An Efficient and Reliable Multi-Cloud Provider Monitoring Solution

(Article begins on next page)

An Efficient and Reliable Multi-Cloud Provider Monitoring Solution

Andrea Sabbioni, Armir Bujari, Luca Foschini, Antonio Corradi

University of Bologna, Computer Science and Engineering Department, Bologna, Italy

Email: name.surname@unibo.it

Abstract—Cloud computing has transformed the way IT services are provisioned and consumed, shifting the burden of configuration and management to the cloud provider. A lot of research has been conducted in the field aimed at optimizing and/or devise new in-cloud service delivery models. Yet, service outages are a norm rather than an exception. A natural evolution for application developers, especially for those applications that must meet high availability requirements, is to rely on resources and services provisioned by multiple cloud simultaneously. The so-called multi-cloud, distributed deployment model coupled with a monitoring solution spanning multiple domains could help in a timely identification and isolation of faulty components, alleviating service impairment issues. To this end, we propose an extension to NoMISHAP, a Platform as a Service (PaaS) multi-cloud middleware, introducing a cross layer monitoring solution for use in distributed cloud environments. Experimental results show the effectiveness of our approach and its ease of adoption for management tasks.

Index Terms—PaaS, middleware, Multi-cloud, Monitoring.

I. INTRODUCTION

Nowadays, cloud computing underpins a vast number of IT services, allowing for an efficient management and scaling of resources. In this context, the Platform as a Service (PaaS) computing model and the emerging technological ecosystem promises to reduce and to facilitate the evolution of business services. This computing model provides the user with a development framework, while shifting the burden of infrastructure management and control to the cloud provider.

In most cloud platforms, PaaS is juxtaposed to a layer of services provided in a pay-per-use fashion which belong to another complementary model of cloud computing commonly known as Mobile Backend as a Service or Backend as a Service (BaaS). This layer is often use by programs and functions in PaaS to further speed up the development, leveraging on solid and optimized services [1]. Over the years, different providers have developed increasingly advanced and heterogeneous BaaS services, allowing a differentiation between the vendors potentially causing a lock-in effect for the customers. This problem is not necessarily intentional but rather a reflection of the vast design space in building a cloud solution. Moreover, increasingly the availability of the workloads hosted on cloud platforms is becoming more and more relevant for many businesses, and any shortage or service discontinuity can cause substantial losses.

Addressing the aforementioned challenges, both academia and industry have devoted a lot of time and effort focusing on solutions ranging from novel service architectures and deployment models, programming abstractions, to specific purpose tools used for service lifecycle management and orchestration. In this front, academic research has been primarily focused more on theoretical forefront, investigating areas ranging from service portability and composition, semantic interoperability, placement and migration strategies, to software engineering aspects [2], [3]. On the industry front, instead, cloud providers have put in place best practices, best exploiting the potential of their solutions; however, all the proposals are typically limited to a single cloud provider and are not suitable to address large scale and multi-region deployment scenarios [4], [5]. Despite the cloud services market's steady growth, no cloud solution provider has really yet proven to be immune from outages and major episodes of service disruptions are a norm rather than an exception.

The relevant point of view for cloud-based application developers, instead, should be a more external perspective where, to overcome such service outages, they could leverage and use together multiple services provisioned in multiple cloud environments. The outcome is a more resilient and lock-in free multi-cloud ecosystem. Indeed, a promising path that is being considered is the use of the so-called multi-cloud integration pattern, a solution that simultaneously exploits multiple cloud platforms for service delivery [6]. While this approach usually implies an overhead in terms of orchestration complexity, the implementation of a practical multi-cloud approach could facilitate the reach of important business objectives like high availability, failover and lock-in avoidance. In this context, an effective and complete monitoring solution spanning different layers of a service plays a key role in fulfilling the users requirements, serving as the basis for an (semi)automatic identification and mitigation of faults [7].

