# Multi-Objective Economic Adaptation for Service Ecosystems on the Cloud

Marios Fokaefs, *Member, IEEE,* Cornel Barna, *Student Member, IEEE,* and Marin Litoiu, *Member, IEEE*

**Abstract**—Autonomic self-adaptation of cloud applications is a fundamental property of modern cloud infrastructures and is an integral part of the DevOps philosophy. It is an ability, which has helped web developers and engineers to effectively and efficiently deal with demand fluctuations in a flexible manner. This volatility of web software systems has created the need for elasticity on the business level of the system as a reflection of the elasticity on the technical level. Along with scaling and adapting the web application or its cloud infrastructure, there is also the need to adapt prices, marketing policies, client satisfaction and so on. This creates opportunities for even more extensive integrations between DevOps and business operations (BizOps) through models and automated tools. In this work, we propose a novel adaptation method based on a multi-objective optimization, which adapts both technical and economic parameters of the deployed software system with the goal to optimize the application's performance and the economic position of all involved business parties, including service and web application providers, and end-users.

**Index Terms**—cloud economics, self-adaptive software systems, web software systems, cost management, pricing, user satisfaction, devops, bizops.

## 1 Introduction

THE popularity of web technologies and the readiness with which they become available to end customers are creating new patterns with respect to the use of software in unprecedented scale. As a response to the increasing scale and volatility of web traffic, cloud technologies have emerged as a novel and flexible solution to the management and infrastructure support of web applications. Cloud management services offer developers the capacity to adapt their applications' infrastructure support by seamlessly commissioning and decommissioning cloud resources according to their current needs.

Given that web systems and their equivalents (i.e., mobile systems, pervasive computing, cyber-physical systems) also constitute a lucrative multi-billion market, the technical volatility of their nature may influence their business and economic aspects in a similarly frequent and violent manner. For example, during sudden fluctuations of web traffic, prices or other means of revenue generation may stop being optimal, implying a rather dynamic nature. Additionally, given the ability of end users to seamlessly switch between similar applications, it makes market shares be equally dynamic and as a result they cannot be taken as granted for present or future analyses.

The central premise of this work is the need to build software service systems, which are self-adaptive not only with respect to their technical environment, but also concerning their business and economic dimensions. Assuming that the development of any kind of software system carries certain costs and there is an anticipated generation of value, be that monetary or otherwise, we have a way to evaluate decisions taken on the technical layer according to their impact on the economic layer of the application. With this validation index, we can create adaptation algorithms with hybrid goals, taking into account both technical and economic parameters and looking to optimize both the performance of the service and its profit capacity. The ultimate goal for these algorithms is to take adaptive actions on both ends; either scale the infrastructure of the service to regulate performance and control cost, or change prices and marketing policies to control revenue generation and user satisfaction.

To this end, we propose a novel adaptation method for web service software systems on the cloud, formulated as a multi-objective optimization. On one hand, such method will ensure that any scaling action will maintain the performance of the deployed service within acceptable levels. On the other hand, among the possible scaling or adaptive actions that maintain performance, it will choose the one that optimizes any or all of the following goals; the profit of the web service provider, the profit of the client application provider, the end-user satisfaction. According to its output, the method may suggest to scale the infrastructure of the service, by adding or removing cloud resources from its topology, or adapt its economic parameters, for example increase or decrease the service price, or change the number of ads to appear to the end-user through the client application. The stakeholders, who make decisions or are affected by them (service provider, application provider and end-user), are independent and their individual goals may be conflicting but correlated. Therefore, the multi-objective optimization will find the solution, which is best for all three under the constraints imposed on each other.

