A Techno-Economic Assessment of Microservices

Ioannis Papakonstantinou
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
mai19056@uom.edu.gr

Sarantis Kalafatidis
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
kalafatidis@uom.edu.gr

Lefteris Mamatas
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
emamatas@uom.edu.gr

***Abstract*—The microservices design paradigm enables applications, usually based on containers, exploiting the flexibility of cloud computing and bringing unique scalability, fault-tolerance and resource-allocation benefits. A number of orchestration facilities, including Kubernetes, target the efficient deployment and operation of containers and are mainly focusing on the maintenance of server resource allocation under predefined thresholds, i.e., through scaling up or down containers to mitigate dynamic changes in the workload. In this work, we highlight the technical capabilities and cost-saving impact of microservices in contrast to traditional monolithic applications, based on a techno-economic analysis. We also investigate the service performance vs resource allocation trade-off, uncovering interesting dynamics when elasticity is driven from service quality metrics. This approach allows the Service Providers (SPs) to balance their profit margins with the customer satisfaction, i.e., reducing the infrastructure cost while keeping the service performance at an acceptable level.**

I. Introduction

Cloud computing [1] is offering computing as a utility, matching dynamically the resource supply with the demand, which leads to better services and significant reductions in the infrastructure expenses. Traditional monolithic applications are not suitable for cloud environments, since they are usually resorting to resource over-provisioning, due to their inability to flexibly adapt to dynamic workloads, increasing the cost of infrastructure and the risks of instabilities due to unexpected peaks in the service resource requirements. Along these lines, the microservices architecture (MA) [2] was proposed as *"an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms"*. Such systems consist of small, single-functionality services that can be implemented, scaled and tested independently, especially towards putting together large-scale services handling dynamic workloads.

Although MAs introduce virtualization and communication overhead, their capability to flexibly tailor the service deployment to the instantaneous dynamics of the client resource demands brings significant application performance, fault-tolerance and infrastructure cost benefits. Such services are mainly based on containers and orchestrated by novel environments, e.g., Kubernetes, towards satisfying particular server resource demand goals. However, the latter mainly consider the containers as black boxes, without explicitly addressing crucial Service Provider (SPs) aspects, such as customer satisfaction or monetary cost of the utilized cloud resources.

Since the infrastructure operational cost is significant for a SP, efficient resource management leads to profit increases, depending on the maintenance of customer satisfaction. The targeted client performance is usually related to particular Service Level Objectives (SLOs), as part of Service Level Agreements (SLA) between the Service and the Infrastructure Providers. Consequently, it is important to study the impact of MAs on these both aspects and enable the appropriate tuning of user Quality of Experience (QoE) and infrastructure monetary cost trade-off. The latter should be continuously maintained through appropriate scaling up or down of microservices with respect to the dynamic workload magnitude, i.e., a capability called elasticity. Elasticity should strike a balance between over-provisioning and under-provisioning [3], avoiding unnecessary server resource utilization and reduced QoE, respectively.

In this paper, we highlight the technical and economical benefits of MA, contrasted to the monolithic application design option, based on a techno-economic analysis. Our findings underline both the server resource efficiency benefits of MA and the infrastructure cost reduction, which are both up to 64%. Furthermore, we argue that a tolerable increase in the application Response Time (RT) can further reduce the operational costs of the applications, e.g., by 72% in our case. Our results suggest that relevant elasticity mechanisms based on upper thresholds in customer performance metrics can be an interesting viable option for Service Providers.

The remainder of this paper is organized as follows. Section II contrasts this paper against the related works. Section III provides our techno-economic analysis, comparing the microservices to the monolithic approach, in terms of application performance, resource utilization and infrastructure cost benefits. The same section also investigates the trade-off between service RT vs server resource allocation and cost. Lastly, Section IV discusses our next steps and concludes this paper.

II. Related works

Here, we discuss works investigating the technical and economic benefits of MAs, as well as elasticity approaches that consider microservices and their economic and client performance impact.

