IAGREE: Infrastructure-agnostic Resilience Benchmark Tool for Cloud Native Platforms

Paulo Souza[1][a], Wagner Marques[b], Rômulo Reis[c] and Tiago Ferreto[d]
Polytechnic School, Pontifical Catholic University of Rio Grande do Sul, Ipiranga Avenue, 6681 - Building 32, Porto Alegre, Brazil

Keywords: Cloud Computing, Cloud Native Platform, Resilience, Benchmark.

Abstract: The cloud native approach is getting more and more popular with the proposal of decomposing application into small components called microservices, which are designed to minimize the costs with upgrades and maintenance and increase the resources usage efficiency. However, microservices architecture brings some challenges such as preserving the manageability of the platforms, since the greater the number of microservices the applications have, the greater the complexity of ensuring that everything is working as expected. In this context, one of the concerns is to evaluate the resilience of platforms. Current resilience benchmark tools are designed for running in specific infrastructures. Thus, in this paper we present IAGREE, a benchmark tool designed for measuring multiple resilience metrics in cloud native platforms based on Cloud Foundry and running upon any infrastructure.

1 INTRODUCTION

Cloud computing is becoming more and more popular among multiple-sized organizations by providing benefits such as commodity of running large-scale applications with no need to care about issues involving local resources. Moreover, cloud also allows a better resources utilization through the pay-per-use pricing model, wherein cloud resources can be allocated and deallocated on demand, so users only have to pay for the resources they are actually being used (Nicoletti, 2016). By adopting such a strategy, customers avoid issues related to underprovisioning, in cases when an application gets more popular than expected resulting in revenue losses due to not having the computing resources enough to meet the demand, or overprovisioning, where applications do not meet the expectations, and the reserved resources end to not being used (Armbrust et al., 2010).

However, not every application architecture exploits the maximum potential of cloud environments. For example, the monolithic architecture in which the application logic is contained into a single deployable unit has shown to be a suitable solution for small systems. However, as the size of the application increases, tasks involving scaling or performing maintenance become more difficult due to challenges related to code readability and over-commitment to a specific technology stack. In this context, the Cloud Native approach arises with the proposal of taking advantage of the microservices architecture to allow on-premises applications to fully exploit the benefits of cloud computing by decomposing applications into microservices that could be deployed and scaled independently and do not need to be built with the same technology stack (Balalaie et al., 2015).

Cloud native platforms offers an additional abstraction layer over the infrastructure by the adoption of PaaS model. These platforms aim to simplify the build, deployment, and management of cloud native applications. These kind of applications are designed to exploits the advantages of the cloud computing delivery model.

Despite the benefits brought by cloud native, there are still concerns regarding topics like resilience, which is the ability to deliver and maintain a certain level of service despite the failure of one or several system components (Abbadi and Martin, 2011). Resilience issues in cloud services can lead to several negative consequences, for example, system instability, bottlenecks or downtime caused by unexpected

[a] https://orcid.org/0000-0003-4945-3329
[b] https://orcid.org/0000-0003-3304-5611
[c] https://orcid.org/0000-0001-9949-3797
[d] https://orcid.org/0000-0001-8485-529X

Souza, P., Marques, W., Reis, R. and Ferreto, T.
IAGREE: Infrastructure-agnostic Resilience Benchmark Tool for Cloud Native Platforms.
DOI: 10.5220/0007728503960403
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 396-403
ISBN: 978-989-758-365-0

workload can lead to business revenue losses (CA Technologies, Inc, 2018). Moreover, applications without proper resilience schemes can demand considerable costs with repair and replacement of cloud components or data loss (Vishwanath and Nagappan, 2010). In cases of applications that do not employ efficient resilience schemes, these kinds of failures can cause SLAs violations, which can negatively affect all the levels of cloud consumers: Final customers will have problems with trying to access the cloud services, Cloud service providers will suffer customer attrition, and cloud platform and cloud infrastructure providers will incur penalties for non-compliance of the accorded SLAs.