To this end, we set to design and implement a pervasive monitoring solution for use in the multi-cloud PaaS context. Without loss of generality, we chose to extend NoMISHAP, a state-of-the-art multi-cloud middleware solution [8]. Our proposal aims to provide a complete monitoring suite able to collect information across all the layers of a multi-cloud PaaS. The gathered monitoring data can serve as input to an (semi)automatic failure recovery module and/or cloud operators, aiding to recover and/or prevent potential hazardous conditions in an efficient way. Finally, it is noteworthy to point out, that our proposal does not substitute existing ones offered by cloud providers. On the contrary, it integrates with them

and fills-in eventual gaps, providing additional information spanning multiple layers.

The structure of the article is as follows: in Sec. II we discuss some technical challenges toward a multi-PaaS monitoring solution along with a survey of related work in the context. Next, Sec. III presents the multi-cloud monitoring solution for high availability based on NoMISHAP while Sec. IV presents the results of the experimentation assessment carried out in a realistic multi-PaaS environment. Finally, Sec. V draws the conclusions.

II. BACKGROUND AND RELATED WORK

We start by discussing some technical challenges in provisioning a multi-PaaS monitoring solution, hinting towards the adopted proxy-based approach. Next, follows a concise survey of literature work in the context along with an overview of the NoMISHAP middleware solution.

A. Toward a Multi-Cloud Monitoring Solution

Most cloud offerings are equipped with complex monitoring solutions which are usually locked-in to their infrastructures and often cannot be externalized to span different cloud environments. Moreover, an integration between the different solutions is hindered by the heterogeneity and lack of standardization across vendors.

Considering the PaaS context, one cannot rely on traditional technologies and approaches such as software defined networking and cloud federation, since no control over the infrastructure at an IaaS level can be assumed and taken for granted [9], [10]. While different cloud providers, provide both IaaS and PaaS offerings, in a PaaS approach we cannot make any *a-priori* assumption or have a knowledge of where the services will be deployed. The same argument stands for the BaaS service offerings exploited by the applications.

In order to address the integration aspect while at the same time hide the complexity of the multi-cloud PaaS, one common engineering approach is that of exploiting one or multiple proxies interfacing the individual environments. The proxy, in fact, is a key element that can exploited to hide the complexity and offer transparency on the client side. This component could in its own turn be hosted on a PaaS platform or reside elsewhere.

In a heterogeneous multi-cloud PaaS infrastructure, the proxy can act as a layer of adaptation at the PaaS side, while, at the client side it can be exploited to transparently implement mechanisms such as load balancing and fault tolerance. Delegating all this complexity to a proxy agent allows also to implement some advanced features such as synchronization of infrastructure state and offered services. In these settings, the user is presented a unified view of resources and shielded from the underlying heterogeneity and complexity of the platforms.

B. The Multi-PaaS Ecosystem

We focus on those solutions that provide a cross-cloud monitoring system as we consider it a key feature for an effective strategy in the pursuit of high availability in cloud environments. As a premise, at the moment of this writing, there is not a comprehensive solution that can effectively deal with multi-cloud PaaS.

A seminal contribution in the context is presented by the open source project OpenNebula [11]. The solution offers a cloud computing toolkit for managing heterogeneous distributed datacenters. It can orchestrate storage, network, monitoring, and security technologies to deploy multi-tier services as virtual machines on distributed infrastructures, combining both datacenter resources and remote cloud resources, according to allocation policies. This project presents a lot of interesting features and a powerful toolkit for multi-cloud environments. However, the proposal targets the integration at the IaaS layer only.

Pursuing a similar objective, different academic projects have addressed the challenging task of provisioning software stacks, libraries for frameworks aimed at the deployment, monitoring and adaptation of cloud-based systems at the IaaS and/or PaaS layers.

FraSCAti is a solution which exploits the extended service component architecture, a technology agnostic standard for distributed service-oriented applications, to deploy federated multi-cloud PaaS infrastructures [12]. Its open service model extends to both the PaaS environment and the SaaS applications hosted on top of it.

soCloud extends the FraSCAti execution engine inheriting its capabilities of a multi-cloud PaaS solution [13]. In addition, it introduces additional features which allow for the management of application/service portability, provisioning, resilience and high availability across multiple clouds.