A number of papers investigate the technical capabilities of microservices by comparing them with monolithic architectures, i.e., experimenting with two versions of the same application, corresponding to both design options. In [4], [5], the authors experiment with web services developed with both Play framework and AWS Lambda as the microservices technology, highlighting server resource savings but with an increase on client RTs. A similar comparative methodology is adopted by paper [6], but with a social network application and container-based microservices, i.e., achieving high scalability and throughput, but lower RTs, in the case of MA. The work [7] observed a significant resource consumption increase in the MA case, mainly due to the containerization overhead, based on single-host experiments. However, the MA demonstrates its full potential with a large number of users and a distributed deployment of service functions, while exercising its novel elasticity capabilities. For example, in [8] the authors recommend the monolithic architectural option in the case of a small load, i.e., less than 100 users. Finally, in [9] the authors compare traditional VMs with container-based services in terms of scalability, highlighting significant operational efficiency in the case of MA.

The survey paper [3] discusses elasticity handling mechanisms in cloud computing, including approaches for microservices. Furthermore, proposals [10], [11] take into account the infrastructure cost, while also consider service performance, but do not have a techno-economic viewpoint. In particular, paper [10] targets service maintenance in accordance with the SLA, at an economically efficient level in terms of locating low-cost cloud resources. In [11], authors perform workload prediction to drive microservices scaling through implementing vertical elasticity strategies, i.e., dynamically adding or removing physical resources.

Here, we highlight the technical and economical benefits of microservices in contrast to monolithic architectures through a techno-economic analysis which decouples the impact of virtualization overhead. Also, we investigate the service performance vs resource allocation trade-off, in the context of MA. Our research results highlight the efficient resource allocation and cost benefits of microservices that could be further improved with elasticity processes targeting conservative service quality

III. Techno-economic study of Microservices

In this section, we evaluate the technical and economic benefits of microservices technology, also investigating the service RT vs the infrastructure costs trade-off. We present the methodology of our analysis and our techno-economic study built around a web application going viral example. We assume a typical web site that publishes articles that suddenly become viral. The web site owner should keep his site online, while minimizing his web hosting expenses, even by sacrificing web visitors' performance up to a tolerable point. Our goal is to investigate how different service deployment configurations, in terms of service architecture types and elasticity strategies, can handle this challenging situation.

A. Methodology

To assess the impact of MA in terms of resource allocation, service performance and infrastructure expenses, we carried out a comparative analysis of two variations of a web application we implemented, i.e., based on both microservices and monolithic architectures, as shown in fig. 1. The application consists of a number of independent service functions, such as: (i) *Nginx* routing the outside requests to particular service functions; (ii) *Main Page* hosting the main webpage of the web application; (iii) *Articles* publishing articles that become viral, returned to the user after an interaction with a MongoDB database; (iv) *Site Admin* and *Comments* implementing the web site administration and the capability of visitors to leave comments, respectively; and (v) *MongoDB* representing a MongoDB database. On the left side of fig. 1, we illustrate a version of the application aligned to the MA paradigm, i.e., considering all the above functions as microservices, while on the right side the monolithic version. In the latter, all functions belong in the same service architecture, besides the MongoDB, which we do not take into consideration, since there are no architectural differences in its deployment between both service architectural options.

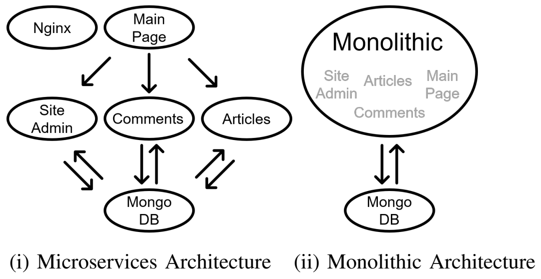

Fig. 1. Alternative architectures of web application

Our analysis is based on the following methodological choices and assumptions: (i) both services are deployed in containers to decouple the impact of microservices technology used, e.g., the Monolithic container hosts all the web application, besides the MongoDB; (ii) we set resource limitations in the containers to emulate a distributed deployment, which are diverse to reflect the different resource-demands of the service functions; (iii) the number of visits to the service is equal to the number of user requests; (iv) the users' satisfaction (i.e., QoE) depends on the average RT; and (v) the infrastructure costs are considered from the CPU utilization view-point. Furthermore, we consider a 3 sec upper limit on the user request RT, i.e., as a Service Level Objective (SLO). According to a survey contacted by Google [12] and illustrated in fig. 2, the user will stop waiting for a response to a request with a 32% probability, when waiting from 1 to 3 sec. Such bounce rate increases with the waiting time. The most significant probability increase is in between 3 and 5 sec, i.e., 58%, which is the reason for our 3 sec choice. Moreover, we assume that all service clients are satisfied when the 95% of the requests receive a response within this timeframe, since most SLAs are considered violated with a lower than 95% success in their SLOs.