One of the possible steps to improve aspects such as resilience of a cloud native platform is monitoring its behavior under different circumstances. In this context, benchmarks play a significant role, since they can be used to measure and to compare different software or hardware configurations. Besides, these tools can also be used to implement improvements at the platform based on collected metrics. Several benchmark tools were proposed; however, there is a lack of benchmark tools designed to measure the resilience of cloud native platforms.

In this sense, this paper presents a benchmark tool designed for measuring the resilience of cloud native platforms. Based on this purpose, we present the following contributions:

- A review of the benefits and challenges brought by the cloud native approach.
- A discussion around the relevance of measuring resilience in cloud native platforms and metrics that can be analyzed to achieve this purpose.
- An infrastructure-agnostic resilience benchmark tool to measure the resilience of cloud native platforms based on Cloud Foundry.

The remaining of this article is organized as follows: In Section 2 we present the theoretical background about some of the current topics on cloud computing such as the need for strategies, which is addressed by approaches such as cloud native, that aims to improve the resources usage efficiency of applications running on the cloud. In Section 3 we present the proposed benchmark tool, which is validated in Section 4, where we present a Proof-of-Concept that includes the use of our proposal to perform experiments in a real-world cloud native platform. Then, in Section 5 we present other benchmark tools and compare them with our proposal, and in Section 6 we present final considerations and research topics to be addressed in future researches.

2 BACKGROUND

Cloud computing consists of a model that provides ubiquitous, configurable, on-demand access to computing resources which include networks, servers, storage, applications, and services through the Internet (Mell et al., 2011). It also provides cost reduction, since customers pay just for the resources that are consumed (Nicoletti, 2016). Due to these features, many organizations employ cloud services to meet the peak demand in their applications through services in the cloud (Gajbhiye and Shrivastva, 2014).

By dealing with on-premises software, customers must manage all levels of abstraction to make their applications up and running, dealing with manners of networking, storage, servers, runtimes, and so on. On the other hand, cloud computing provides service models that focus on delivering different levels of abstraction to meet the needs of customers more efficiently.

There are still many challenges in Cloud Computing. For instance, the interoperability among different cloud services is still a big challenge. Also, Cloud Computing is powered by virtualization, which provides several independent instances of virtual execution platforms, often called virtual machines (VMs). However, applications have lower performance on VMs than on physical machines. The overhead generated by the virtualization layer (hypervisor) is one of the leading concerns in the context of virtualized environments. Therefore, identifying and reducing such overhead has been the subject of several investigations (Li et al., 2017; SanWariya et al., 2016; Chen et al., 2015).

In this context, Cloud Native approach emerges with the proposal of decomposing applications into small components designed to operate independently called microservices. Each microservice is executed on a container, so its possible to upgrade or replace a microservice without having to modify the entire application. Such modularity allows minimizing the costs with upgrades and maintenance and increase the resources usage efficiency (Amogh et al., 2017). On the other hand, running microservices in separated containers brings some challenges such as preserving the manageability of the platforms, since the higher the number of containers an applications has, the greater the complexity of ensuring everything is working as expected. Due to these challenges, there is a concern on increasing the customers' trust in the services provided.

Service Level Agreements (SLAs) (which are contractual agreements between service providers and its customers) play a significant role in this context, since they specify the quality of service (QoS) guaranteed by the providers so customers can verify if the delivered QoS befits that one was accorded (Patel et al., 2009). One element which is considered in SLAs is resilience, which is the ability to deliver and maintain a certain level of service despite the failure of one or several system components (Abbadi and Martin, 2011).

There are tons of elements that can affect the operation of cloud services and must be considered when building resilience mechanisms. One of the current leading causes of failures in cloud computing services is human errors, which is characterized by failures caused by human actions (e.g., a cloud operator accidentally erases a database registry required by a system vital component). Other causes of system failures are disasters (like earthquakes and tornadoes) or software failures (bugs or errors caused by malicious software) which can also lead to critical failure scenarios (Colman-Meixner et al., 2016).