Cloud4SOA introduces a broker-based architecture, enabling a scalable approach to heterogeneous PaaS offerings of semantic interconnection between different cloud providers sharing the same technology [14]. The architecture is equipped with management and monitoring services providing the appropriate flexibility to handle public, private and hybrid deployment models.

Both FraSCAti and Cloud4SOA are elaborated examples proposing a multi-layer integration approach for the multi-cloud domain, however, they require the deployment of additional physical resources, either on-site or off-site, hosting the control logic. Moreover, we argue that the integration of applications and services in the above settings, poses a high barrier of entry for developers, inhibit the effective usage of the frameworks.

Addressing heterogeneous environments, mOSAIC focuses on both IaaS and PaaS layers by allowing applications to specify their service requirements through an ontology [15]. The proposal relies on a brokering mechanism exploited to search for the best set of services meeting application requirements. The main outcome of the mOSAIC project is a common communication API for multi-cloud resources.

C. The NoMISHAP Middleware

NoMISHAP aims at providing a transparent support for high availability exploiting multiple PaaS provider simulta-
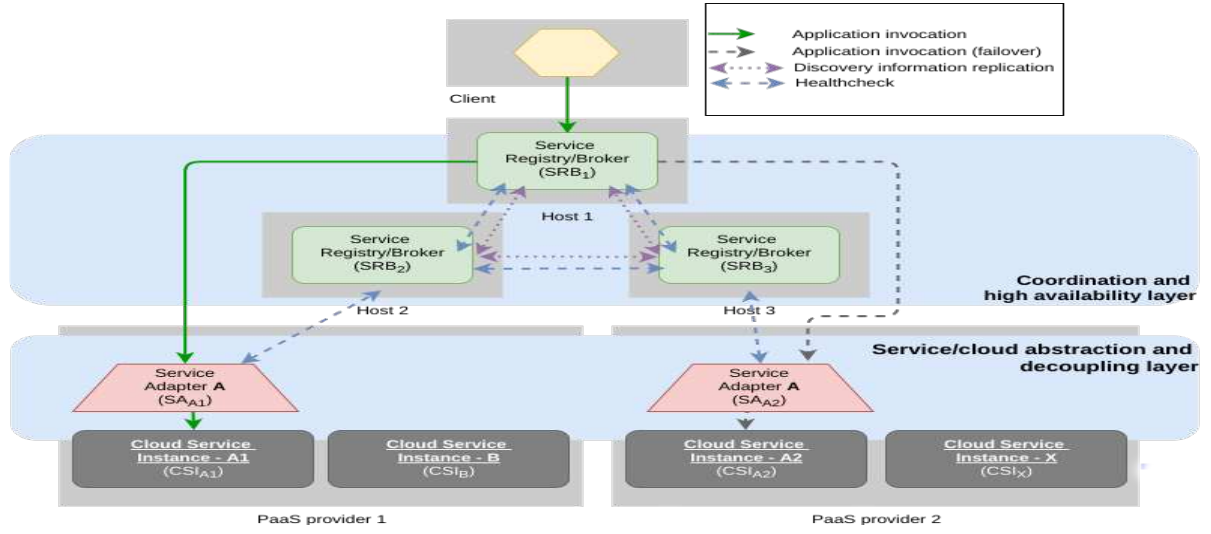
Fig. 1. NoMISHAP architecture [8].

neously [8]. This is achieved thanks to a built-in abstraction and adaption layer of services and functions that simplify and unify the access to services available by the underlying PaaS. As a consequence, cloud developers can concentrate only on essential core business code, developed once and deployed on multi-PaaS solutions.

