Network Failures in Cloud Management Platforms: A Study on OpenStack

Hassan Mahmood Khan[a], Frederico Cerveira[b], Tiago Cruz[c] and Henrique Madeira[d]
University of Coimbra, CISUC, DEI, Portugal

Keywords: Cloud Computing, Network Failure, OpenStack, Cloud Management Platform, Fault Injection, Dependability.

Abstract: Cloud Management Platforms (CMPs) have a critical role in supporting private and public cloud computing as a tool to manage, provision and track resources and their usage. These platforms, like cloud computing, tend to be complex distributed systems spread across multiple nodes, thus network faults are a threat that can lead to failures to provide the expected service. This paper studies how network faults occurring in the links between the nodes of the CMP can propagate and affect the applications that are hosted on the virtual machines (VMs). We used fault injection to emulate various types of network faults in two links of the OpenStack CMP while a common cloud computing workload was being executed. The results show that not all network links have the same importance and that network faults can propagate and cause the performance of applications to degrade up to 50% and a small percentage of their operations to fail. Furthermore, in many campaigns some of the responses returned by the applications did not match the expected values.

1 INTRODUCTION

With the development of computer technology, the range of applications and their complexity has expanded, along with a fast increase in the usage of computing resources. To sustain this trend, there is a need for a large and diversified pool of computing resources that can satisfy the clients' requirements. Cloud computing offers shared, scalable, ubiquitous, reconfigurable, and simple on-demand access to computing resources (such as networks, storage, processing units, applications, and services) via configurable Internet services that can be quickly deployed and released with reduced management overhead. Due to some characteristics of cloud computing, such as scalability, agility and cost-effectiveness, many businesses tend to place their application services on the cloud.

With the ongoing increase of application requirements and considerable progress in cloud computing system research, many researchers are focusing on cloud management platforms and their dependability and fault tolerance. Cloud management platforms, such as OpenStack (OpenStack, 2022), are a collection of software modules and tools that provides a framework to create and manage both public cloud and private cloud.

Dependability is "the ability to deliver service that can justifiably be trusted" (Avižienis et al., 2004). Cloud computing system dependability is crucial for cloud service providers, brokers, carriers, and users worldwide. Fault injection is an important method that can accelerate the occurrence of faults in a controlled manner in a system. It aids in understanding how the system behaves when stressed in unusual ways, hence helping make it more fault tolerant.

To establish cloud computing as a trustable platform for cloud stakeholders, it must achieve levels of dependability that are comparable to the dependability offered by classical dedicated infrastructures. In other words, clients will shy away from migrating their applications to the cloud, particularly the business-critical ones, if by moving they are significantly worsening its dependability.

This research paper addresses the impact that moving an existing workload from a dedicated infrastructure to a CMP-based cloud infrastructure can have on the dependability of the applications. We focus on network faults that can occur in the network links between the nodes of the CMP and how a fault in these links can propagate up to the hosted applica-

[a] https://orcid.org/0000-0002-7974-2108
[b] https://orcid.org/0000-0002-0180-4815
[c] https://orcid.org/0000-0001-9278-6503
[d] https://orcid.org/0000-0001-8146-4664

Khan, H., Cerveira, F., Cruz, T. and Madeira, H.
Network Failures in Cloud Management Platforms: A Study on OpenStack.
DOI: 10.5220/0011851400003488
In *Proceedings of the 13th International Conference on Cloud Computing and Services Science (CLOSER 2023)*, pages 228-235
ISBN: 978-989-758-650-7; ISSN: 2184-5042

tion and affect the timeliness and correctness of the service provided. To emulate network faults, fault injection is used with the help of Sidekick, a network traffic shaper that supports various fault models, such as packet loss, bandwidth congestion or latency.

The results show that it is possible that hosted applications, running on the VMs that have been provisioned using the CMP, are affected by network faults in the links that connect the nodes of the CMP. The experienced failure modes include performance degradation (i.e., the service takes longer to fulfil the requests), failed operations (e.g., due to a timeout) and even operations that return invalid results. However not every network link has the same impact. From the two considered scenarios (i.e., network links where faults were injected) only one affected the service provided by the hosted applications.