Resilience issues in cloud services can lead to several negative consequences, for example, system instability, bottlenecks or downtime caused by unexpected workload can lead to business revenue losses. Moreover, applications without proper resilience schemes can demand considerable costs with repair and replacement of cloud components or data loss (Vishwanath and Nagappan, 2010). In cases of applications that do not employ efficient resilience schemes, these kinds of failures can cause SLAs violations, which can negatively affect all the levels of cloud consumers: Final customers will have problems with trying to access the cloud services; Cloud service providers will suffer customer attrition, and cloud platform and cloud infrastructure providers will incur penalties for non-compliance of the accorded SLAs.

Cloud Computing takes advantage of the traditional IT model by providing dynamic resource provisioning instead of delivering computational power according to the peak demand (which leads to unnecessary costs with computing resources). However, failures are an undesired but inherent aspect when scaling applications in the cloud (Zhang et al., 2010). Moreover, infrastructure manners such as the adopted virtualization strategies can impact the resilience of applications.

Given that, one of the key aspects on achieving resilience in cloud environments is building a resilient infrastructure that implements failure prevention techniques like the isolation of applications' layers and components so even if some components get compromised the rest of the system will be able to operate normally (Salapura et al., 2013). In this context, the relevance of measuring resilience stood out since it is important to verify the efficiency of the implemented resilience strategies. Accordingly, as shown in Figure 1, some resilience metrics should be taken into account to measure a system or a platform:

- **Mean Time To Failure (MTTF):** often known as uptime, MTTF refers to the average amount of time that a system is available. MTTF is widely used in SLA contracts to indicate how reliable a system or a platform is based on the amount of time it has spent without failing.
- **Mean Time To Detect (MTTD):** refers to the average amount of time it takes to detect a failure. This metric is useful to measure the efficiency of monitoring and incident management tools.
- **Mean Time To Repair (MTTR):** this metrics denotes to the average time spent to repair a failed system. MTTR is a valuable metric to find ways to mitigate downtime.
- **Mean Time To Between Failures (MTBF):** refers to the average time that elapses between failures. It is a useful metric to predict the reliability and availability of a platform accurately.

3 IAGREE ARCHTECTURE DESIGN

IAGREE is an infrastructure-agnostic resilience benchmark tool for cloud native platforms. It allows the user to measure the resilience of Cloud-Foundry based platforms by simulating failing scenarios of a sample REST application while receiving user requests (Figure 2). Our tool use commands provided by the Cloud Foundry Command Line Interface (cf CLI), which is used by Cloud Foundry and its based platforms and allows the deployment, management, and monitoring of applications. This makes IAGREE compatible to any platform based on CF. Besides that, this benchmark tool also enhances the agnostic to infrastructure characteristic from Cloud Foundry.

IAGREE uses features provided by a Cloud Foundry component called Loggregator, which is responsible for collecting and streaming logs from the applications and the platform itself.

Our proposal takes advantage of Loggregator's architecture to analyze the resilience provided by the platform while applications are receiving requests from multiple users and failing events occur. To simulate real user accesses, IAGREE generates multiple concurrent requests using Siege, an open source tool that allows simulating multiple user HTTP requests.

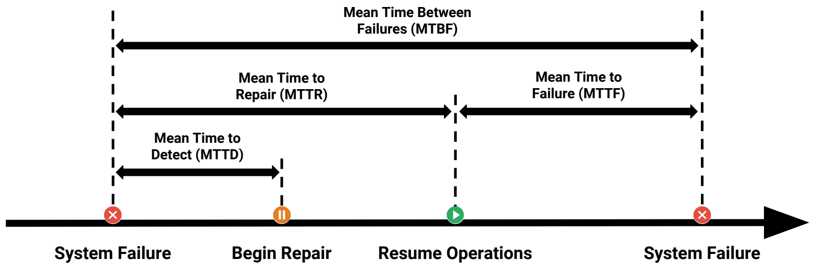
Mean Time Between Failures (MTBF)
Mean Time to Repair (MTTR)
Mean Time to Failure (MTTF)
Mean Time to Detect (MTTD)
System Failure
Begin Repair
Resume Operations
System Failure

Figure 1: Different measures covered by resilience metrics.

