ChaosTwin: A Chaos Engineering and Digital Twin Approach for the Design of Resilient IT Services

Filippo Poltronieri*, Mauro Tortonesi*, Cesare Stefanelli*,
* Distributed Systems Research Group, University of Ferrara, Ferrara, Italy
{filippo.poltronieri,mauro.tortonesi,cesare.stefanelli}@unife.it

***Abstract*—Chaos Engineering represents an interesting software engineering methodology to improve the resilience of a complex IT system operating in a live production environment by injecting simulated faults, observing the system reaction, and devising mitigating solutions. However, Chaos Engineering is an expensive practice with a high setup and operation overhead and it often focuses on the evaluation of the system behavior from a relatively narrow technical perspective instead of a more comprehensive business level one. To enlarge the audience of Chaos Engineering there is the need for novel solutions that can give service providers the tools to deal with the deployment and testing of complex IT services. To fill this gap, this paper presents ChaosTwin, a novel solution exploring an innovative approach to apply Chaos Engineering to a digital-twin, i.e., a virtual representation of a physical object or a system. By creating realistic digital twin of an IT service, injecting faults on the digital twin and evaluating how different service configuration and fault management strategies would perform from a business level perspective, ChaosTwin provides useful guidance to service providers in finding cost-effective service configurations that can minimize the negative effects of unpredictable events. Experimental results, collected from the evaluation of a realistic case study, demonstrate how ChaosTwin is capable of minimizing both the associated costs and the effects of injected Chaos faults.**

***Index Terms*—Chaos Engineering, Digital Twin, Business Driven IT Management (BDIM), Cloud Computing, Optimization.**

I. Introduction

Software and hardware faults can provoke severe harm to the availability of complex IT services, which rely on the orchestration of multiple independent microservices on large-scale scenarios on top of virtualized resources (VMs/containers, storage, NVFs, etc.) instantiated on a set of data centers around the globe. As a result, both academia and industry have dedicated a significant effort to investigate how to properly manage and minimize (or mitigate) their impact.

Chaos Engineering is a relatively new technique that was proposed to test the resiliency to faults of software in production (real-world deployment). Chaos Engineering was first formalized and proposed by Netflix Inc. with Symian Army project [1], a is a suite of tools developed with the goal of improving the reliability of the company cloud services. The main idea behind Chaos Engineering is to identify a steady-state, i.e. the expected behavior, of a running system and then trying to inject faults to stress the system and verify the results. Example of injected fault can be increased latency, VM breakdown, and data-centers outages. This is to increase the testing methodologies for software developers / cloud/ infrastructure managers, thus testing real-world event on a running system. The main objective of Chaos Engineering is to identify if a system is resilient to adverse events and to plan modification to make it more reliable to this sort of faults.

The software engineering approach followed by Chaos Engineering presents the undeniable advantage of evaluating software systems in their entirety - including hardware breakdowns and software bugs - and in their intended deployment environment. In fact, given the high promises of this technique, Cloud providers such as Amazon and Microsoft have started to offer chaos injection simulators to verify the reliability of distributed and complex software tools. However, Chaos Engineering also suffers from several drawbacks. Namely, the injection of faults on live production systems is an expensive practice, which requires a relatively high amount of resources and time to be properly set up. In addition, Chaos Engineering practices typically focus on IT level evaluation and only recently have started realizing that fault mitigation is essentially a business level concept, and as such it requires to consider business value as a fundamental metric in fault mitigation policy adoption [2].

It would be interesting to extend the software engineering approach pursued by the Chaos Engineering discipline by also considering an alternative approach at the design level. More specifically, we argue that *a digital twin approach*, based on the accurate reenactment of the IT system and on the evaluation of the impact and mitigation strategies of faults injected on that model, *could bring many of the advantages of the Chaos Engineering approach with a much smaller price tag and enabling much faster feedback cycles at the design level*.

This represents a relatively unexplored research avenue, which requires new tools capable of exploring alternative IT service architectures by applying the principles of Chaos Engineering and to evaluate their performance through a comprehensive analysis *at the business level*, with the purpose of identifying a trade-off between convenience and fault resiliency. Incorporating Chaos Engineering practices into “configuration exploration” would enable to address risk management and to forge more resilient IT service deployments that can minimize the effects of faults / outages events.