The remaining parts of the paper are structured in the following manner: The background of cloud management platforms, their dependability, and network failure are covered in Section 2. The experimental configuration is discussed in Section 3. The findings collected from the experiment are discussed in Section 4. Threats to validity of this study are presented in Section 5 and conclusions drawn and future work are described in Section 6.

2 BACKGROUND

Cloud computing can be regarded as an ensemble of networked, virtualized computers that are constantly provided and presented as unified computing resources in accordance with service-level agreements between the service provider and the consumer make up a cloud, which is a distributed system (Kumari and Kaur, 2021). Cloud computing makes it possible to access, configure, and manipulate a shared pool of reconfigurable computing resources, including as networks, servers, storage, applications, and services, in a manner that is pervasive, convenient, and on-demand.

The systematic monitoring, control, administration, and maintenance of cloud computing infrastructure, services, and resources is referred to as cloud management. A cloud management platform, or CMP, is a collection of software used to manage cloud environments (Cocozza et al., 2015). The primary objective of the CMP is to make it possible to improve resource management and monitoring of cloud resources (Lu et al., 2020). Some examples of CMPs currently available include Abiquo, CloudStack, Eucalyptus, Nimbus, openQRM, OpenStack (OpenStack, 2022), Open Nebula, Apache Virtual Computing Lab (VCL), and HP's CloudSystem Matrix. OpenStack (OpenStack, 2022) is one of the most prominent CMP that uses pooled virtual resources to build and manage clouds.

Several methodologies have been utilized to assess the dependability of cloud computing systems, including analytic, state space, statistical, PetriNet, simulation methods (Dantas et al., 2012) and fault injection. Fault injection emulates faults in a target system to produce failures similar to real-world failures but at a faster pace (Natella et al., 2016).

Recent research has focused on dependability of cloud computing systems. Ju et al. (Ju et al., 2013) examined OpenStack's resilience by introducing failures (by terminating VMs or service processes), network partitions (by prohibiting connection between two subnets), and network traffic delay and packet losses (by disrupting REST service requests). (Cerveira et al., 2015) introduce CPU and memory bit flips to evaluate hypervisor and VM isolation when affected by transient hardware faults. (Pham et al., 2016) employed fault injection against OpenStack to inject and analyze the caused failures. A study by (Cotroneo et al., 2019) employed fault injection and failure analysis to explore the consequences of failures in the widely used OpenStack CMP. Their results show that software bugs can propagate inside the components of the CMP, thus suggesting the usage of more advanced testing and fault tolerance techniques (Cotroneo et al., 2022).

There are various benchmarks for evaluating a cloud platform from numerous perspectives. Among these, the Yahoo Cloud Serving Benchmark (YCSB) (Cooper et al., 2010) is a well-known key-value data storage benchmark, which supports the majority of key-value store databases. YCSB purpose is to provide a framework and a collection of common workloads for measuring the performance of various key-value and cloud serving stores

Apache Cassandra (Cassandra, 2022) is an open-source, distributed NoSQL database that uses wide-column partitioning. Facebook created Apache Cassandra by combining Google's Bigtable data and storage engine concept with Amazon's Dynamo distributed storage and replication techniques

3 EXPERIMENTAL SETUP

To evaluate the impact of network failures on the OpenStack CMP and the applications hosted on it, fault injection of network faults was performed. This section describes the experimental setup, including how Openstack was configured, the used workload, the type of network faults emulated, the fault injection process and the flow of the experiments.

3.1 Environment Configuration

We have opted to conduct our experimental evaluation using the most commonly used CMP, OpenStack (more specifically, we used OpenStack Xena). Figure 1 shows the experimental design comprising OpenStack, workload and benchmark applications on the hosted VMs and network traffic shaper. OpenStack was configured over three nodes, which are:

Controller node: It runs identity service, image service, web dashboard, networking agents and management portions associated to compute and networking. It also includes supporting services i.e. SQL database, message queue, Network Time Protocol.

Compute node: It runs the hypervisor that supports the instances (i.e. virtual machines). In our setup, the used hypervisor was KVM 4.2.1. It also runs a networking service agent that connects instances to virtual networks and provides firewalling services to instances via security groups.