In essence, the architecture of NoMISHAP can be summarized in two macro layers as shown in Figure. 1 and concisely summarized below:

1) The coordination and high availability layer which is responsible to intercept application invocations, redirecting them to the corresponding service while applying policies for load balancing and failover management.
2) The service/cloud abstraction and decoupling layer provides a uniform API for heterogeneous applications, enabling homogeneous invocations between different providers. This layer manages interactions with cloud providers, fetches eventual configuration files and/or environment variables needed to properly interact with services and in order to expose a homogeneous REST API.

NoMISHAP proposes a novel external perspective not limited to one cloud provider by including all main service abstraction and adaptation, service brokering functions, and internal middleware components. This constitutes the basis for our monitoring proposal.

Compared to prior effort in the domain, this proposal poses a lower barrier of entry, allowing for the integration of individual PaaS services spanning multiple environments through the use of lightweight proxies. Moreover, the proposal is application and cloud provider-agnostic providing cloud-based services with both an architectural model and a ready-made implementation granting cloud-based applications high availability across cloud vendors.

III. A MULTI-CLOUD MONITORING SOLUTION

We start by first identifying the monitoring data sources and than discuss the approach and tools adopted constituting the data processing pipeline.

A. Monitoring Data Sources

Figure 2 depicts a component-oriented approach of our proposal. Starting from the bottom, in order to maintain a complete and consistent view of the PaaS layer, we need to collect metrics pertaining to BaaS service components the application relies on and the NoMISHAP middleware components deployed on the PaaS. Unfortunately, only few PaaS providers expose state statistics related to the BaaS services. To this end, filling in this gap, we can consider as a good representation of performance and state, the historical data (transparently) collected by the PaaS proxy while satisfying user requests.

Another source of data is provided by the performance metrics as perceived by the client proxy tasked with the job of forwarding user requests to the available PaaS proxies. This intermediary component collects and computes metrics related to every call executed over the NoMISHAP middleware. The gathered data are then periodically sent for digestion to the data aggregation service.

Finally, another piece of the puzzle in the data gathering effort is on the client side. Collecting metrics on the client side enables us to estimate the perceived end-to-end QoS and can help detect incidents and/or faults. This end-to-end measurements can be compared with the prior ones, denoting different logical segments of the communication, and could help diagnose performance issues.

One should not neglect health data related to the actual functional components of the architecture. Depending on the features provisioned by the PaaS vendor, different solutions are viable for extracting health information concerning the proxy
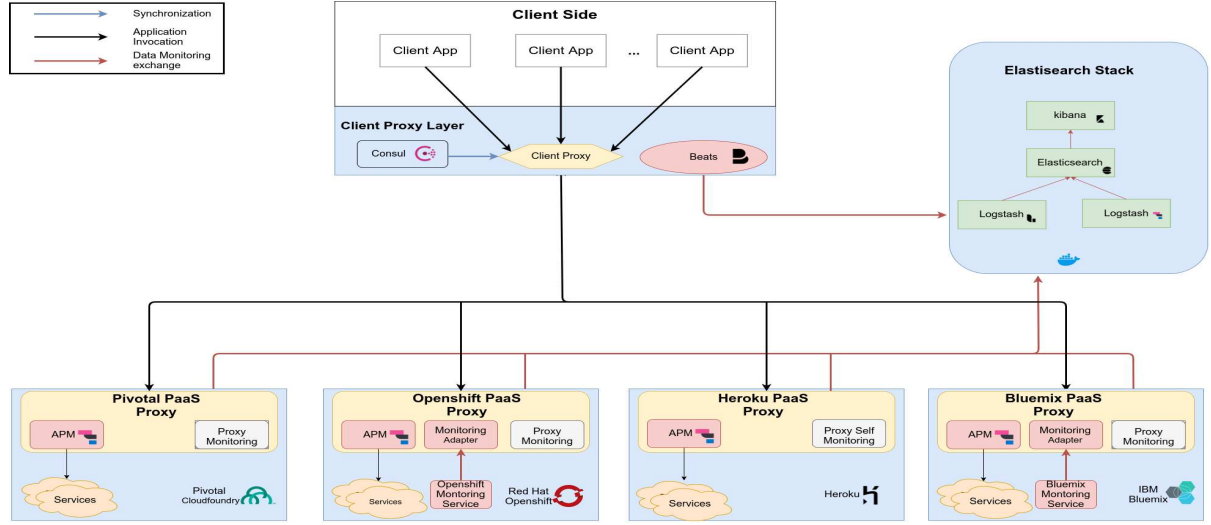
Fig. 2. The proposed monitoring solution based on the NoMISHAP middleware. From the bottom up, one finds the PaaS proxy component tailored to the specific PaaS environment. The component is extended to contemplate for monitoring functionalities pertaining to the PaaS component(s) being used by the application and the proxy itself (gray). Moving up, one finds the client-proxy component, mediating user requests toward the NoMISHAP middleware. Aside, is the logically centralized data aggregation service consisting of a (distributed) Elasticsearch service instance.