The rest of the paper is organized as follows. In Section 2, we introduce the problem and motivate the need for additional economic and business optimization and how such methods contribute to multi-faceted development models like DevOps. In Section 3, we outline the proposed methodology in terms of the assumed ecosystem for the web software system on the cloud, the multi-objective optimiza-

• *All authors are with the Department of Electrical Engineering and Computer Science, York University, Toronto, ON, Canada.*
*E-mail: fokaefs@yorku.ca, cornel@cse.yorku.ca, mlitoiu@yorku.ca*

tion problem and how it can be used for adaptive actions, both economic and performance related. Section 4 presents a variety of experiments to demonstrate the performance of our method and its sensitivity to external input parameters. Section 5 discusses a selection of related research works and our work is concluded in Section 6.

## 2 Economic and Performance Optimization for Cloud Applications

While cloud computing and virtualization technologies created new opportunities to guarantee the performance of software systems and better manage their infrastructure costs, the task is neither trivial nor completely straightforward. The two goals, performance and cost optimization, will have to be achieved simultaneously, but they may be conflicting. Additionally, cost management is just one side of economic optimization. While it is tempting to remove resources when possible in order to reduce costs or add resources to improve performance, neither guarantees profitability by itself, since long-term effects on revenue and customer satisfaction need to be taken into account. In this section, we motivate the need for a more sophisticated service management method, which would combine technical, performance, economic and business considerations in making scaling decisions for cloud resources. In the process, we highlight the challenges imposed by the property of cloud computing as an economy of scale and the need to bring closer the technical decision makers with the economic decision makers to optimize the entire space of the problem.

### 2.1 Cloud Computing as an Economy of Scale

Cloud computing has already been recognized to behave as an *economy of scale* [1], [2]. Economies of scale refer to the cost advantages when production increases [3]. In practice, this occurs because the fixed cost of some resource, e.g., machinery, is spread out over more units of product, therefore the average cost per unit decreases as more output is achieved. In the context of cloud computing, the fixed cost resources are the virtual machines and the product units are the served requests; the more requests are served by the virtual machines the more the average cost per request is reduced. Eventually, the progress of the average cost resembles a saw-like graph (Figure 1 and the cost is optimal when the incoming traffic approaches the capacity of the current infrastructure.

To better illustrate this phenomenon, we present a simple example, which was first introduced in [1]. Let us assume that we have a small software company "Maps Co.", which offers a map web service. The company needs 2 VMs to support its daily traffic, under the assumption that each VM can handle up to 10 requests per minute or about 600 requests per hour. Beyond this capacity, a VM becomes saturated, in terms, for example, of its CPU utilization, which will have an impact on the service's performance and consequently on its quality to end-users. Additionally, for each VM the company is charged with 60 cents per hour ($\$0.6/h$) and it charges 0.12 cents per request ($\$0.0012/req$). At full capacity, 1,200 requests per hour, the company needs

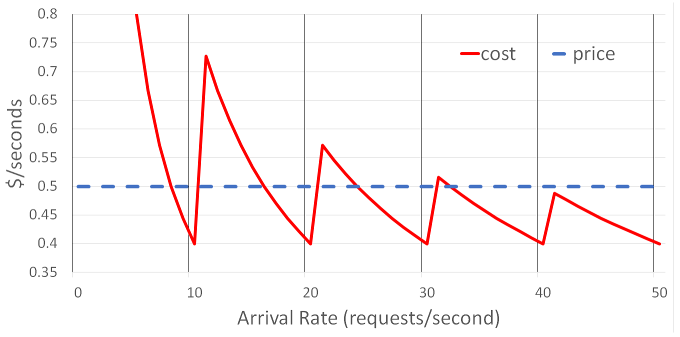

Fig. 1. Economy of Scale: Average cost and price per request. (Figure from [1])

2 VMs at a total cost of $\$1.2/h$, while it generates $\$1.44/h$, resulting in $0.24 net profit.

Let us now assume that the service starts receiving traffic marginally above its capacity, i.e., 1,201 requests per hour. As already mentioned, this extra traffic will bring the infrastructure to saturation and from a purely performance perspective it can trigger a scaling action to ensure consistent quality. However, an analysis from the economic perspective will give us a different picture of the situation; the third VM that needs to be added will increase the total costs to $\$1.8/h$, while the revenue with the single extra request will increase to $\$1.4412/h$. This will result in losses of $\$0.3588/h$. This scenario illustrates the complexity of scaling decisions in a real setting, where software is both a technical and a financial artifact. Addressing potential performance issues arising from fluctuations in traffic may cause problems to the economic position of the system. Therefore, it is evident that any decision mechanism will need to take a holistic approach and somehow take into account all aspects of the system, including performance, economics, but also business, with respect to clients and competitors.