This paper introduces ChaosTwin, a novel tool based on the experience built upon the BDMaaS+ research project [3], [4] which pursues an innovative approach that integrates Chaos

Engineering techniques into the configuration of modern IT services operating in the hybrid Cloud. More specifically, this paper embraces Chaos Engineering principles not for testing software resiliency itself, i.e bug fixing, but as optimization approach for maximizing the whole system reliability. The main objective of this paper is to propose a valuable tool for assisting service providers that need to distribute an IT service on a global scale. The presented methodology represents a novel idea that takes into consideration both a cost and a resiliency perspective to find suitable and minimal costs IT service configurations.

II. Related Work

The principles of Chaos Engineering were published in [5] as a manifesto for the developers that want to embrace this technique. One of the key tool developed by Netflix Inc. is Chaos Monkey [6]. Chaos Monkey was developed to randomly terminates virtual machine and containers running in a production environment. Netflix claims that exposing services to failures can incentivize software engineers to improve their resiliency. Guided by this interest in Chaos Engineering other software solutions have emerged. Among these efforts, Amazon Inc. has started to provide a Chaos Engineering service called AWS Fault Injection Simulator.

If Chaos Engineering is becoming a well accepted practice in software production, it is not yet well explored in scientific literature. Basiri et al. summarize the practices of Chaos Engineering in [7]. More specifically, the authors start by analyzing the Netflix's experience and then formalize a system model for Chaos Engineering. In [8] the authors provide a simplified illustration of Chaos Engineering in a simulated testbed within the NS-3 simulator. The testbed consists of a small-case simulated Netflix environment on which is possible to inject faults and verify the outcomes. Differently in [9], Torkura et al. analyze Chaos Engineering from a security perspective, focusing on the effects caused by malicious events, i.e. cyber attacks, and misconfiguration. Then, the authors extend their vision in [10], in which they propose an auditing and monitoring tool for Cloud infrastructures.

On the other hand, some works focus on fault injection as valuable technique to stress the capabilities of a software infrastructure. In [11], the authors propose GRINDER, an open source toolkit to ease the burden of the adoption of fault injection techniques in software testing. In [12], the authors introduce the interesting concept of survivability for service function chaining in case of critical service disruption. Cotroneo et al. present a dependability benchmark for NFV that integrates fault injection techniques in [13], which also compares the performances of hypervisor-based and container-based virtualization paradigms. Then, Simonsson et. all analyze the problem of system call errors in containerized applications by presenting a novel fault injection framework called ChaosOrca in [14].

This paper adopts an higher level perspective and embraces the principle of Chaos Engineering for the development of a management framework that enables service providers to explore the best configurations for IT services that minimize the effect of adverse faults. This work is different from both the works that apply chaos engineering to test software resiliency [7]–[10] and also from the works that analyze system and platform level resiliency [11], [13], [14]. Differently from these works, we integrate Chaos Engineering and Digital Twin to evaluate the resiliency of IT service configuration from a higher and business-level perspective.

III. ChaosTwin

ChaosTwin is an extension of the BDMaaS+ research project, a simulation-based optimization framework providing a comprehensive Business-Driven (BD) evaluator for IT service configurations of software components operating in the hybrid Cloud [4]. ChaosTwin leverages the BDMaaS+ capabilities to create a digital twin version of a Cloud-based IT system, evaluate its business-level performance, and find the optimal configuration for that system, extending those capabilities to explicitly consider fault injection and management strategies.

The overall architecture of ChaosTwin is illustrated in Fig. 1, which shows the main components and their interactions. Among them, the ChaosTwin Engine represents the central component of the tool, containing several subcomponents that implement its functions: reenacting the digital twin version of the IT system in configuration x (Digital Twin Reenactment), implementing fault mitigation strategies for configuration x (Fault Manager), evaluating the business impact of the digital twin in configuration x (Business Impact Analysis), finding the configuration with the best business-level performance (Optimization), and deciding whether the performance gains are worth to switch from the current configuration x_0 (Decision Making).

ChaosTwin was designed to maximize the accuracy of the digital twin. To this end, it leverages sophisticated functions that build models of the digital twin at the workload, network, and service level. To facilitate the definition of the model, ChaosTwin can automatically build a model of a real system by interfacing with a Monitoring Agent that feeds it information about the real system behavior as illustrated in Fig. 1. ChaosTwin can also integrate smoothlessly with a real system, operating as a (sophisticated) controller. More specifically, the output of the Decision Making component can be fed to an Actuation Agent that interacts with an Orchestrator component to reify the best performing configuration identified by the tool on a real system. Finally, all the functions of ChaosTwin can be accessed from a dedicated Console component.

