ Fault Target	Sprout	Homestead	ESXi host	All targets
Compute faults	100%	100%	47%	67%
I/O faults	68%	58%	37%	48%
All faults	79%	69%	38%	**55%**

The problem of designing a reliable NFVI is even more evident if we consider what happens to the IMS after introducing additional fault tolerance mechanisms. We enabled the VMware HA cluster capability, in order to mask the failure of VNFs by automatically migrating and restarting VMs after a crash or an overload. We then performed the fault injection experiments a second time, but we did not obtained a significant improvement of the experimental availability and of the risk score. In fact, the automatic migration and restart of VMs proved to be too slow and does not achieve an acceptable availability. Fig. 10 shows this behavior, in which we depict the network activity of the Sprout VNF respectively in fault-free and in two faulty runs (with and without the HA cluster capability, respectively) over a period of 5 minutes (the duration of an experiment). A VM crash is injected at time 100: when HA cluster is disabled, the Sprout VNF is no more active after the crash; when HA cluster is enabled, the VNF is automatically restored at time 160. Unfortunately, 60 seconds are too much to guarantee a quick recovery and a high availability. This result suggests to pay more efforts towards improving the boot time of the VM, and to increase the capacity of the nodes to speed-up the recovery process. Again, we remark that a careful fault injection experimentation is required to guide designers towards a reliable and performant NFVI.

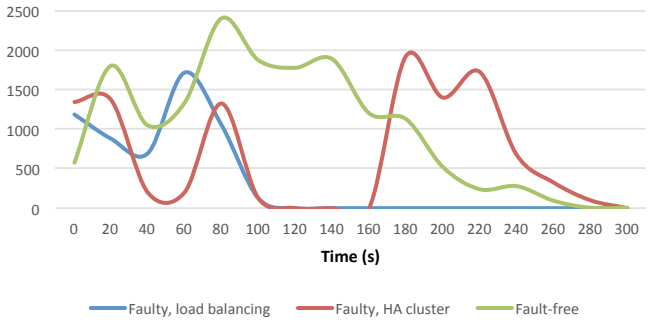


Fig. 10. Network throughput during the injection of a VNF crash.

V. RELATED WORK

Dependability benchmarking is a general framework for comparing the dependability of computer systems in the presence of faults [20]. A key aspect of dependability benchmarking, which makes it different from simple fault injection, is that it represents an agreement that is accepted both by the computer industry and by the user community: The benchmark specifies in detail the measures, the domain in which these measures are considered valid and meaningful, and the procedures and rules to be followed, to enable users to implement the benchmark for a given system and to interpret the results. Dependability benchmarks have been proposed for several types of systems, such as transaction processing systems, as described in [20]. The evaluation framework established by dependability benchmarks is today partially integrated in the *ISO/IEC Systems and software Quality Requirements and Evaluation* (SQuaRE) standard, which defines an evaluation module, the *ISO/IEC 25045*, that deals with the assessment of the *recoverability* of IT systems in the presence of accidental faults, and defines two measures, the *resiliency* (ratio between the throughput obtained in absence and presence of faults), and the *autonomic recovery index* (degree of automation in the system response against a threat) [21].

With the softwarization of network functions, it becomes important to extend the scope of dependability benchmarking to NFV. To this purpose, the general principles of dependability benchmarking need to be tailored for NFV, by identifying appropriate measures, faultloads and workloads. This work represents a step towards this goal, by proposing a set of KPIs for characterizing performance and availability, and an experimental approach for the quantitative evaluation of performance degradation and unavailability of NFVIs.

Several fault injection techniques and tools have been developed for the dependability evaluation of complex and distributed systems, including distributed filesystems [22], OLTP systems [23], [24], multicast and group membership protocols [7], [25], and real-time communication systems [26]. More recently, fault injection techniques and tools have been developed for cloud computing software. In [27], Ju *et al.* discuss the testing of fault resilience of the OpenStack cloud computing platform. They inject faults targeting communication among OpenStack services, namely service crashes (by killing service processes) and network partitions (by disabling communication between two subnets). Fault injection identified several types of bugs, such as erroneous return code checking from OpenStack

services, timeout bugs (e.g., indefinite waits for a failed service), and erroneous state transitions in the lifecycle of VMs. *Fate* [28], and its successor *PreFail* [29], are tools aimed at testing cloud software (including Cassandra, ZooKeeper, and HDFS) against *multiple faults* from the environment, including disk failures, network partitions, and crashes of remote processes. They inject faults by intercepting method calls (e.g., library calls for disk or network I/O), and raising exceptions instead of normally executing method calls. Multiple faults are injected during an experiment, in order to test exception handlers and recovery routines when faults keep occurring during their execution. CloudVal [30] is a framework to test the isolation among a hypervisor and its VMs (e.g., whether faults in a VM can propagate their effects outside the VM). The framework provides a set of tools (supporting KVM and Xen) that adopt debugger-based techniques to inject "soft" faults in memory and CPU registers, guest misbehavior, leaks and CPU losses. In summary, all these studies adopt fault injection for testing of specific cloud computing components.

We remark that the dependability evaluation of NFVIs should go beyond the testing of its individual components. In fact, the dependability of NFVIs results from the tight interactions among several components, where fault tolerance is introduced at several layers, as discussed in section II. Moreover, differing from traditional IT cloud infrastructures, NFVIs have more stringent performance and dependability requirements inherited from the telecom applications they are meant for. Therefore, our methodology jointly evaluates performance and dependability of the NFVI as a whole, following a holistic approach and leveraging on fault injection.