Going back to the example, if we extrapolate the numbers even further, we can see that if the traffic increases to 1,500 requests per hour, the total revenue generated by the extra traffic will cover the extra costs from the third VM and the system becomes marginally profitable. Even further increase to 1,700 requests per hour will bring the system's profit to the previous levels, when there were only two VMs. This analysis shows that there can exist different strategies under different circumstances and as a result there is a need for a more dynamic decision making process, so as to give the flexibility to successfully react to different situations.

### 2.2 Integration of DevOps with BizOps

The multidimensional nature of the scaling problem, as we have discussed it so far, including performance, economic and business considerations, creates new requirements to approach software development from a more holistic point of view. This cultural shift is already in progress with *DevOps* [4], which advocates the integration of IT operations management with the development process and vice versa. In practice, the new paradigm recommends that the two teams, operations and development, mingle, work closely together and generally maintain a consistent line of communication. This has clear implications on processes, but also on the expertise of software professionals [5].

With our work, we aim to further extend the notion of DevOps for web software systems to also include the business operations (BizOps) side of the system to a hybrid we call the *DevOps/BizOps* paradigm. To achieve any kind of hybrid process, we further claim that there need to exist two driving forces, namely *integration* and *automation*. Integration implies any method that brings together the knowledge and skill sets of the software professionals towards a common goal or practice. This way, everything is available to everyone at every step of the software's lifecycle, from development to maintenance to management and the final delivery to end-users. On the other hand, automation is necessary so that not everything and everyone is required at all times and the process is streamlined in order to be efficient. Practically, integration makes automation possible and both enable the hybrid model, either DevOps or DevOps/BizOps. In our work, as we detail next, integration is achieved through the definition of a complex model, which combines technical (response time, utilization, arrival rate) with economic (price, advertisements, cost) and business parameters (client utility) of the subject software system, and with the multi-objective optimization, which considers the payoffs and possibly conflicting interests of all business stakeholders of the system. The model and the optimization are incorporated as the analysis and planning phases of a MAPE autonomic management system (besides monitoring and execution) to enable automation, the second necessary property for the DevOps/BizOps integration.

To fulfil the DevOps philosophy, each of the three teams (development, operations and business) has its own mission and is responsible for specific tasks, but, eventually, they all need to collaborate and communicate. Development is the team who is primarily responsible for functionality. They implement the required features at development time, but also determine and document the quality parameters and non-functional properties of the system by dynamically analysing and profiling the system at runtime. The team will also need to define the value created by the functional properties of the system. The operations team is concerned with the quality and performance of the system mainly at runtime. At deployment time, the input from the development team is analysed to determine the resources needed to deploy the system and make it available to users. Secondly, the team builds the performance models of the system, based on the same input and the available resources, to manage the software at runtime and determine the client utility and generated value based on its runtime quality. Finally, the business operations heeds the client needs and how the competition addresses those to find what are the features that are to be focused on and passes those to the other teams before development. After deployment, the team is responsible to maintain the system's profitability and its competitiveness within the ecosystem by dynamically adapting the strategies and policies according to environment changes and input from the other teams.