A. Business-Level Evaluation

From a mathematical formulation perspective, ChaosTwin attempts to solve the following optimization problem:

$$\underset{x \in \mathbb{S}}{\arg\min} \quad BI(x) \tag{1}$$

where x represents the IT service configuration, including deployment configuration and fault mitigation strategies, $BI(x)$

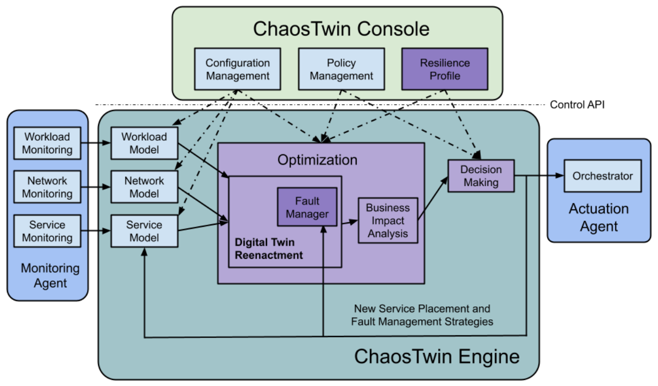

Fig. 1. Architecture of the ChaosTwin tool.

the Business Impact for configuration x, and $\mathbb{S}$ is the space of all possible configurations. In fact, ChaosTwin allows users to seamlessly add what-if scenarios in the evaluation in accordance with chaos engineering techniques by enlarging the perimeter of $\mathbb{S}$.

ChaosTwin implements $BI(x)$ as an extension of the BDMaaS+ BD evaluation of a given configuration x by also considering, for a given configuration x, fault mitigation costs in addition to the 3 previously considered components: IT spending cost evaluation, SLA violation penalties estimation, and business (mis)alignment penalties [3]. Formally, it is:

$$BI(x) := IC(x) + SLO(x) + BAP(x) + \theta \times FF(x). \quad (2)$$

where $IC(x)$ calculates the operational costs caused by running the system with configuration x, $SLO(x)$ that calculates the costs caused by SLO violation penalties, $BAP(x)$ considers all costs caused by the adoption of a service configuration which is not aligned to the business objectives, and $FF(x)$ the number of failed requests for configuration x.

As the main indicator to measure the effects of an injected fault, we select the number of failed requests $FF(x)$. In fact, this metric was specifically developed as part of the BDMaaS+ extension to support Chaos Engineering practices. By leveraging on the failed request metric, the optimizer can find a different configuration for software components to allocate both in terms of the number of VMs to allocate and in terms of the geographical location of VMs. For example, in the case of an adverse event causing an outage in a data center, a possibility is to redirect part of the requests to the closest data center or to select another data center that is running the particular software component. These mechanisms should be supported and implemented at the platform level, which should also estimate the redirection costs, such as the increased latency due to the redirection. In fact, $FF(x)$ is the number of failed requests when the system is in configuration x and θ is a weight factor for balancing the optimization of both components of (2).

IV. Fault Management in the Digital Twin

To fully apply Chaos Engineering techniques and evaluate the resilience of IT service configurations, ChaosTwin simulates chaos engineering events on a digital twin that reproduces a real-life IT service operating in a federated Cloud. ChaosTwin aims to include fault-resiliency into the optimization process to minimize both associated IT costs and adverse effects of outage events. By doing this, ChaosTwin can drive the optimization process to service configurations that can better tolerate: i) partial or full outages in data centers ii) VMs malfunctioning, iii) increased inter-data center communication latency.

Towards this goal, we designed different events for simulating chaos practices on the Digital Twin: *VM Outage* (VMO), *DC Outage* (DCO), and *Latency Variation* (LV). These are representative Chaos-like faults that we implemented within ChaosTwin to enable their reproducibility on the digital twin. VMO events are to simulate errors/faults in VMs installed in Cloud data centers that make them unavailable for a configurable period of time $vmu(t)$. ChaosTwin leverages random variables with configurable distributions for modeling the rate of VMO events during the simulation. When a VMO event occurs, ChaosTwin marks the VM as *not running* and then reboots the VM. It is worth noting that while the VM is unavailable it would not serve requests, which will be served by another VM (if available).