We simulate the workload (i.e, the client requests) with the *Apache JMeter* [13] benchmarking / load-testing tool. *JMeter* provided us with the service quality measurements,

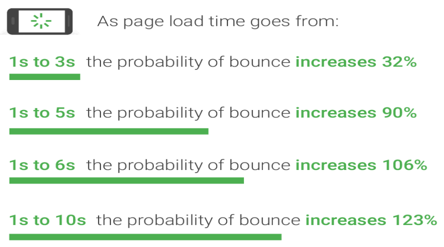

Fig. 2. RTs and bounce rates [12]

i.e., the requests' response-time, which we maintained below the SLO, for at least the 95% of requests. We implemented all microservices as Docker [14] containers, while Docker Swarm orchestrates their deployment, e.g., implementing the container replicas auto-configuration and the communication among them. We used *Docker Stats* to measure the real-time resource allocation, in terms of CPU usage.

The techno-economic analysis that follows compares experimentally the monolithic with the microservices version of the studied web application, assuming realistic patterns of user visits. Additionally, it investigates the potential of alternative application deployments to handle a large number of users with tolerable performance, while reducing significantly the cloud resource expenses.

B. Techno-Economic Analysis

Our analysis consists of two basic steps: (i) a comparative experimental analysis between different service architectures and elasticity strategies, assuming a viral number of users; and (ii) cost calculations producing estimated infrastructure costs for the different application deployment configurations. We consider three different cases of the studied service: (i) a *Monolithic* version that utilizes enough containers to support the given number of users without violating the SLO; (ii) a *CPU-based MA* deployment that scales the microservices based on a 80% CPU threshold, i.e., a typical configuration in Kubernetes; and (iii) an *SLO-based MA* deployment that attempts to maximize cost savings through scaling the microservices according to the chosen 3 sec SLO value. The goal of each application deployment case is to keep the 95% of the RTs less than 3 sec. Practically, we add new containers whenever the latter rule is violated for all cases or a container is utilizing more than the 80% of its CPU in the *CPU-based MA*.

In the first series of our experiments, we emulate 17,500 requests in total by 175 active *Apache JMeter* threads, while 8 CPU cores are available for each experimental run. The duration of experiments is around 15 min. For simplicity, our experiments are particularly focusing on the *Articles* functionality of the application, i.e., delivering viral articles. Consequently, we allocate bare minimum resources to all microservices, except the *Nginx* (i.e., handling all incoming requests), *Articles* and *Monolithic* services.

In table I, we show the CPU Cores required for the 3 application deployments. We see that the *Monolithic* service requires 5 CPU cores, while the *CPU-based MA* 1.80 and the *SLO-based MA* 1.40. This advantage of MAs derives mainly from the Single Responsibility Principle (SRP) characterizing them, since they adapt to the diverse demands among services.

TABLE I
CPU CORES PER SERVICE DEPLOYMENT

Services	Monolithic	CPU-based MA	SLO-based MA
Monolithic	5.00	0.00	0.00
Articles	0.00	0.50	0.10
Site Admin	0.00	0.10	0.10
Comments	0.00	0.10	0.10
Main Page	0.00	0.10	0.10
Nginx	0.00	1.00	1.00
Total	**5.00**	**1.80**	**1.40**

In table II, we present the experimental results for the 3 service deployment cases, i.e., *Monolithic*, *CPU-based MA*, and *SLO-based MA*, in terms of CPU Cores consumed and statistical data related to the measured RT (RT). The RT measurements have been collected by the *JMeter* load testing tool, are in ms and express the average and median RT from all the requests. The 95% Line expresses the time, in ms, which was not exceeded by the 95% of the requests (i.e., the 95% percentile) and the error presents the percentage of failed / non-served requests. In these results, we observe an inverse relation between the CPU Cores consumed and the RT, i.e., a quick response to the requests should be supported by significant processing resources. In addition, the RT remains in all cases below 3 sec and there are no errors. We note that our techno-economic analysis is based on the assumption that the user QoE depends on the average RT.