Storage node: This node contains the disks that are provisioned for the instances.

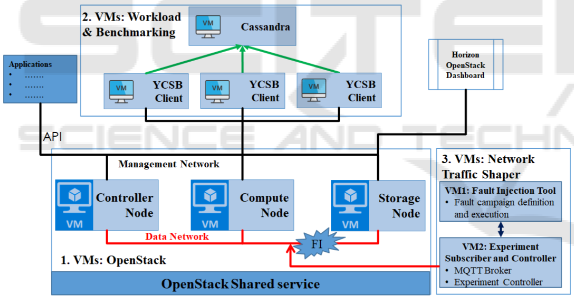

Figure 1: Experimental setup, CMP, Workload VMs, and network traffic shaper.

Our OpenStack network configuration consists of two networks. The first is the *management network*, which provides external access (to the Internet or to a private network) to all nodes for administrative purposes (e.g., package installation or security updates). The other network is the *data network*, which directly connects the Controller, Compute and Storage nodes. Due to the different roles, it is a good practice to keep these two networks separate of each other. In our experiments, we focus on the *data network*, because it is the network that supports the CMP, whereas the *management network* has a support role.

For the experimental evaluation, we derived two scenarios as to evaluate how faults in different network links may affect the system differently:

- C1: fault injection on the data network connection between compute node and storage node;
- C2: fault injection on the data network connection between the controller node and storage node.

3.2 Network Traffic Shaper

There was need to use a tool capable of emulating various kinds of network faults in the least intrusive and most reproducible manner possible to perform network faults injection. For that purpose, the network traffic shaper tool named Sidekick was used to inject communications-related disturbances in a controlled way. Although in this paper Sidekick is used to evaluate the effect of network faults in a system, it can also be used to evaluate the performance and robustness of communications protocols, APIs or services.

The fundamental operation model for this tool is presented in Figure 2.

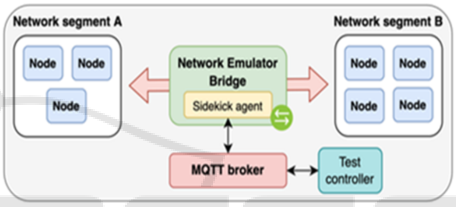

Figure 2: Sidekick operational model.

A Network Emulator bridge based on a Linux virtual appliance provides the means to transparently constrain/disturb the traffic between communicating peers on different network segments. This appliance is configured with three network interfaces: two for the transparent bridge and one for out-of-band experiment control. Network traffic shaping capabilities are provided by the native Linux TC and Netem subsystems to constrain bandwidth or inject disturbances such as packet losses, jitter, or latency.

To integrate these capabilities within a fault injection framework, the Sidekick agent provides remote control of the traffic shaper for integrated experiment management. This agent provides the capabilities of remote scheduling for time-triggered activation, configuration through an easy-to-use JSON profile, operation in foreground or background mode, and trigger precision around 0.02s, on average (for overall system load <20%).

Sidekick allows for experiments to be scheduled, by supplying an ID for a group of affected IP addresses (obtained from a configuration file), the starting moment, encoded in UNIX timestamp/epoch format (UTC) and the test duration (in seconds). For instance, to activate a scheduled test, an MQTT plaintext message has to be sent to the platform MQTT server, to the topic "<session_name>/scheduler", with the format "groupID:start:duration".

Experiments can be configured in two ways: Dynamic Configuration, by employing new configuration modes pushed via MQTT messages. For this purpose, the PUSH command allows to push new test profiles for a specific group over-the-wire. Static Configuration, by utilizing a JSON file that establishes the nominal and faulty conditions for groups of IP addresses. The Sidekick command set provides further resources to activate test modes, force clock synchronization and control logging and reporting, among other operations.