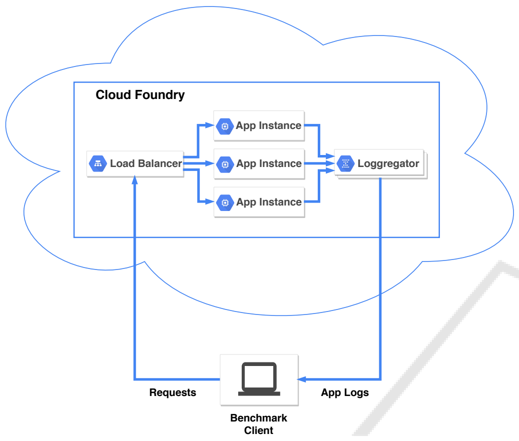

Figure 2: IAGREE benchmark tool design.

As illustrated in Figure 3, upon running IAGREE, the first step is to ask for API endpoint and credentials (email and password) in order to establish a connection to the platform. As soon as the benchmark confirms that cf CLI was able to authenticate the user into a Cloud Foundry environment, it deploys a sample application to receive the requests. Once the application deployment is finished, the benchmark starts a log stream from the Loggregator and asks the user to define three parameters: *i)* the amount of time the application will be tested; *ii)* how many simulated users will try to access the application simultaneously; and *iii)* how many fails per minute the application will experience. Once those parameters are informed, the benchmark tells Siege the desired configuration and the tests start. In the meantime, IAGREE may sends requests to a route of the application which is configured to crash the whole instance. since there is the load balance component, any instance can stop. This process is described in Algorithm 1.

As the benchmark finishes running the tests, it parses the data from the log stream and presents the results denoted by the following metrics: MTTF, MTBF, MTTD, MTTR, mean response time per requests, transactions per second, and throughput.

```
while $time_{current} <= time_{total}$ do
    send $N$ simultaneous requests
    foreach $failureInstant\ f_i \in F$ do
        send a request that will crash an app
          instance
    end
end
```

Algorithm 1: Strategy adopted by the proposed benchmark to measure resilience of cloud native platforms.

4 PROOF-OF-CONCEPT EXPERIMENTS

We conducted a proof-of-concept running our proposal on Pivotal Web Services (Pivotal Web Services, 2018), which is a Cloud-Foundry based platform that uses EC2 servers located in Northern Virginia (United States). We did choose PWS since it automatically handles distributing multiple instances applications across multiple availability zones. We performed tests in 8 different configurations gathering in each of them MTTF, MTBF, MTTD, MTTR, mean response time per requests (s), transactions per second, and throughput (MB/s). Table 1 summarizes the set of configurations explored by our experiments.

Table 1: Tested specification.

App Instances	Total Time	Failures per Minute	Concurrent Users
2	10 minutes	2	50
			100
		4	50
			100
4		2	50
			100
		4	50
			100

4.1 Mean Time To Failure

The results presented in Figure 4 showed that the impact on an application's availability caused by the number of failures per minute varies according to the number of instances an application have, especially

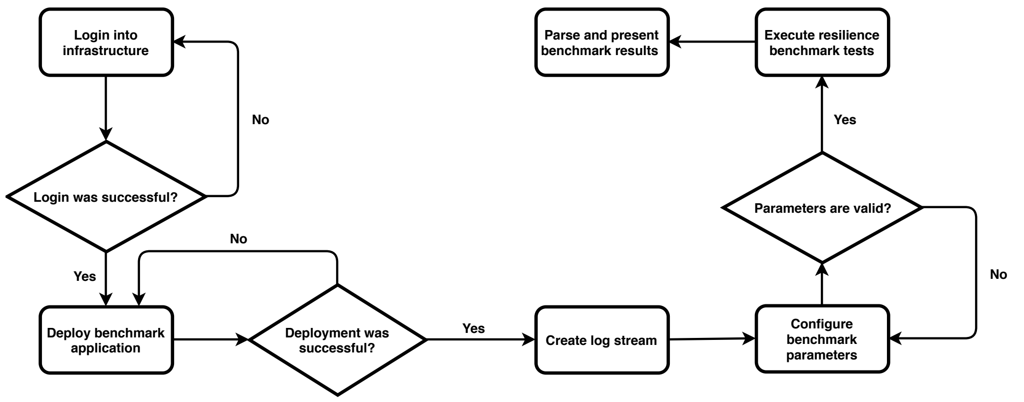
Login into infrastructure
No
Login was successful?
Yes
No
Deploy benchmark application
Deployment was successful?
Yes
Create log stream
Configure benchmark parameters
Parameters are valid?
No
Yes
Execute resilience benchmark tests
Parse and present benchmark results