itself. In scenarios where the PaaS solution exposes health data related to the services, our monitoring engine can tap to the PaaS monitoring service directly. Otherwise, the proxy can itself collect metrics about state and performance while handling requests, measuring their response times.

B. A Component-based Approach

To collect the data at the PaaS level, we rely on an adapter software component (Figure. 2) which interfaces with the BaaS service layer and PaaS, capable of acquiring the metrics from provider monitoring services, successively forwarding them to the data aggregation service. The data, if needed, might be subject to preliminary computation and/or filtering.

To collect data pertaining to the proxy itself, the component was extended to contemplate a data gathering module. This additional module could also be distributed elsewhere, acting as a standalone component. To this end, we exploit the Elasticsearch APM module which have the capability of intercepting REST calls, steered through the proxy and executed against BaaS services, producing advanced statistics like errors and response time. All the data gathered are successively sent to an APM server which is responsible for data aggregation and forwarding of the latter to a logically centralized aggregation service.

The available PaaS environments comprising the multi-cloud solution, notify their presence and register with the client proxy, acting as an intermediary dispatching user requests. This layer needs an up-to-date view on the underlying resources, that is the available PaaS environments. To fulfill this job it relies on a synchronization service embodied by *Consul.io*, a service networking solution to connect and secure services across any runtime platform. The component acts as an entry point for nearly all operations, and service perfor-

mance is critical for the overall throughput and health of the multi-cloud environment.

To fetch advanced metrics on nodes hosting the synchronization service, we exploit the Beats component present in Elasticsearch. Beat services, are additional modules that can be installed on nodes, allowing to intercept and expose advanced and complete metrics about the state of a hosting node and eventual services installed. The extracted metrics are then reliably sent to the aggregation point.

The proposal is not complete without provisioning an analytics engine capable of visualizing and/or reacting to changes in the environment. Moreover, it is desirable that this engine itself could scale on a per-need basis. To this aim, this functionality resides and is embedded on the Elasticsearch analytics. In the proposal, the later component constitutes the final endpoint responsible for gathering, computation and storage of the metrics of interest, also enabling the visualization, reporting in a near real time fashion. Thanks to the adoption of sharding techniques, Elasticsearch is able to scale, through a policy-based engine, in presence of high computational burden, replicating functional components even cross-cluster. At the end of the provisioned pipeline stands Kibana, a tool capable of aggregating and visualizing data with advanced infographics. All these services, part of the Elasticsearch stack, are capable also of monitor the hosting infrastructure, providing advanced information and alerts on the status of the clusters.