3.3 Workload

The YCSB (Cooper et al., 2010) key-value data storage benchmark is widely used and represents a common use case found in cloud computing (key-value stores). For our experiments we paired YCSB with the open source, distributed NoSQL database, Apache Cassandra (Cassandra, 2022). Cassandra was installed in one VM and three other VMs were used to run the YCSB clients that send operation requests to Cassandra. The four VMs used for the workload aim to replicate AWS t2.small (VCPUS 1, Memory 2GB, Storage 10 GB) (AWS, 2022). We used the *workloada* of the benchmark, that consists on 50% proportion of read-operations and 50% proportion of update-operations. We configured record count as 1000 and operation count as 5000, based on the performance and hardware resources of our setup. Finally, we activated the configuration of YCSB that checks the data integrity of the read operations, thus verifying the integrity of the results.

The usage of a workload at the level of the VM in opposition to a workload that exercises the management operations of OpenStack is justified because we desire to focus on studying the impact on the applications that are hosted in the VMs, and because even a workload running on the VMs will cause OpenStack to be exercised (e.g., there will be network traffic between the Compute and Storage node due to the disk reads and writes triggered by the workload).

3.4 Fault Model

In our experiments, three different types of network faults are emulated: packet loss, latency and network congestion. These three types of faults were chosen because previous work has shown them to be representative of network faults that occur in the wild (Qi et al., 2021) (Cotroneo et al., 2022).

For the fault injection experiment campaign, we have grouped the network faults according to their intensity into three levels (Low, Medium and High). Furthermore, we also vary the duration that the network fault remains active by three levels. In other words, we assume that a network fault will be fixed and normal operation will resume after some time. Table 1 shows the values used for fault intensity and fault duration in our fault injection campaigns.

Table 1: Network fault types and configuration.

Fault Injection Intensity	Fault type			
	Network Congestion	Packet Loss	Latency	FI Exec Duration
Low	250 Mbps	25%	0.5s	30s
Medium	100 Mbps	50%	1.0s	45s
High	0.5 Mbps	75%	3.0s	60s

The fault injection campaigns were designed to evaluate the impact of each network fault type across different durations. In each experiment, a certain fault type is picked and then its intensity and duration are varied. In total, we executed 27 experiment configurations (100 runs per each configuration) where fault type and duration varied. Over the 27 configurations, we amassed a total of 5400 experiment runs (27 configurations x 2 scenarios x 100 runs).

3.5 Experimentation Flow

Figure 3 presents the flow of an experiment execution (or experiment run). Before the initialization of the experiments, the hosted VMs (Cassandra and YCSB instances) are already provisioned and running. The flow of an experiment run comprises the initialization of the workload on the hosted VMs. For each experiment run, we load a fresh copy of the database state to ensure a clean experiment environment. The workload is always executed for at least 10 sec (warm-up time) before any fault is injected. After this warm-up period, Sidekick is executed with one specific configuration (i.e., fault type, intensity, and duration). During the fault injection experiment, the workload may become unstable or malfunction. After the fault injection duration, the "Keep Time" interval enables the workload to return back to a normal state.

Thus, the experiment flow can be summarized into the following steps:

1. Load Cassandra workload on the YCSB benchmark in OpenStack Experiment Environment.
2. Run Cassandra workload on the YCSB benchmark in OpenStack Experiment Environment: At Time T, launch the workload and wait to receive normal behaviour (warm-up 10 sec).

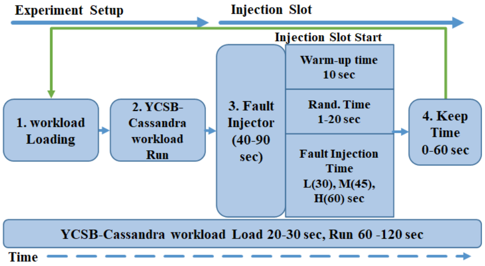
Experiment Setup
Injection Slot
Injection Slot Start
1. workload Loading
2. YCSB-Cassandra workload Run
3. Fault Injector (40-90 sec)
Warm-up time 10 sec
Rand. Time 1-20 sec
Fault Injection Time L(30), M(45), H(60) sec
4. Keep Time 0-60 sec
YCSB-Cassandra workload Load 20-30 sec, Run 60 -120 sec
Time

Figure 3: Flow of an experiment run.

3. Fault Injection: Fault injection begins (at T= T+10 (1-20 sec randomization)), inject specific faults type for 30 sec to 60 sec duration for each iteration.
4. Keep Time (Back to Normal after a failure, if any). As the fault injection stops, the system tended to Normal behaviour (at Ti= T+ 80-30 sec).