On the other hand, DCO intends to simulate the outage of an entire data center that would become unavailable for a configurable period of time $dcu(t)$. When a DCO is injected into the simulation, ChaosTwin marks the data center as "unavailable" for the time being and thus making it impossible to process requests on that data center. Let us specify that DCO is to simulate severe adverse events that are less likely to happen in real-life deployment. However, the recent case of the fire in a Belgium data center that destroyed several machines caused several applications to stop working for several days [15].

Then, leveraging LV, ChaosTwin can inject increased latency on a communication endpoint, which is identified by two locations, e.g two data centers. For example, ChaosTwin users can choose to simulate an increased latency between two data centers located in the same or different regions to verify the negative effects on the IT service configuration, the need to replicate software components, and so on.

Finally, along with the modeling of specific chaos engineering events, it is crucial to define new metrics for measuring the response of the service. To this end, ChaosTwin implements the 'failed requests" metric for estimating the negative effects of an injected chaos fault. Within the ChaosTwin framework, a failed request indicates that a request was not served because, at a given time, there was no active VM capable of serving the given request. We believe that minimizing the number of failed requests will in turn improve the resiliency of an IT service configuration in dealing with adverse and unexpected events.

TABLE I
THE SERVICE TIMES FOR EXECUTING THE SOFTWARE COMPONENTS ON DIFFERENT VM TYPES

Component	medium VM	large VM	xlarge VM	xxlarge VM
VS	45ms	30ms	15ms	10ms
AS	25ms	20ms	15ms	10ms
SS	20ms	15ms	12ms	8ms

TABLE II
THE SLAS FOR THE VIDEO STREAMING SERVICE

Workflow	Locations	SLA
VS	East and West Coast USA, Europe	400-650 ms
VS	South-America, Japan	500-750 ms
AS	East and West Coast USA, Europe	200-300 5ms
AS	South-America, Japan	250-350 ms

V. USE-CASE SCENARIO

To illustrate the proposed approach, we devised a realistic case study built upon the experience of previous works [4], [16]. To follow the example of Netflix, which first proposed the adoption of Chaos Engineering, we decided to model a use case study that represents a video streaming service operating on a global scale. We believe this to be an interesting case study for testing Chaos Engineering practices as it maps the scenario of a service provider offering an IT Service to multiple locations around the globe.

A. Service Description

We envision a realistic use case that simulates the delivery of multimedia contents to a global scale audience. More specifically, the use case represents a digital twin of a canonical video streaming service, which requires at the client-side the users to download both a video and a audio track, and optionally a subtitle file for the video. This scenario consists of 3 software components, which we indicate with *VS*, *AS*, *SS*. VS is a Web Server that distributes video contents, e.g. video files that do not contain audio tracks or subtitles. Then, AS is a Web Server that distributes the audio tracks for the video files. A video file can be associated with multiple audio tracks, one for each language as users can select their preferences. Finally, SS is a Web Server that distributes subtitle files for the video tracks. For each software component we modeled an according response time that depends on the type of VM on which it is running as illustrated in Table I.

Along with the software components, we defined two service workflows: *WF-01* and *WF-02*). *WF-01* represents the workflow to retrieve a particular video content, thus resulting in a three components workflow (VS, AS, SS). To reduce the number of workflows we consider that for each video request there must be a subtitle track request, even if the user does not request it. Instead, *WF-02* is to simulate a request for changing the audio track and/or the subtitle track for a particular video that he is watching. This workflow involves the AS and the SS services from which the user requests the proper audio and/or subtitle tracks.

As for the data-centers, we consider a federation of 8 different Cloud facility locations. More specifically, we consider 6 public Cloud data centers, namely Amazon EC2's *us-west-1*, *us-east-1*, *eu-west-1*, *ap-southeast-2*, *sa-east-1*, and *ap-southeast-1* data centers, and 2 private Cloud data centers, respectively located in northwestern USA and in Japan.