As a last remark, when dealing with a large amount of data, it might occur that direct forwarding to the Elasticsearch service might not be feasible. To alleviate the data burden, we can act on them by applying transformations and/or filtering policies. This duty is tasked to the Elasticsearch Logstash service component which can perform stateless computations

	#Nodes	Provider	Virtualization Technology	RAM/Host (GB)	#CPU/Host
Locust	3	Garr Cloud	Virtual Machine	4	2
Consul	5	2 Azure Public cloud 3 Private Hosting	Virtual Machine	8	2
Elastic Stack	2	Private Hosting	Docker Container	16	4
Pivotal	2	Pivotal Web Services	Cloud Foundry Container	0.5	1
OpenFaaS on OpenShift	2	Private Hosting	Kubernetes Container	1	2
Bluemix	2	IBM Bluemix	Cloud Foundry Container	0.25	1
Heroku	1	Heroku	Dynos	0.5	1

TABLE I
SYSTEM CHARACTERISTICS FOR EACH CLOUD PROVIDER SOLUTION EMULATED WITHIN OUR TESTBED.

to the ingress streams. This allows to scale and replicate these service instances independently on a per-need basis. Moreover, Logstash has the ability to retain metrics and send only when the analytics service is not overloaded contributing to the scalability of the monitoring infrastructure when facing load peaks.

IV. EXPERIMENTAL RESULTS

In the following, we discuss the multi-cloud testbed used to assess our proposal. The monitoring extensions source code along with the testbed configuration can be found in [16].

A. Testbed and Configuration

We set up a test infrastructure emulating a real scenario as shown in Figure. 2. For the purpose of this experimentation, we consider a PDF conversion service provisioned in each of the considered cloud environments. Concerning the multi-cloud PaaS environment, we chose to rely on three top solutions available in the market: (i) Heroku [17], (ii) Cloud Foundry [18], and (iii) OpenShift [19]. The characteristics of the respective testbeds are summarized in Table I.

To simulate a group of concurrent users on the platform, we exploit Locust; a distributed tool used to execute infrastructure load tests. To this aim, we set up three Locust nodes and locally installed our client proxy used to transparently mediate user requests against the NoMISHAP middleware.

The test lasted 12 hours reaching a peak of 5000 requests/s. To assess the soundness of our proposal, we simulate a fault-like behaviour in one subsystem, corresponding to a network congestion event on the client network whose link capacity suddenly drops from 1 Gbps to 200 Kbps. The aim is to asses our proposals both in terms of being capable of identifying the change in behaviour and in pinpointing the source of this *anomaly*. We expect that the metrics gathered from all the subsystems coupled with the knowledge of the relationship between the components should enable a fast discovery of the cause.

B. Performance Assessment

Let us now delve into the behaviour of the monitoring system and check whether the proposed extension fulfills the design objective. Figure 3 shows the latency (ms) as perceived by the client, issuing service requests. In this configuration, the data pertaining to this metric are gathered as soon as they are produced. A spike can be observed around 16:10, the time of the expected anomaly.
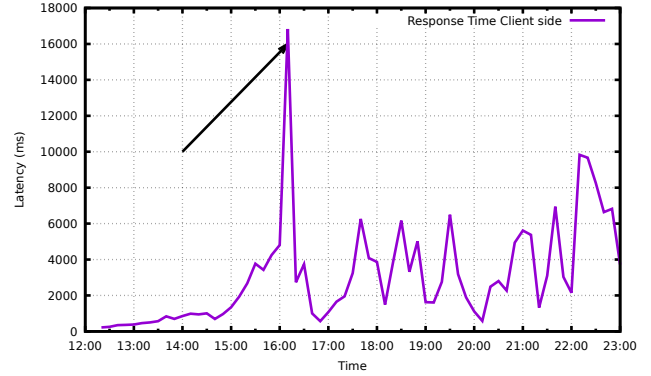

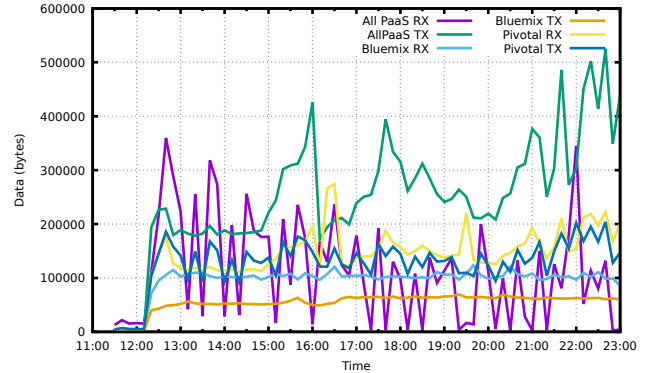Fig. 3. Average response time evolution measured at a Locust endpoint.