TABLE II
EXPERIMENTAL RESULTS

Deployment	CPU Cores	Avg.	Med.	95% Line	Er. %
Monolithic	5	76	70	138	0
CPU-based MA	1.8	648	606	1,233	0
SLO-based MA	1.4	1,529	1,575	1,814	0

However, considering this assumption and comparing the first two deployments (i.e., *Monolithic* & *CPU-based MA*), the observed $572ms$ increase in RT is tolerated, since it is around the $\frac{1}{4}$ of the SLO and produces significant cost savings (i.e., a 64% reduction in the CPU usage). We also highlight in the same results the flexibility offered by the MA, since not all microservices consume significant resources. An efficient service orchestrator needs to scale up the *Nginx* and *Articles* microservices only, in this particular example. In *Monolithic* case, the entire application needs to be scaled up, leading to inefficient resource management.

Comparing the third service deployment with the other two, we observe that there is even more space to trade service performance (i.e., RT) for more efficient resource management. In such a case, we are approaching closer to the SLO, but achieve a 74% reduction of CPU usage. Although in demanding real-time applications this strategy may cause instabilities, it highlights the significant cost benefits of a system that implements service-quality driven elasticity, especially with conservative performance goals.

We attempt to extend our findings with a cost analysis based on real cloud prices and user requests values. According to the Amazon Web Services website [15], a typical cloud server with 32GB of RAM, 30GB SSD, 8 CPU Cores, 24/7 peak availability and Linux OS, costs $747 per month. We consider a high workload scenario with over 2 bil. visits per month to resemble viral conditions. For this reason, we assume having similar traffic with Netflix. Netflix received 2 bil. visits in February 2020, according to SimilarWeb [16], in which approximately 4 pages have been browsed per visit. This is equivalent to 2,873,563 requests per 15min. Moreover, we simplify our analysis and consider no additional technical limitations and costs by scaling up our results with the number of visits from Netflix, i.e., multiplying the required amount of resources in our experiment by 165 (i.e., ≅2,873,563/17,500). Table III presents the overall costs produced by these calculations.

TABLE III
CLOUD COST ANALYSIS

Deployment	Exp. Cores	Pred. Cores	Servers	Cost
Monolithic	5	825	104	$77,688
CPU-based MA	1.8	297	38	$28,386
SLO-based MA	1.4	231	29	$21,663

The column *Exp. Cores* lists the required CPU resources for the 17,500 visits in our experiment, while the *Pred. Cores* those predicted for the considered large amount of users, i.e., calculated by multiplying the *Exp. Cores* column with 165. The third column, i.e., *Servers*, is produced by dividing the previous column with the 8 CPU Cores of the Cloud Providers' plan, which calculates the number of demanded Cloud Servers. The *Cost* is produced by the multiplication of column *Servers* with the amazon server price of $747. These numbers could have been significantly higher with a more complex service, rather than our toy web site implementation.

The table III, at least in a theoretical level, indicates that the cloud resource costs in the case of *Monolithic* service deployment is 2.74 and 3.59 times higher than the *CPU-based MA* and *SLO-based MA* deployments, respectively.

We complement our analysis with an additional experiment that now emulates all user visits, i.e., to validate our previous findings. We set the following parameters in the experiment: (i) 375,000 visits to the web application; (ii) 750 *JMeter* threads run in parallel; and (iii) a 15 min experiment duration. This number of visits is around the double of google.gr in Feb. 2020.

In table IV, we present the results of our experiment, focusing on the containers requiring significant resources, i.e., the *Monolithic*, *Articles* and *Nginx*. The average time in *Monolithic* configuration is lower by 194ms, but requires the double amount of resources, in terms of CPU Cores, compared to *SLO-based MA* configuration. The 95% Line metric confirms again the adherence to the SLA.

Table V provides a cloud cost analysis based on the assumption that some additional CPU cores are required for the rest of microservices, as well as 2 Cores for the OS. Since we assume 8 CPU Core servers, the *Total Cores* become 16 (i.e., requiring two physical servers) and 8 in *Monolithic* and *SLO-based* applications, respectively.

TABLE IV
EXPERIMENTAL RESULTS

Deployment	Containers	Avg.	95% Line
Monolithic	10 Monolithic	1,737	2,611
SLO-based MA	3 Articles - 2 Nginx	1,931	2,309

TABLE V
CLOUD COST ANALYSIS

Deployment	Cores	Total Cores	Servers	Cost
Monolithic	10	16	2	$1,494
SLO-based	5.3	8	1	$747

This supports the same conclusions identified earlier, since there is an equal 50% reduction in both CPU cores cloud costs. Of course, this requires more extensive experimentation but highlights, even in this setting, that MA can produce significant cost savings, which can be further increased with service-oriented elasticity strategies and conservative service performance goals that may be unnoticeable from the users.