Figure 3: IAGREE process flow.

as the number of users accessing the application increases. In other words, the higher the number of users accessing the application, the higher is the relevance of scaling mechanisms since if an application crashes due to insufficient resources, its availability could be severely impacted. Moreover, in the scenarios with 2 failures per minute, the application with 2 instances suffered a 44% availability loss as the number of users accessing it increased from 50 to 100. On the other hand, the application with 4 instances only got an 8% availability loss. These results indicate that the impact caused by a given number of failures could be significantly higher in applications with few instances.

4.2 Mean Time To Detect

The results in Figure 5 regarding the average time that the platform took to detect failures show that in scenarios where applications are suffering from several and constant crashes, the higher the number of users accessing the application, the greater the time the platform would take to detect the failures. Especially in scenarios where failures do not crash the entire instance (e.g., the web server stops working, but the container where it is running continues to work) and therefore are undetectable for instance healing tools, monitoring mechanisms must parse more massive amounts of application logs to detect failures as the number of requests increases, which could lead to delayed detection of application failures. Besides, detecting the failures in the 4 instances application took 12% longer, since the higher the number of instances the app has, the higher the complexity of ensuring that everything is working as expected.

4.3 Mean Time Between Failures

The results presented in Figure 6 highlighted the correlation between the number of application instances and the frequency of failure occurrences. In general terms, the mean time between failures is highly dependent on the number of failures per minute specified to the benchmark. However, as we can see comparing the results of 2 and 4 instances, the bigger the number of instances that an application have, the smaller the interval between failures. This happens because load balancers have a buffer that stores the incoming requests to be distributed across the application instances, then in scenarios with multiple simultaneous requests, increasing the number of application instances reduces the delay to a request to be processed. As the proposed benchmark simulates an application failure by sending a request to a specific application URL configured to crash the web server, increasing the number of instances of the sample application resulted in few waiting time to those requests to be processed, which in its turn reduced the interval between failures.

4.4 Mean Time To Repair

According to the results presented in Figure 7, in the experiments with the 2 instances application, increasing the number of failures from 2 to 4 caused a delay to the platform to repair the application failures. However, such a phenomenon does not appear in the experiments with the 4 instances application. Besides, considering that MTTR covers the time to detect failures (see Figure 1), the results also highlighted the negative impact of having too frequent failures in applications with few instances, since the 2 instances application took longer to be repaired despite demanding less time to detect failures than the 4 instances application.

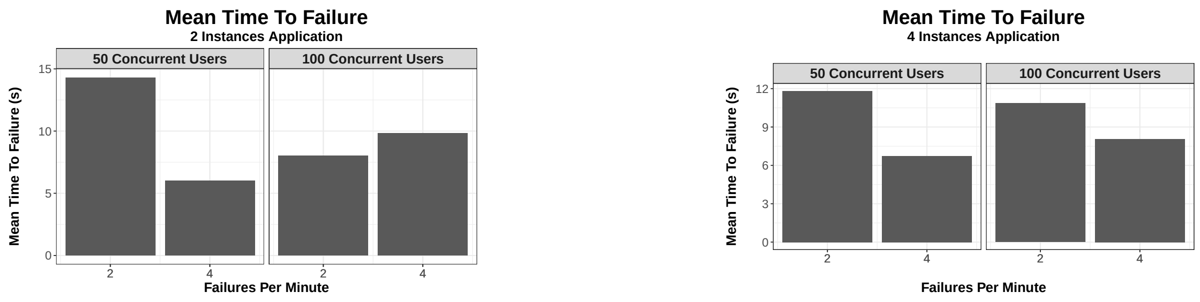

Figure 4: Mean Time To Failure.