## 3 A Hybrid Adaptation Strategy for Cloud Ecosystems

According to the proposed method, the web software operates within an extended ecosystem with multiple stakeholders acting as providers and consumers, including cloud and service providers on one end and client applications and end-users on the other end. There are technical interactions between these parties, e.g., clicks by end-users and service requests by client applications, which result in cloud resource demands. Towards the opposite direction there are equivalent economic transactions; cloud providers charge for resources, service providers charge for service requests, while client applications generate revenues through advertisement. Using technical parameters, like traffic and performance metrics, and nominal economic parameters, for example cost per VM and revenue per ad, as input, we define functions to calculate service and client application profits, and user satisfaction to complete the underlying economic model for the web system.

As the incoming traffic of the web system fluctuates, it affects the system's performance and economics and eventually the end-user satisfaction in terms of response time. Based on performance and economic models, the proposed method employs the multi-objective optimization to scale the infrastructure of the application (i.e., add or remove VMs) in order to regulate its performance, while at the same time optimizing the economic position of the three stakeholders by changing, if necessary, the service price and the number of ads per click to appear in the client application.

### 3.1 Web Software Ecosystem and Economic Model

In our work, we assume that the web software system operates within an extended ecosystem, as shown in Figure 2. The figure depicts the structure of a 4-party ecosystem, as well as the technical and economic transactions between the stakeholders. In this form, the ecosystem consists of:

- *cloud providers*, who offer virtual resources, such as virtual machines, network bandwidth, data storage, platform-as-a-service;
- *web service providers*, who offer a back-end web service or microservice, which exposes special functionality or a proprietary dataset;
- *client web application providers*, who offer a web application, which consumes the web service and exposes it through a front-end interface to end-users;
- *end-users*, who use the web application.

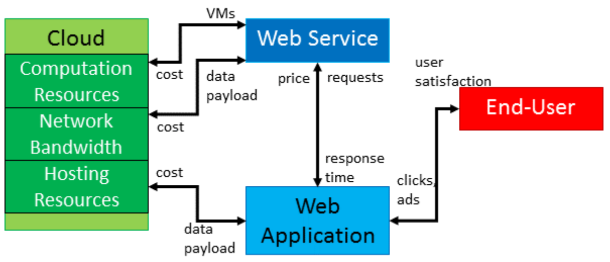

Fig. 2. Cloud-deployed web software ecosystem

The ecosystem is modelled in the likes of a distribution channel or a supply chain with upstream and downstream suppliers (providers) and downstream consumers (clients).

Cloud providers act as upstream suppliers offering computation, network, storage and hosting resources to software developers. In the case of public clouds, the provider charges for the consumption of the virtual resources based on their usage by the consumers, for example VM usage is charged per hour and bandwidth is charged per data transferred. We use these direct charges as costs for software providers, either service or application providers, who host their software on the cloud. In the case of private, dedicated or even hybrid clouds, where part or all of the cloud resources are integrated with the software provider, we can still consider resource costs either through the amortization of capital and equipment or through the value that the resources create for the software provider.

The service providers act as downstream suppliers; they consume resources from the cloud provider to be able to offer their service to web applications. Therefore, the service provider will have to incur costs for leasing VMs, network bandwidth and other cloud resources. According to the elasticity principle, these costs vary as the traffic fluctuates and the provider has to adapt the service's infrastructure as a response. In our work, we consider the service provider to be charged for VMs in the topology, as well as for data transfer, given an average payload per outgoing response per request. On the other hand, the service provider charges for the use of the service by client applications. Although there exist a number of pricing models for web services [6], one of the most popular ones, preferred by such big service providers as Google[1], is the utility model with a free tier; the provider allows clients to issue requests to the service up to a limit per time period (e.g., per day or per hour) for free and then charges them for any additional requests on a per request basis. In this work, we exclude the free tier from consideration for simplicity. Based on the costs and revenues for the service provider, we define the corresponding profits as follows:

$$\pi_S = p_S \times \lambda - p_i \times W - n \times p_n \times \lambda$$

where $p_S$ is the price per request in $\$/req$, $\lambda$ is the arrival rate of the web service in $req/s$, $p_i$ is the cost for VMs in $\$/s$, $W$ is the number of VMs in the cloud topology of the service, $n$ is the average data payload of a response to a request in $KB$ and $p_n$ is the cost per data transfer in $\$/KB$.