Then, we modeled users' requests according to a uniform distribution with probability 0.8 for *WF-01* and 0.2 for *WF-02*. In addition, these users are distributed in the following locations: East Coast USA, West Coast USA, South America, Asia, and Europe. The divisions are of varying sizes and account for a different share of requests: 11.1%, 16.6%, 27.8%, 33.3% and 11.1% respectively. Furthermore, the aggregated flow of requests has a constant intensity - and whose interarrival times can be modeled with a Pareto distribution with location 1.2E-4 and shape 5, corresponding to 6,666.66 requests per second. The requests emanating from each division will be automatically forwarded to the closest Cloud data center, as a common practice for the Route 53 system.

Finally, to allow a user to correctly reproduce a multimedia content all three software components must provide a response within the time window defined by SLAs. We summarized the SLAs for the three software components at the respective service delivery locations in Table II. Let us specify that, if one of the workflow component fails, the whole workflow is invalidated.

VI. EXPERIMENTAL EVALUATION

To experiment with the Chaos Engineering tools of our framework, we devise a Chaos Engineering experiment upon the Video Streaming Service described in Section V. The main objective of this experiment is to verify the output of injected Chaos tools into a simulated IT service and verify if allocation policies can minimize the outcomes of these faults. Furthermore, with this preliminary experiment, we want to formalize a Chaos engineering scenario that we will use as a baseline for future works.

To implement this experiment, we describe a severe fault profile that would affect the video streaming application illustrated in the previous section. More specifically, this experiment schedules VMOs for VMs running within the Amazon EC2 data center **ir** using a random variable with exponential distribution with a rate of $\lambda = 0.8$. To increase the randomness of the simulation we chose to terminate VM randomly without specifying the software component type. When a VMO occurs, the VM is marked as unavailable for 2 seconds, which is the time we configured to reenact a VM reboot. Along with VMOs, we simulate the complete blackout of two data centers for 15 seconds. The first one is a private data center, while the second one is the Amazon EC2 data center located in California, USA.

We configured ChaosTwin to run this experiment for 30 iterations of the memetic optimization algorithm to analyze the resulting IT service configurations. More specifically, the memetic algorithm combines a tailored version of Quantum Particle Swarm Optimization (QPSO) and a local search procedure (interested readers can refer to [4]) to find an IT service configuration that minimizes the value of (2). Let us specify that the optimization policy (2) aims at decreasing

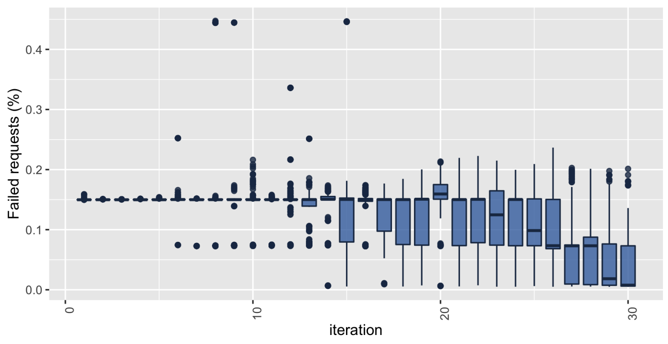

Fig. 2. The distribution of the percentage of failed requests during the optimization process for the severe outage experiment.

operation costs (VMs renting prices), SLO violation penalties, and faults associated with a VMs placement. Therefore, at the beginning of the optimization process configuration solutions would present a larger number of instantiated VMs, which will be minimized during the iterations of the memetic algorithm. It is needless to say that configurations with a large number of VMs instances would likely be more resilient to injected faults, i.e., replicas mitigate per se the side effects of sporadic VMOs within the same data center and DCOs at the cost of increased serving latency (requests must be routed to a different data center). Consequently, it is important to verify if ChaosTwin can minimize the side effects of injected faults when the number of instantiated VMs start to decrease.

To this end, Fig. 2 shows the distribution of the percentage of failed requests during the 30 iterations of the optimization algorithm. Let us note how after the 15-th iteration, the distribution of failed requests starts to decrease, thus indicating the optimization policy can find service configurations presenting increased reliability to the injected chaos faults. These results prove that ChaosTwin implements a valuable approach in minimizing the number of failed requests, which are almost distributed around 0.05% in the final stages of the optimization process. Furthermore, the ChaosTwin optimization policy also minimizes the associated operational costs, which in the very first iterations are distributed around 550,000 USD/day to reach about 30,000 USD/day in the final iterations. This also demonstrates that operational costs minimization policy can be associated with fault mitigation strategies with promising results. Finally, even if these results represent a preliminary stage of our research effort, their soundness calls for further investigation and an in-depth validation of ChaosTwin.