VI. Conclusion

Performance and reliability are critical objectives for the widespread adoption of NFVIs. In this paper, we presented a dependability evaluation and benchmarking methodology for NFVIs. Based on fault injection, the methodology analyzes how faults impact on VNFs in terms of performance degradation and service unavailability. The case study on the IMS showed how the methodology can point out dependability bottlenecks in the NFVI and guide design efforts. Future work will extend the evaluation to different NFVI architectures, by also considering alternative virtualization technologies.

Acknowledgment

This work has been partially supported by the project PON-FSE-MIUR DISPLAY (PON02_00485_3487784) and by Huawei Technologies Co. Ltd.

References

[1] NFV ISG, "Network Functions Virtualisation - An Introduction, Benefits, Enablers, Challenges & Call for Action," ETSI, Tech. Rep., 2012.

[2] ——, "Network Functions Virtualisation (NFV) - Network Operator Perspectives on Industry Progress," ETSI, Tech. Rep., 2013.

[3] A. Manzalini, R. Minerva, E. Kaempfer, F. Callegari, A. Campi, W. Cerroni, N. Crespi, E. Dekel, Y. Tock, W. Tavernier *et al.*, "Manifesto of edge ICT fabric," in *Proc. ICIN*, 2013, pp. 9–15.

[4] European Union Agency for Network and Information Security, "Cloud computing certification." [Online]. Available: https://resilience.enisa.europa.eu/cloud-computing-certification

[5] NFV ISG, "Network Functions Virtualisation (NFV) - Virtualisation Requirements," ETSI, Tech. Rep., 2013.

[6] ——, "Network Function Virtualisation (NFV) - Resiliency Requirements," ETSI, Tech. Rep., 2014.

[7] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. Fabre, J. Laprie, E. Martins, and D. Powell, "Fault injection for dependability validation: A methodology and some applications," *IEEE TSE*, vol. 16, no. 2, pp. 166–182, 1990.

[8] L. L. Peterson and B. S. Davie, *Computer Networks, Fifth Edition: A Systems Approach*, 5th ed. Morgan Kaufmann Publishers Inc., 2011.

[9] NFV ISG, "Network Functions Virtualisation (NFV) - Virtual Network Functions Architecture," ETSI, Tech. Rep., 2013.

[10] E. Bauer and R. Adams, *Reliability and Availability of Cloud Computing*, 1st ed. Wiley-IEEE Press, 2012.

[11] Quality Excellence for Suppliers of Telecommunications Forum (QuEST Forum), "TL 9000 Quality Management System Measurements Handbook 4.5," Tech. Rep., 2010.

[12] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. SoCC*, 2010, pp. 143–154.

[13] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the Weather Tomorrow?: Towards a Benchmark for the Cloud," in *Proc. DBTest*, 2009.

[14] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *Proc. CCA*, 2008.

[15] HP Networking Performance Team. Netperf HomePage. http://www.netperf.org/netperf/.

[16] Clearwater, "Project Clearwater - IMS in the Cloud," 2014. [Online]. Available: http://www.projectclearwater.org/

[17] G. Carella, M. Corici, P. Crosta, P. Comi, T. M. Bohnert, A. A. Corici, D. Vingarzan, and T. Magedanz, "Cloudified IP Multimedia Subsystem (IMS) for Network Function Virtualization (NFV)-based architectures," in *Proc. ISCC*, 2014.

[18] Gayraud, Richard and Jacques, Olivier and Day, Robert and Wright, Charles P. SIPp. http://sipp.sourceforge.net/.

[19] M. Brown, A. Kapur, and J. King, "VMware vCenter Server 5.5 Availability Guide," Tech. Rep., 2014.

[20] K. Kanoun and L. Spainhower, *Dependability Benchmarking for Computer Systems*. Wiley-IEEE Computer Society, 2008.

[21] J. Friginal, D. de Andrés, J.-C. Ruiz, and R. Moraes, "Using Dependability Benchmarks to Support ISO/IEC SQuaRE," in *Proc. PRDC*, 2011, pp. 28–37.

[22] R. Lefever, M. Cukier, and W. Sanders, "An experimental evaluation of correlated network partitions in the Coda distributed file system," in *Proc. SRDS*, 2003, pp. 273–282.

[23] M. Vieira and H. Madeira, "A dependability benchmark for OLTP application environments," in *Proc. VLDB*, 2003, pp. 742–753.

[24] A. Bondavalli, S. Chiaradonna, D. Cotroneo, and L. Romano, "Effective fault treatment for improving the dependability of COTS and legacy-based applications," *IEEE TDSC*, vol. 1, no. 4, pp. 223–237, 2004.

[25] B. Helvik, H. Meling, and A. Montresor, "An approach to experimentally obtain service dependability characteristics of the Jgroup/ARM system," *Proc. EDCC*, pp. 179–198, 2005.

[26] S. Dawson, F. Jahanian, T. Mitton, and T. Tung, "Testing of fault-tolerant and real-time distributed systems via protocol fault injection," in *Proc. FTCS*, 1996, pp. 404–414.

[27] 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.

[28] H. S. Gunawi, T. Do, P. Joshi, P. Alvaro, J. M. Hellerstein, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, K. Sen, and D. Borthakur, "FATE and DESTINI: A Framework for Cloud Recovery Testing," in *Proc. NSDI*, 2011, pp. 238–252.

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

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