Figure 3 shows that there could be a constant warm-up time of 10 seconds and variations in the randomization time and fault injection time that will vary the keep time. i.e. In a Case, a maximum "keep time" of 80 seconds, assuming a 1-second minimum fault injection initiation randomization time and a 30-second fault injection duration.

4 RESULTS AND ANALYSIS

This section presents and analyses the results obtained from the fault injection campaigns in both the C1 (network fault in the link between compute and storage node) and C2 (network fault in the link between controller and storage node) scenarios. The following results depict the impact of fault injection from three perspectives. Firstly, it was desired to understand how a network fault in the CMP would affect the performance of the applications running in the hosted VMs. Secondly, it was studied whether network faults in the network links of OpenStack can cause the applications in the hosted VMs to suffer service failures. Finally, we evaluated whether network faults in the CMP could cause unsuccessful operations, unanswered requests or data corruption, thus affecting the availability and correctness of the provided service.

4.1 Performance Impact

In scenario C1, the performance impact is presented and evaluated for the experiments that target the network between compute node and storage node within the CMP while the hosted VMs are executing the workload. We can observe in Figure 4 that as the duration of the network fault increases, and, most importantly, as the intensity of the fault increases (in this case, network congestion), the throughput is reduced. In the low intensity and duration configuration (Low NC, Low ET), the throughput is virtually similar to that when no network fault is injected. Whereas in the highest intensity and duration (High NC, High ET), the throughput is reduced to about half.

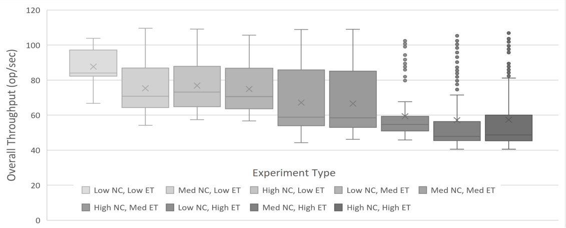

Figure 4: Throughput C1: Network Congestion (NC).

In Figure 5, which refers to the packet loss fault type, we see a similar trend of decreasing throughput as the intensity and the duration increase, whereas for network congestion faults the predominant factor was the fault intensity, here the estimated time appears to have a stronger influence (e.g., there is a noticeable break in throughput when moving from a medium duration to a high duration, while keeping the same intensity). The results for low packet loss show an unexpected pattern where throughput increases as fault duration increases. However this behaviour is reproducible and was experienced again in a second round of confirmatory experiments.

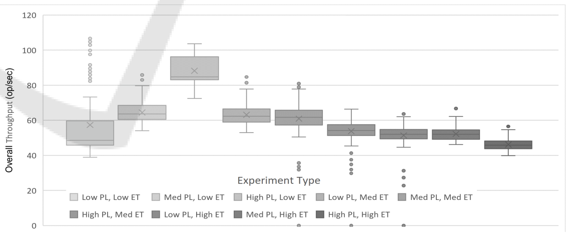

Figure 5: Throughput C1: Packet Loss (PL).

Figure 6 depicts the results for the network fault type that injects latency into the network. It shows that as the fault intensity increases, the deviation of the throughput greatly increases. At the same time, the mean throughput also decreases, but at a slower rate. The fault duration appears to have a smaller effect on the throughput than the intensity. This suggests that the network link studied in the scenario C2 has very little to no importance regarding the impact on the performance of the hosted VMs. Thus we can conclude that this network link does not require special redundancy or fault tolerance mechanisms. On the other hand, the network link studied in the scenario C1 has a noticeable impact and may need to receive specific fault tolerance mechanisms.

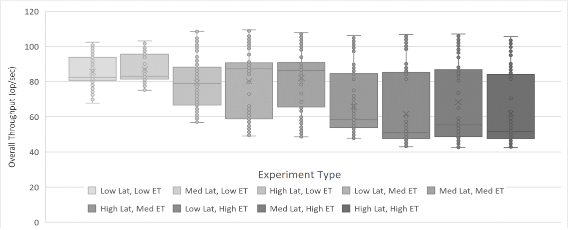

Figure 6: Throughput C1: Latency (Lat).