Fig. 4. Average response time of BaaS services called by each PaaS.

In scenarios where acquiring the client perceived data is not a feasible option, one can still rely on the data provided by the other components. Indeed, to investigate the cause of this increase in latency, one can follow the chain of invocations through the different components, dark arrows in Figure. 2, and check whether the monitoring system has registered any changes in this particular moment in time.

The first component in the chain is comprised by the BaaS services provisioned in the different PaaS environments. This layer can contribute to a substantial increase in the total response time in scenarios when one or more services go down and also depending on the service resource quota allocation and sustained load.

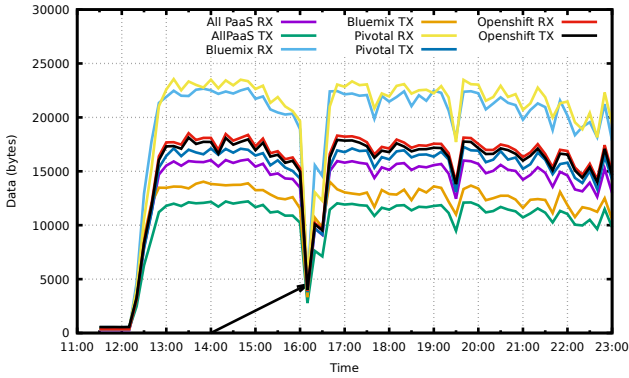Thanks to the monitoring stack added to the PaaS proxy,

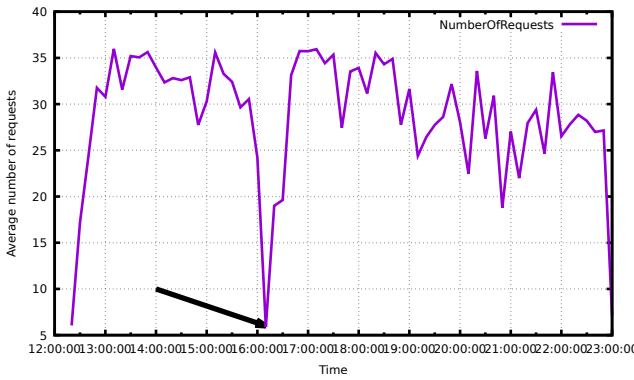Fig. 5. Average network traffic registered by PaaS proxies.

Fig. 6. Average number of requests per second sustained by the client proxy.

we are able to monitor even the services not under our direct control and Figure. 4 shows the individual BaaS service load evolution in time. The client proxy forwards user requests in a round robin fashion. Overall, every single BaaS service contemplated in our assessment is present and exhibits periodic and justifiable behaviour with periodic fluctuations in terms of received and transmitted data.

Moving upward in the invocation chain, is the PaaS proxy itself. Zooming in to some of the statistics gathered from this component, no anomaly emerged except from a lower the number of requests received around the anomaly time (Figure. 5). What this result shows, is that the PaaS proxy itself was not the cause of the increase in the response time as noticed by clients.

There is an additional and last component in our architecture capable of providing further insights into diagnosing the problem. This component is the client proxy whose statistics cover the communication path with the actual user client. At the same time it can sense when a service offered by a PaaS provider is unavailable or even when is the PaaS proxy is unreachable. In fact, it could happen that the PaaS proxy is not able to communicate e.g., a fault state, and the above metrics are our only chance to detect this scenario occurring. Figure 6 shows the number of requests issued against the client proxy, showing a reduction in their number at around 16:10, the time

frame under scrutiny. From this basis, we can deduce that the problem, sudden reduction in sustained load, was on the client side, hence outside the control of the multi-cloud environment.