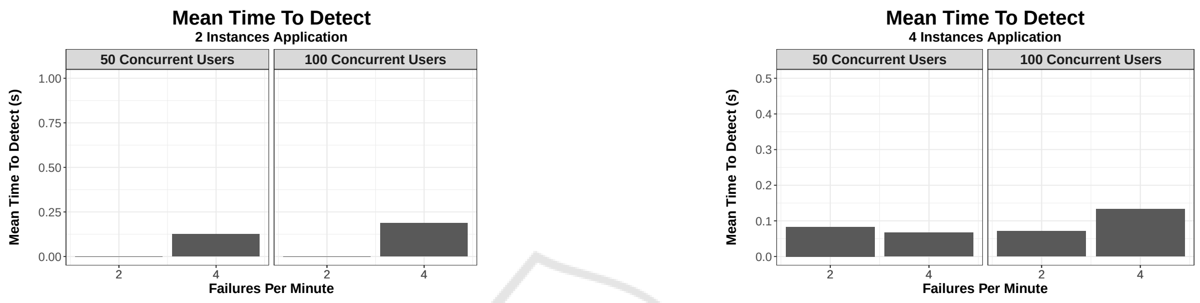

Figure 5: Mean Time To Detect.

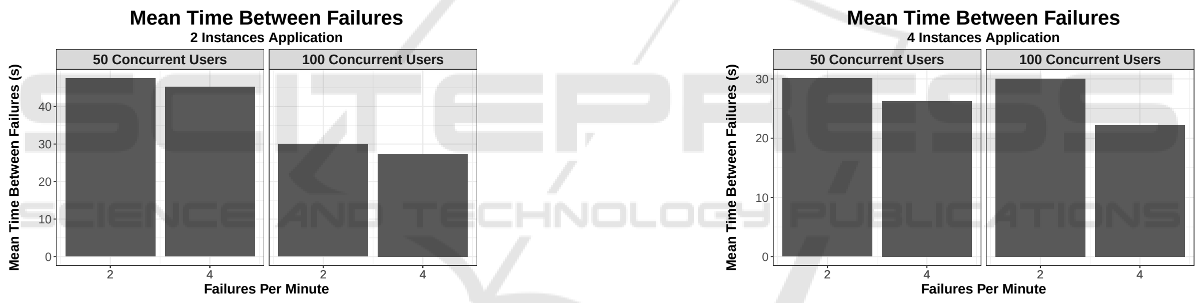

Figure 6: Mean Time Between Failures.

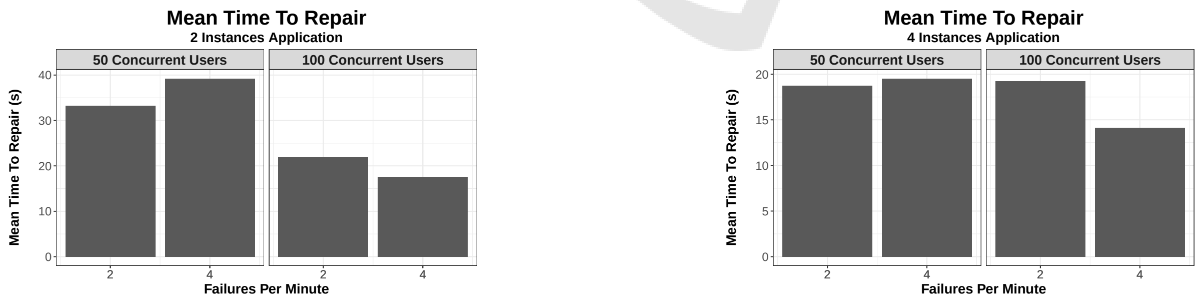

Figure 7: Mean Time To Repair.

5 RELATED WORK

Rally (Rally, 2018) provides a framework for evaluating the performance of OpenStack components as well as full production OpenStack cloud deployments. Rally automates and unifies the deployment of multiple OpenStack nodes, cloud verification, testing, and profile creation. Rally generically does that, making it possible to check whether OpenStack will meet the business requirements.