VII. Conclusions and Future Works

Chaos Engineering was first introduced by Netflix Inc. in 2010 as a novel technique to test the reliability of software running in a Cloud-based production environment. Later, this practice started to attract the interest of a wider community of scholars, software developers, and Cloud providers such as Amazon Inc. and Microsoft Inc. that are now beginning to provide chaos engineering services for their customers. However, the majority of these services were designed to find software vulnerability and not for exploring the adverse effects of faults on Cloud-based IT systems.

To fill the gap from a system-wide perspective, this paper presented a first attempt for including Chaos Engineering techniques into VMs placement / service configuration techniques for IT services running in the hybrid Cloud. The novel idea we presented in this paper is to provide a management framework capable of exploring the cost-efficient deployment of global-scale IT service with an a-priori resilience to fault events. We proposed *ChaosTwin* a comprehensive framework to help service providers in selecting resilient and fault-prone configurations for their IT services. Finally, future research directions will entail a more comprehensive validations of our model and the adoption of different optimization policies.

References

[1] "The netflix simian army," https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116, accessed: 2021-04-26.

[2] M. Korolov, "Chaos Engineering Moves Beyond 'Breaking Stuff' to Highlight Business Value," https://thenewstack.io/chaos-engineering-business-value/, accessed: 2021-06-11.

[3] M. Tortonesi and L. Foschini, "Business-driven service placement for highly dynamic and distributed cloud systems," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 977–990, 2018.

[4] W. Cerroni, L. Foschini, G. Y. Grabarnik, F. Poltronieri, L. Shwartz, C. Stefanelli, and M. Tortonesi, "BDMaaS+: Business-driven and Simulation-based Optimization of IT Services in the Hybrid Cloud," *IEEE Transactions on Network and Service Management*, p. to appear, 2021.

[5] "Principles of chaos engineering," https://principlesofchaos.org/, accessed: 2021-04-26.

[6] "Netflix, chaos monkey tool," https://github.com/netflix/chaosmonkey, accessed: 2021-04-26.

[7] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.

[8] T. Luong and A. Zubayer, "Simulation of chaos engineering for Internet-scale software with ns-3," 2018.

[9] K. A. Torkura, M. I. H. Sukmana, F. Cheng, and C. Meinel, "Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure," *IEEE Access*, vol. 8, pp. 123 044–123 060, 2020.

[10] K. Torkura, M. Sukmana, F. Cheng, and C. Meinel, "Continuous auditing and threat detection in multi-cloud infrastructure," *Computers and Security*, vol. 102, 2021.

[11] S. Winter, T. Piper, O. Schwahn, R. Natella, N. Suri, and D. Cotroneo, "Grinder: On reusability of fault injection tools," in *Proceedings of the 10th International Workshop on Automation of Software Test*, ser. AST '15. IEEE Press, 2015, p. 75–79.

[12] G. Kibalya, J. Serrat, J.-L. Gorricho, J. Serugunda, and P. Zhang, "A multi-stage graph based algorithm for survivable service function chain orchestration with backup resource sharing," *Computer Communications*, vol. 174, pp. 42–60, 2021.

[13] D. Cotroneo, L. De Simone, and R. Natella, "Nfv-bench: A dependability benchmark for network function virtualization systems," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 934–948, Dec 2017.

[14] J. Simonsson, L. Zhang, B. Morin, B. Baudry, and M. Monperrus, "Observability and chaos engineering on system calls for containerized applications in docker," *Future Generation Computer Systems*, vol. 122, pp. 117–129, 2021.

[15] "OVH Strasbourg data centre destroyed by fire, customers told to activate DR," https://cloudcomputing-news.net/news/2021/mar/10/ovh-strasbourg-data-centre-destroyed-by-fire-customers-told-to-activate-dr-updates/, accessed: 2021-04-26.

[16] W. Cerroni, L. Foschini, G. Y. Grabarnik, L. Shwartz, and M. Tortonesi, "Service Placement for Hybrid Clouds Environments based on Realistic Network Measurements," in *2018 International Conference on Network and Service Management (CNSM 2018) - miniconference track*, November 2018.