IV. CONCLUSIONS AND NEXT STEPS

Cloud providers offer elastic resource allocation that could be matched from flexible microservices-based architectures, to boost performance and reduce infrastructure expenses. In this paper, we conducted a techno-economic study and concluded that: i) although microservices may introduce communication and containerization overhead, the flexibility introduced exceeds this barrier and offers efficient resource management, leading to significant cost savings for the service provider; and ii) an elasticity threshold based on client RT and a targeted conservative value can further reduce over-provisioning of resources and maximize the cost savings, with a negligible impact on the customer satisfaction. Our next plans include: i) an extension of this study based on larger deployments and more complex applications (e.g., multimedia services); ii) the implementation of a microservices orchestration platform that predicts dynamic load and performs elasticity to maximize infrastructure cost savings for a given service performance target, as well as experimentation in novel large-scale test-beds, including EdgeNet [17].

ACKNOWLEDGMENT

This work is partially supported by the "GSRT FUNDING FOR THE YEAR 2018 (Award for the participation in competitive E.U. projects) [Measuring Mobile Broadband Networks in Europe (MONROE)]" which is funded by the Ministry of Development and Investments- General Secretariat for Research and Technology as a reward for the implementation of the project «Multi-homing with Ephemeral Clouds on the Move (MONROE-MEC)- HORIZON 2020, GA -Project Number: 644399» funded by the European Commission.

REFERENCES

[1] M. Armbrust, et al., "A view of cloud computing", Commun. ACM, vol. 53, no. 4 pp. 50-58, 2010.

[2] “Microservices a definition of this new architectural term”. [Online]. Available: http://martinfowler.com/articles/microservices.html

[3] Al-Dhuraibi, Yahya, et al. "Elasticity in cloud computing: state of the art and research challenges." IEEE Trans. Services Comput., vol. 11, no. 2, pp. 430-447, 2017.

[4] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, “Evaluating the monolithic and the microservice architecture pattern to deploy Web applications in the cloud,” In Proc. of CCC 2015, pp. 583-590, 2015.

[5] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, “Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures,” In Proc. of CCGrid 2016, pp. 179-182, 2016.

[6] V. Singh and S. K. Peddoju, “Container-based microservice architecture for cloud applications,” 2017 Int. Conf. Comput., Commun. Automat. (ICCCA), pp. 847–852, 2017.

[7] Ueda, Takanori, T. Nakaike, and M. Ohara. "Workload characterization for microservices." 2016 IEEE Int. Symp. Workload Characterization (IISWC). IEEE, pp. 1-10, 2016.

[8] Al-Debagy, Omar, and P. Martinek. "A Comparative Review of Microservices and Monolithic Architectures." 2018 IEEE 18th Int. Symp. Comput. Intelligence Inform. (CINTI). IEEE, pp. 149-154, 2018.

[9] H. Kang, M. Le, and S. Tao, “Container and Microservice Driven Design for Cloud Infrastructure DevOps,” In Proc. of IC2E, pp. 202-211. 2016.

[10] Prachitmutita, Issaret, et al. "Auto-scaling microservices on IaaS under SLA with cost-effective framework." 2018 Tenth Int. Conf. Adv. Comput. Intelligence (ICACI). IEEE, pp. 583-588, 2018.

[11] Agarwal, Preyashi, and J. Lakshmi. "Cost Aware Resource Sizing and Scaling of Microservices." Proc. of 2019 4th Int. Conf. Cloud Comput. Internet of Things, pp. 66-74 , 2019.

[12] Daniel An, "Find out how you stack up to new industry benchmarks for mobile page speed". [Online]. Available: https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/

[13] “Apache JMeter - Apache JMeterTM”. [Online]. Available: https://jmeter.apache.org/

[14] “Empowering App Development for Developers | Docker.” [Online]. Available: https://www.docker.com/

[15] “Amazon Web Services, AWS Cloud”. [Online]. Available: https://calculator.aws//estimate

[16] “Similarweb.com - Digital World Market Intelligence Platform,”. [Online]. Available: https://www.similarweb.com/

[17] “EdgeNet | A Kubernetes-based internet edge testbed.” [Online]. Available: https://edge-net.org/