Cloud CMP (Li et al., 2010) is a benchmark tool developed to provide a systematic comparator of the performance and cost of cloud providers. CloudCMP measures the elasticity, persistent storage and networking services offered by a cloud along with metrics that directly reflect on the performance of customer applications. This tool was tested in various cloud providers, including Amazon Web Services, Microsoft Azure, Google AppEngine and Rackspace CloudServers (LI et al., 2010).

HiBench (HiBench, 2018) is an open source benchmark suite for Hadoop. This tool consists of synthetic micro-benchmarks and real-world applications. The application currently has several workloads classified into four categories: Microbenchmarks, Web Search, Machine Learning, and Data Compression. The microbenchmarks provide several algorithms such as Sort, WordCount, Sleep, enhanced DFSIO and TeraSort provided by Hadoop. These benchmarks are used to represent a subset of real-world MapReduce jobs. Web Search uses PageRank and Nutch indexing benchmark tools. These benchmark tools are used since large-scale search indexing is one of the most significant uses of MapReduce. The machine learning workloads provide Bayesian Classification and K-means Clustering implementations. It worth noting that HiBench also provides several other machine learning benchmark alternatives such as Linear Regression and Gradient Boosting Trees.

Cloud Suite (CloudSuite, 2018) is an open source benchmark tool for cloud computing focused on providing scalability and performance evaluation on real-world setups. Cloud Suite has a suite of benchmarks that represent massive data manipulation with tight latency constraints such as in-memory data analytics. Among the tasks included in this set of benchmarks, there are MapReduce, media streaming, SAT solving, web hosting, and web search tasks. Cloud Suite is compatible with private and public cloud platforms, and the authors emphasize that it is integrated into Google's PerfKit Benchmarker[1] that helps to automate the benchmarking process.

Yahoo! Cloud Serving Benchmark (Cooper et al., 2010) can be used to perform elasticity and scalability evaluation in different cloud platforms such as Amazon AWS, Microsoft Azure, and Google App Engine. The authors defined two benchmark tiers for evaluating cloud serving systems: The first tier focuses on the latency of requests when the database is under load. It aims to characterize this trade-off for each database system by measuring performance as the user requests increase. The second tier focuses on examining the impact on performance as more machines are added to the system. In this tier, there are two metrics: Scaleup and Elastic Speedup.

Chaos Monkey (Chaos Monkey, 2018) is a benchmark tool created by Netflix that randomly terminates instances of virtual machines and containers inside the production environment to simulate unexpected failure events. Chaos Monkey follows the principles of chaos engineering. Chaos Monkey was designed to work with any backend that provides support for Spinnaker, which is an open source, and a multi-cloud continuous delivery platform for releasing software changes with the intention to give velocity and confidence.

To the best of our knowledge, our proposal (IAGREE) is the only infrastructure-agnostic benchmark tool that allows measuring the resilience of any cloud native platform based on Cloud Foundry. Table 2 summarizes the differences between IAGREE and the previous work.

6 CONCLUSIONS

Companies have adopted cloud computing because of the functionalities it provides in a simple way and also for the value that is lower than having an on-premise solution. The cloud native approach was driven by cloud computing and container technology, though there is still resistance to migrating to cloud native platform solutions. One of the reasons for this resistance is the lack of how to reliably evaluate which platform provides the resilience requirements for the applications.

Thus, in this paper, a benchmark tool for resilience for cloud native platforms called IAGREE was proposed. This tool supports platforms based on Cloud Foundry like the Pivotal Application Service. Besides, it inherits the ability of the CF to be agnostic to the infrastructure provider, running on both the industry's top cloud service providers (AWS, GCP, Microsoft Azure) and on-premise.

A set of experiments was also performed to validate the proposed tool, using Pivotal Web Services as a platform provider. The results obtained by IAGREE were analyzed based on the following resilience metrics: Mean Time To Repair, Mean Time Between Failures, Mean Time To Detect, Mean Time To Failure. As future work, we intend to perform experiments on a larger scale and with different configurations. We also intend to execute them using different infrastructure providers in order to compare their performance.

[1]PerfKit Benchmarker. Available at: <github.com/GoogleCloudPlatform/PerfKitBenchmarker>.

Table 2: Comparison between IAGREE and other cloud benchmark tools.