For our second scenario, C2, we have injected faults between the controller node and storage node. Figure 7 shows no impact of fault injection on this network link.

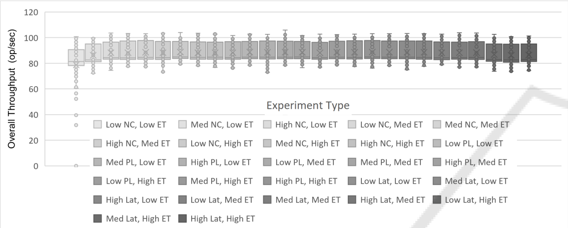

Figure 7: Throughput C2: Network Congestion (NC), Packet Loss (PL), and Latency (Lat).

4.2 Workload Operations Failures

After studying the impact on the performance of the applications, we focused on whether the provided service is being correctly performed. It is possible for the workload to show no performance effect, while producing invalid responses or failing to perform operations. In this subsection the focus is on whether the read and update operations of the YCSB workload were successfully completed or not.

For scenario C1, as shown in Figure 8, which refers to the failed read-operations, network congestion was the fault type with the least impact. Of the nine combinations where network congestion was used, the mean number of failed operations was 1. Although in absolute terms a single failed read operation may seem inconsequential, the important observation to retain is that there were failed operations. Ideally we would expect to see that a network fault in the CMP does not lead to any failed operation of the applications hosted over it. Since this is not what the results show, it means that migrating an application (e.g., a key-value store) from dedicated infrastructure to cloud computing will bring a decrease in its dependability.

The figure also depicts that packet loss was the fault type with the highest impact, followed by latency.

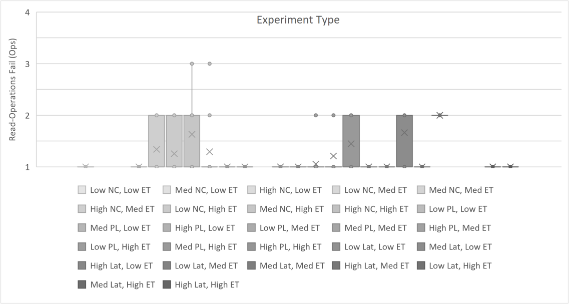

Figure 8: Read-operation Failed C1: All.

Figure 9 shows the number of update-operations that failed to successfully execute, for scenario C1.

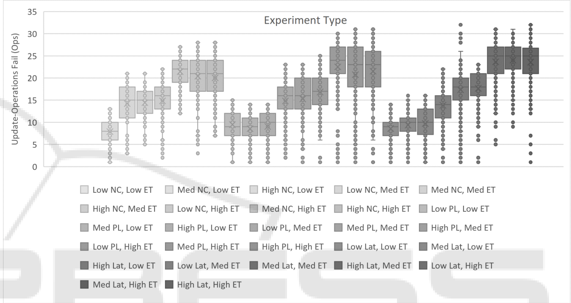

Figure 9: Update-operation Failed C1: All.

The results show that as the intensity and duration of the fault increases, the number of failed operations also increases noticeably. Some combinations showed no impact in the number of failed update-operations. For example, a low network congestion and low duration never caused any failed operation. However, the other two types of network faults (Latency and Packet Loss) caused failed operations even when the fault intensity was low. The absolute amount of failed operations varied. For example, when we have Low PL and Low ET, there is a mean of 10 failed operations. The mean number of failed operations reached as high as 25 (e.g., High PL and High ET). When talking about percentage of failed operations, the resulting percentages are very small. However, once again the important observation to take is that these failures occur at all.

For scenario C2, we have not found any operation failure for the read-operations and update-operations.

4.3 Operations Correctness Check

Other than verifying if the operations were completed successfully, we also verified if the operations returned the correct value. The verification of the correctness of operations is evaluated through the integrity check embedded in YCSB. This check verifies whether a read operation returns the expected result.

For scenario C1, the results in Figure 10 show that fault injection impacted the correct completion of the operations. In a correct execution, on average about 2500 read operations would be performed. Thus a value of *Integrity Verify Operations* lower than 2500 means that there were incorrect responses being returned.