The application providers are also downstream suppliers. On one hand, they use web services and consume hosting and network resources from the cloud provider, while, on the other hand, they offer a client application to end-users, through which the web service becomes available to them. Hosting services by cloud providers are usually charged based on the number of incoming requests to the application (cost per request) and the response's payload (per data transfer). We can combine these two costs in one, which is incurred by the application provider and is payable to the cloud provider. Additionally, the application provider is charged for service requests independently, a cost which is payable to the service provider on a per-request basis. With respect to income, web applications have a number of methods to generate revenue [7], out of which one of the most popular is advertisement. According to this model, ads will appear on special banners in pages of the web application, for which an intermediary ad broker, for example Google AdSense[2] will pay money to application providers regardless of whether the banner is clicked by the users. Under the assumption that every click/request to the application produces a new web page, the application can display an average number of ad banners per web page, for which the broker will pay an average revenue per one thousand appearances ($RPM$). Based on the costs and revenues for the application provider, we define the corresponding profits as follows:

$$\pi_A = \frac{\gamma \times RPM}{1000} \times \lambda - H \times \lambda - p_S \times \lambda$$

where $\gamma$ is average number of ad banners per web page, $RPM$ is the average revenue per thousand ad impressions in \$, $\lambda$ is the arrival rate in $req/s$, $H$ are the hosting costs in $\$/s$ payable to the cloud provider, and $p_S$ is the price per request in $\$/req$ payable to the service provider.

The end-users represent the downstream consumers of the ecosystem, who are eventually the ones to use the client application. The users generate clicks to the web application, which are translated in requests to the web services, thus generating the incoming traffic to the overall web software system. Although there can exist a number of relationships between clicks and service requests (e.g., many clicks generate a few requests or vice versa), our method assumes that every click generates exactly one request to the back-end service. If the relationship is known (linear or otherwise), it can be easily replaced in our method. It is rare for web applications to directly charge end-users for using the web application. As we have already mentioned, revenue for web applications is generated through advertisements and the traffic created by end-users. However, "cluttered" interfaces with ad banners, which may also affect content placement, can be detrimental for the users' perception of the application's quality. In addition, user perception on application quality is affected by how fast the application responds to user requests. Based on these two factors, response time and number of ad banners per web page, we define *user satisfaction* as value index for end-users. In practice, we base our definition on a combination of previous works. Litoiu and Barna [8] define the notion of *utility* as a function of response time, normalized according to an upper and a lower bound. Chen et al. [9] define user satisfaction as a function of response time and price. Given that end-users are not directly charged, in our definition, we replace price with number of ad banners per web page. We also use the notion of normalization by Litoiu and Barna [8] for both response time and ad banners. Formally, we define user satisfaction as:

$$U = a \times \frac{\gamma_u - \gamma}{\gamma_u - \gamma_l} + b \times \frac{R_u - R}{R_u - R_l}$$

where $a$ and $b$ are weight factors, such that $a + b = 1$, $R$ is the monitored response time of the web system, with $R_u$ and $R_l$ be the upper and lower bounds respectively, and $\gamma$

1. https://developers.google.com/maps/faq#usage_pricing

2. https://www.google.com/adsense/start

is the average number of ad banners per web page, with $\gamma_u$ and $\gamma_l$ be the upper and lower bounds respectively.

## 3.2 Multi-Objective Optimization

Out of the of four types of stakeholders, only two are considered "active" in the context of our work. "Active" stakeholders are considered those who can take adaptive actions, technical or economic, as a response to changes. In the presented ecosystem, service providers can change the virtual infrastructure, increase or decrease the number of VMs, or the service price, and application providers can change the number of ad banners per web page. Cloud providers provide input to the ecosystem, in resources, and to the economic model, as cost for resources. On the other end, end-users generate traffic, receive the effect of changes and eventual quality of the web software system, and impose economic and quality constraints to the system through their user satisfaction, but their specific behavior or decisions are not part of the proposed analysis.