V. CONCLUSION

We presented a monitoring solution for use in distributed, multi-cloud environments. To asses and validate the design rationale, we presented an experimental analysis involving a realistic testbed consisting of three heterogeneous PaaS environments. As a future work, we plan to built upon the solution and extend its architectural principles to those of edge computing. In this context, an interesting research question is that of extending the proposal to contemplate for a Function as a Service (FaaS) layer.

REFERENCES

[1] Mell, P. and Grance, T. The NIST definition of cloud computing, NIST Spec. Publ. 800 7, 2011.
[2] Kolb S. and Röck C. Unified Cloud Application Management. In Proc. of IEEE SERVICES, San Francisco, CA, 2016.
[3] Zhang Q. *et al*. Dynamic Service Placement in Geographically Distributed Clouds. IEEE JSAC, vol. 31, no. 12, pp. 762–772, 2013.
[4] Amazon AWS [Online]. Patterns for High Availability. Availalbe at: http://en.clouddesignpattern.org/index.php/Main_Page#Patterns_for_High_Availability. Last accessed: March 2020.
[5] IBM Bluemix [Online]. Provide High Availability and Disaster Recovery. Available at: http://www.ibm.com/developerworks/cloud/library/cl-high-availability-and-disaster-recovery-in-bluemix-trs/index.html. Last accessed: March 2020.
[6] Petcu, D. Multi-Cloud: Expectations and Current Approaches. In Proc. of the International Workshop on Multi-cloud Applications and Federated Clouds, New York, NY, USA, 2013.
[7] Ghazi, R. *et al*. Cloud monitoring: A review, taxonomy, and open research issues. Journal of Network and Computer Applications, vol. 98, 11-26, 2017.
[8] Acquaviva, L., Bellavista, P., Bosi, F., Corradi, A., Foschini, L., Monti, S.and Sabbioni, A. NoMISHAP: A Novel Middleware Support for High Availability in Multicloud PaaS. IEEE Cloud Computing, vol. 4, no. 4, 2017.
[9] Nabi, M., Toeroe, M. and Khendek, F. Availability in the cloud: State of the art. Journal of Network and Computer Applications, vol. 60, pp. 54–67, 2016.
[10] Mekikis, P-V. *et al*. NFV-Enabled Experimental Platform for 5G Tactile Internet Support in Industrial Environments. IEEE Transactions on Industrial Informatics, vol. 6, no 3, 1895-1903, 2020.
[11] Milojičić, D., Llorente, I. M. and Montero, R. S. OpenNebula: A Cloud Management Tool. IEEE Internet Computing, vol. 15, no. 2, pp. 11–14, 2011.
[12] Paraiso, F., Haderer, N., Merle, P., Rouvoy, R. and Seinturier, L. A Federated Multi-cloud PaaS Infrastructure. In Proc. of the IEEE CLOUD, 2012.
[13] Paraiso Fawaz and Merle, P. soCloud: a service-oriented component-based PaaS for managing portability, provisioning, elasticity, and high availability across multiple clouds. Computing, vol. 98, no. 5, pp. 539–565, 2016.
[14] F. Dandria, S. B. Cloud4SOA: Multi-cloud Application Management Across PaaS Offerings. In Proc. of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, 2012.
[15] Zeginis, D. *et al*. A user-centric multi-PaaS application management solution for hybrid multi-Cloud scenarios. Scalable Computing: Practice and Experience, vol. 14, 2013.
[16] NoMISHAP Repository [Online]. Available at: https://github.com/NoMishap.
[17] Heroku [Online]. Available at: https://www.heroku.com. Last accessed: March 2020.
[18] CloudFundry [Online]. Available at: https://www.cloudfoundry.org. Last accessed: March 2020.
[19] Openshift [Online]. Available at: https://www.openshift.com/. Last accessed: March 2020.