Benchmark	**Evaluated Metrics**	**Output**	**Supported Infrastructures**
Rally	Elasticity	CLI and HTML	Infrastructure Agnostic
Cloud CMP	Elasticity	JSON	Amazon Web Services, Microsoft Azure, Google Cloud Platform
HiBench	Scalability and Performance	CLI	Amazon Elastic Compute Cloud, Microsoft Azure
Cloud Suite	Scalability and Performance	CLI	Infrastructure Agnostic
Yahoo! Cloud Serving Benchmark	Elasticity and Scalability	CLI and HTML	Amazon Web Services, Microsoft Azure, Google Cloud Platform
Chaos Monkey	Resilience	HTML	Amazon Web Services, Microsoft Azure, Google Cloud Platform, Kubernetes-based Platforms
IAGREE	***Resilience***	***CLI and JSON***	***Infrastructure Agnostic***

ACKNOWLEDGEMENTS

This work was supported by the PDTI Program, funded by Dell Computadores do Brasil Ltda (Law 8.248 / 91).

REFERENCES

Abbadi, I. M. and Martin, A. (2011). Trust in the cloud. *information security technical report*, 16(3-4):108–114.

Amogh, P., Veeramachaneni, G., Rangisetti, A. K., Tamma, B. R., and Franklin, A. A. (2017). A cloud native solution for dynamic auto scaling of mme in lte. In *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017 IEEE 28th Annual International Symposium on*, pages 1–7. IEEE.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50–58.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2015). Migrating to cloud-native architectures using microservices: an experience report. In *European Conference on Service-Oriented and Cloud Computing*, pages 201–215. Springer.

CA Technologies, Inc (2018). Available at: https://bit.ly/2BVZhsx. Accessed: 2018-12-24.

Chaos Monkey (2018). Chaos monkey. Available at: https://github.com/Netflix/chaosmonkey. Accessed: 2018-12-24.

Chen, L., Patel, S., Shen, H., and Zhou, Z. (2015). Profiling and understanding virtualization overhead in cloud. In *Parallel Processing (ICPP), 2015 44th International Conference on*, pages 31–40. IEEE.

CloudSuite (2018). Cloudsuite. Available at: https://github.com/parsa-epfl/cloudsuite. Accessed: 2018-12-24.

Colman-Meixner, C., Develder, C., Tornatore, M., and Mukherjee, B. (2016). A survey on resiliency techniques in cloud computing infrastructures and applications. *IEEE Communications Surveys & Tutorials*, 18(3):2244–2281.

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. ACM.

Gajbhiye, A. and Shrivastva, K. M. P. (2014). Cloud computing: Need, enabling technology, architecture, advantages and challenges. In *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference-*, pages 1–7. IEEE.

HiBench (2018). Hibench suite. Available at: https://github.com/intel-hadoop/HiBench. Accessed: 2018-12-24.

Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM.

Li, Z., Kihl, M., Lu, Q., and Andersson, J. A. (2017). Performance overhead comparison between hypervisor and container based virtualization. In *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on*, pages 955–962. IEEE.

Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

Nicoletti, B. (2016). Cloud computing and procurement. In *Proceedings of the International Conference on Internet of things and Cloud Computing*, page 56. ACM.

Patel, P., Ranabahu, A. H., and Sheth, A. P. (2009). Service level agreement in cloud computing.

Pivotal Web Services (2018). Available at: https://pivotal.io/platform/pivotal-application-service. Accessed: 2018-12-24.

Rally (2018). Rally benchmark. Available at: https://github.com/openstack/rally. Accessed: 2018-12-24.

Salapura, V., Harper, R., and Viswanathan, M. (2013). Resilient cloud computing. *IBM Journal of Research and Development*, 57(5):10–1.

SanWariya, A., Nair, R., and Shiwani, S. (2016). Analyzing processing overhead of type-0 hypervisor for cloud gaming. In *Advances in Computing, Communication, & Automation (ICACCA)(Spring), International Conference on*, pages 1–5. IEEE.

Vishwanath, K. V. and Nagappan, N. (2010). Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204. ACM.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18.