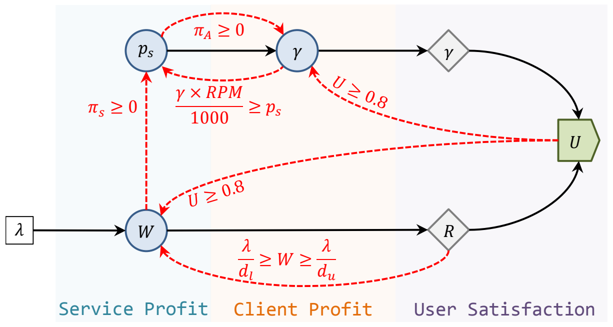

Fig. 3. Optimization problem and variable constraints. Decision output variables (circles), input to objectives (diamonds), input to model (squares), objective functions (squared arrows), constraints (red arrows), variable passing (black arrows).

The stakeholders have different and in most cases conflicting objectives, which, on one hand, can be achieved with independent actions, but, on the other hand, they impose constraints and affect each other. For example, the service provider cannot increase the price per request above the client's marginal profit, because then the client may look to switch web services for something with lower usage costs. Therefore, any adaptive actions need to simultaneously satisfy all parties' objectives to the best degree possible. For this reason, we formulate decision-making process of all parties as a *multi-objective optimization* problem. Formally,

$$\underset{p_S,W,\gamma}{\text{maximize}} \begin{cases} \pi_S = p_S \times \lambda - p_i \times W - n \times p_n \times \lambda \\ \pi_A = \frac{\gamma \times RPM}{1000} \times \lambda - H \times \lambda - p_S \times \lambda \\ U = a \times \frac{\gamma_u - \gamma}{\gamma_u - \gamma_l} + b \times \frac{R_u - R}{R_u - R_l} \end{cases} \quad (1)$$

$$\text{subject to} \quad \frac{\gamma \times RPM}{1000} \geq p_S \quad (1)$$
$$\pi_S \geq 0 \quad (2)$$
$$\pi_A \geq 0 \quad (3)$$
$$U \geq 0.80 \quad (4)$$
$$\gamma_u \geq \gamma \geq \gamma_l \quad (5)$$
$$\frac{\lambda}{d_l} \geq W \geq \frac{\lambda}{d_u} \quad (6)$$

, where $d_u$ and $d_l$ are the upper and lower bounds of the capacity of a single VM in $req/s$. We define the capacity of a VM as the number of requests that it can accept per time unit, e.g., per second, without exceeding a particular level of CPU utilization. Without loss of generality, we assume that the capacity of a cloud topology is linear to the number of VMs. For example, a topology of 4 VMs has twice the capacity of a topology with 2 VMs of exactly the same type and size.

The objectives of the problem correspond to the goals of the three stakeholders of the ecosystem, service and application providers, and end-users. The objectives are to be maximized simultaneously by finding appropriate values for the decision variables; the service price, the number of VMs in the service topology and the number of ads per web page. As it can be seen in Figure 3, service price ($p_S$) and number of VMs ($W$), as decision variables, are passed as input to the service goal. Subsequently, number of ads ($\gamma$), as a decision variable, is passed as input to the user goal. In addition, number of VMs, based on the input arrival rate ($\lambda$), determines the response time ($R$) of the system, which is passed as input to the user goal, as well. The first constraint dictates that the service price cannot be more than the nominal revenue per request the web application makes from advertisements. This constraint imposes an upper bound to the service price, as shown in the figure. The service price is also constraint by the number of VMs, so that the profit of the service is positive (second constraint in the problem). This imposes a lower bound to the service price. Similarly, the number of ads is constrained by the price, so that the client's profit is always positive (third constraint), and by the user satisfaction (fourth constraint). The fourth constraint ensures that any solution will not drop user satisfaction below 80%. The fifth constraint bounds the number of ads. Finally, response time, and by extension user satisfaction, constrains the number of VMs, based on their capacity and the input arrival rate (sixth constraint).In practice, this last constraint is related to the quality of service; the upper bound prevents the optimization problem from adding unnecessary resources, which will only increase the costs, but have no impact to performance, while the lower bound ensures that the topology will have enough VMs so that it doesn't become saturated, eventually affecting the service's response time and by extent the users' satisfaction.