In some cases, 80% to 90% of read operations failed to pass the correctness test. As an example, for low NC, Low ET, the mean incorrect operations are slightly less than 400 (84%), while in some runs it exceeded over 600 (∼76%).

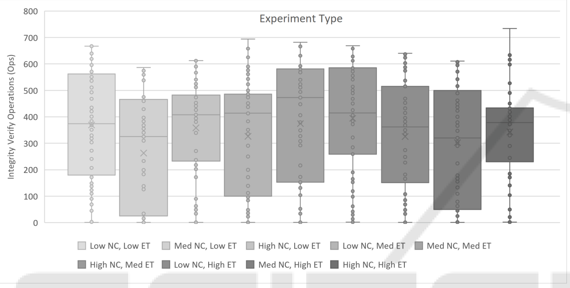

Figure 10: Integrity Verify Operations C1: Network Congestion.

Figure 11 depicts the number of successfully passed integrity verification operations when packet loss was injected. Here we can observe of the highest amount of failed checks, which occurred with high packet loss and high duration (High PL, High ET) and lead to only about 350 correct operations (86%).

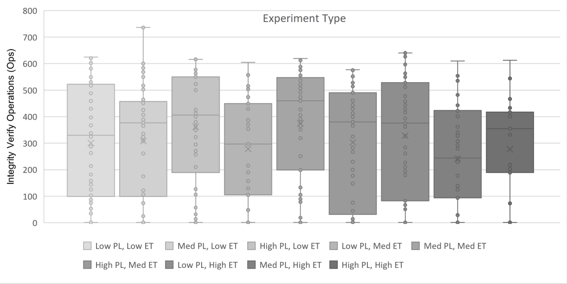

Figure 11: Integrity Verify Operations C1: Packet Loss.

Figure 12 depicts the same results but wrt. latency network faults. The most surprising result refers to low latency with both low and medium duration, which saw unusually high numbers of failed checks, whereas the remaining combinations caused none to only a few failed checks. This pattern was verified by repeating the same experiments twice and a root cause analysis will be performed as future work.

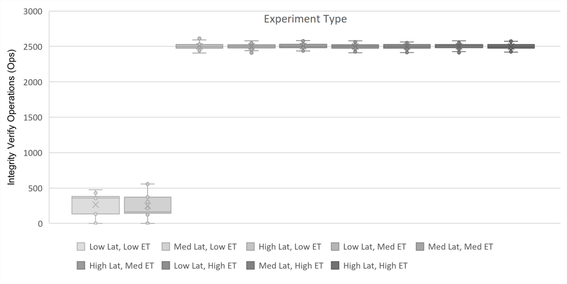

Figure 12: Integrity Verify Operations C1: Latency.

Once more, network faults injected in scenario C2 showed no impact. It demonstrates that the network link studied in C2 has no effect in the behaviour of the hosted applications, and showing that different network links of the CMP have different importance.

In summary. network faults in the CMP, specially those that cause congestion or packet loss in the network, can cause incorrect results to be returned, usually at a very high percentage. The explanation for this high percentage is likely found behind the fact that once one operation in the workload fails, then the next operations may be affected.

5 THREATS TO VALIDITY

The main threat to the validity of our results derives from the choices taken when building the experimental setup and the representativeness of the used fault models and its parameters. To mitigate the threats with respect to the experimental setup, we followed the OpenStack implementation guidelines (OpenStack, 2022) and fulfilled the minimum hardware and software requirements. The experiment setup is relatively simple. Real-world deployments are likely to be more complex and require more computing power.

We opted to use the YCSB benchmark paired with Cassandra as our workload. This workload represents a common cloud use case (key-value stores), however workloads of different types and areas need also to be evaluated, as the results may differ.

There is a need to balance experiment duration and accuracy. For that reason the durations chosen for the faults can be considered relatively short network failures in the real-world can often exceed many hours, however it would have been impractical to emulate such long failures. Nevertheless, we consider this not to be a significant problem because the impact of longer failures is going to be more pronounced than that of shorter failures.