It is neither trivial nor always possible that a single solution is found that optimizes all objectives at once. For this reason, other methods are used to produce a number of possible *Pareto optimal* solutions. A Pareto optimal solution is one for which none of the objective functions can be improved without degrading some of the other objective values. A popular method to produce such solutions are genetic or evolutionary algorithms. These algorithms will produce Pareto efficient solutions, such that they satisfy the constraints of the problem. In our work, we use the NSGA-II [10] genetic algorithm to produce the solutions. Eventually, out of the total possible solution, we need to pick one. This is achieved by imposing an external condition, an overall goal, which is usually specific to the domain. In the proposed method, we normalize each of the objectives over the maximum value produced by the genetic algorithm's solutions and then calculate a weighted average as:

$$\underset{p_S, W, \gamma}{\text{maximize}} \quad \frac{w_S \times \pi_S^i}{\underset{i}{max}(\pi_S^i)} + \frac{w_A \times \pi_A^i}{\underset{i}{max}(\pi_A^i)} + \frac{w_U \times U^i}{\underset{i}{max}(U^i)}$$

where $i$ is the index of the Pareto optimal solution and $w_S + w_A + w_U = 1$.

The weights dictate the specific overall goal. For example if either $w_S = 1$, or $w_A = 1$, or $w_U = 1$, we optimize the objective of a particular stakeholder, service, application and user, respectively. Any other combination of weights will attempt to find compromising solution over all stakeholders. When all weights are equal, we consider this to be the optimization of the ecosystem's welfare.

### 3.3 Multi-Objective Adaptation

The output of the optimization after applying the overall goal provides us with (a) a service price, (b) a number of VMs to support the service and (c) a number of ad banners to appear in the web application. We use the second argument to scale the topology of the service if necessary. In practice, the proposed elasticity method follows a MAPE architecture for adaptive software systems [11]. The monitoring (M) component of the architecture gives us the arrival rate for the service, which corresponds to service requests and application clicks, and the resulting response time of the service. These metrics are passed as input to the optimization problem, which constitutes the analysis (A) component. The number of VMs and the number of ads, as output from the problem constitute the planning (P) component and they are passed to the execution (E) component, usually an orchestration engine or a general autonomic management system, so that the topology is scaled and the application's configuration is adapted with the updated number of ads. The optimization problem is invoked every MAPE cycle, for example every few minutes. The genetic algorithm produces a set of valid solutions and the one that maximizes the overall goal is applied by the elasticity method.

### 3.4 Assumptions and Limitations

In our work, we make a number of assumptions, either for simplicity or for realizability. In this section, we discuss some of these limitations and point out how they can be mitigated or why they may be valid in an actual setting. An important assumption pertains to the structure of the ecosystem as assumed by the optimization problem. First, we consider the cloud provider to be a passive party in the ecosystem, who simply provides infrastructure support and costs. In reality, the cloud provider may also decide on restraining their clients' usage or change the prices of their resources, practically affecting the service provider's cost policy. A popular example of dynamic cloud resource pricing is the case of *spot instances*, for which the price is the result of demand and client behavior. Our work can be easily extended to include the cloud provider as an active player, with a relevant profit model, an additional objective and a few more constraints. In order for the cloud provider to be considered an active player, we will need to define an additional profit model with costs, most probably from capital expenses and energy costs, and revenues, coming directly from the virtual resources. Having this model, it would be straightforward to add a fourth objective function in our optimizer and a few additional constraints. The rest of the methodology will remain the same. At this point, it would also be useful to clarify that when we talk about web services, we do not talk about cloud services. the two parties, cloud provider and service provider, are separate and the goal of our method is to optimize the software layer.