6 CONCLUSIONS

Network faults are an unavoidable reality in any large-scale complex distributed systems, as is often the case for the infrastructure that supports cloud computing. In this paper, an experimental evaluation of the impact that network faults can have in a cloud computing system was performed. We focused on the CMP, more specifically in OpenStack, due to its popularity and due to being a complex distributed system where the network plays an important role. Fault injection of 3 different types of common network fault was performed with the help of Sidekick.

The results show that different network links have different importance in the impact experienced by the applications hosted on the infrastructure. The results show that network faults affecting the link between the compute node and the storage node can cause applications running on the infrastructure to fail to provide correct service, even if the network faults only lead to increased latency or reduced bandwidth. These results serve as the basis for future work on the development of fault tolerance mechanisms for CMPs that increase its tolerance of network faults while carrying minimal cost and overhead. Furthermore, as future work, we will carry out more experiments featuring more complex setups and setups where autoscaling is present, as to evaluate how network faults affect these setups.

ACKNOWLEDGEMENTS

This work is funded by the FCT - Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit - UIDB/00326/2020 or project code UIDP/00326/2020. This work is also supported by Project Reference ECSEL/0017/2019 and 876852-ECSEL-RIA-VALU3S, financed by Fundação para a Ciência e a Tecnologia, I.P./MCTES through national funds (PIDDAC) and funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Sweden, Italy, Spain, Portugal, Czech Republic, Germany, Austria, Ireland, France and Turkey.

REFERENCES

Avižienis, A., Laprie, J.-C., and Randell, B. (2004). Dependability and its threats: a taxonomy. In *Building the Information Society*, pages 91–120. Springer.

AWS (2022). Amazon ec2 instance types, https://aws.amazon.com/ec2/instance-types/, access date:2022-09-30

Cassandra (2022). Apache cassandra, https://cassandra.apache.org, access date: 2022-09-30.

Cerveira, F., Barbosa, R., Madeira, H., and Araujo, F. (2015). Recovery for virtualized environments. In *2015 11th European Dependable Computing Conference (EDCC)*, pages 25–36. IEEE.

Cocozza, F., López, G., Marın, G., Villalón, R., and Arroyo, F. (2015). Cloud management platform selection: A case study in a university setting. *Cloud Computing*, 2015:92.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154.

Cotroneo, D., De Simone, L., Liguori, P., Natella, R., and Bidokhti, N. (2019). Enhancing failure propagation analysis in cloud computing systems. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 139–150. IEEE.

Cotroneo, D., De Simone, L., and Natella, R. (2022). Thorfi: a novel approach for network fault injection as a service. *Journal of Network and Computer Applications*, 201:103334.

Dantas, J., Matos, R., Araujo, J., and Maciel, P. (2012). Models for dependability analysis of cloud computing architectures for eucalyptus platform. *International Transactions on Systems Science and Applications*, 8(5):13–25.

Ju, X., Soares, L., Shin, K. G., Ryu, K. D., and Da Silva, D. (2013). On fault resilience of openstack. In *Proceedings of the 4th annual Symposium on Cloud Computing*, pages 1–16.

Kumari, P. and Kaur, P. (2021). A survey of fault tolerance in cloud computing. *Journal of King Saud University-Computer and Information Sciences*, 33(10):1159–1176.

Lu, Y., Cheng, H., Ma, Y., and Wu, S. (2020). Research on the technology of power unified cloud management platform. In *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, volume 9, pages 770–773.

Natella, R., Cotroneo, D., and Madeira, H. S. (2016). Assessing dependability with software fault injection: A survey. *ACM Computing Surveys(CSUR)*,48(3):1–55

OpenStack (2022). Openstack- open source cloud computing platform software.

Pham, C., Wang, L., Tak, B. C., Baset, S., Tang, C., Kalbarczyk, Z., and Iyer, R. K. (2016). Failure diagnosis for distributed systems using targeted fault injection. *IEEE Transactions on Parallel and Distributed Systems*, 28(2):503–516.

Qi, Y., Fang, C., Liu, H., Kang, D., Lyu, B., Cheng, P., and Chen, J. (2021). A survey of cloud network fault diagnostic systems and tools. *Frontiers of Information Technology and Electronic Engineering*, 22(8):1031–1045.