A second assumption with respect to the structure of the ecosystem is the degree of integration between the different stakeholders. For example, the software provider may not necessarily use public cloud resources to support the web software. Although there is a visible shift towards public clouds, especially among small to medium businesses, for various reasons it is not unprecedented for firms to employ their own resources, either for cost or privacy purposes. In the context of our work, as long as a nominal cost for the use of a VM per time unit and a nominal cost for web traffic data payload can be calculated, our method can be used as is. Additionally, it is more often than not the case that service providers may also offer a graphical interface to expose their software directly to end-users. This implies that service and application providers can be fully integrated as a single stakeholder and the application no longer has the burden of a service price per back-end request. This can be alleviated either by simply merging the service and application objectives and removing the service price as a decision variable or assuming that there is a nominal cost for service consumers, which we can match to the service price and, thus, use the proposed methodology as is. Moreover, for simplicity reasons, in this work we assume that there is a single entity per type of stakeholder in the ecosystem, one cloud provider, one service provider, one application provider and one end-user. It is part of our immediate future plans to study a vertically extended ecosystem with multiple parties. This will add to our analysis the notions of competition, composition, and the notion of hybrid clouds, where software providers may use different services from different cloud providers or a combination of private and public resources. A final aspect of the structure assumption is that of the information flow between stakeholders. In the context of our work, we assume that the ecosystem is an environment of perfect information, in other words everybody knows everything. An imperfect information ecosystem can still be a valid environment for our method using prediction and estimation models to calculate values for the unknown variables. Nevertheless, when the components of the software system are hosted in an actual integrated environment the perfect information assumption is not too unrealistic.

Such environment is offered by the IBM Bluemix platform[3], where back-end microservices and front-end web applications are hosted in the same environment having access to common cloud resources and management services, one of which can be our multi-objective optimizer.

A third assumption has to do with the nature and role of the web application in the overall system. In our analysis and our experiments later, we view the web application as a hollow vessel that simply exposes the web service to end-users. In reality, the functionality of an application could be much more intricate. The practical impact to the proposed method is that, first, the relationship between clicks and service request is not as simplistic as the one described in this paper, and, second, the overall response time of the system depends on both the web service and the web application. We already commented on the first factor saying that if there exists a function or a model that connects clicks and service requests, it can be used and be easily integrated with the proposed method. Response time is a more complicated factor and gets even more complex when considering that loading ads, and in fact a variable number of those, will have an additional impact on the application's response time. We have decided to leave this task out of the scope of this work and focus more on the multi-dimensionality of the decision problem. In any case, response time is provided as input to the optimization problem and it can come from any source, complex or not.

The model and the ecosystem are designed to be simple enough so that they can be easily understood and so that they can be parameterized and extended to fit more properties and more complex structures. Nevertheless, these assumptions do not threaten the applicability or the practicality of the proposed method. One question pertaining to these properties is who owns the autonomic management system, which employs the adaptation method. In the context of software adaptive systems, it is expected that either the service or the application provider will invoke such a manager. In our experiments, we consider the service provider to be the decision maker and somehow propagate the effect of the decision to the application, so that it can be adapted as well. In the scenario where service and application are in fact one entity, the problem does not exist as the integrated software provider is the sole decision maker and has ownership of all the information necessary to invoke the adaptation method. We are planning to consider this setting in our immediate future work.

## 4 Evaluation

In order to validate the performance and the applicability of the proposed method, we designed a number of experiments in a simulated cloud environment. We have used realistic simulation models built with reference to Amazon EC2 and tested in prior work [12]. The first set of experiments is designed to evaluate how the method performs when trying to optimize different objectives for different stakeholders or for the ecosystem overall. The second set of experiments are sensitivity tests to investigate the method's performance under a variety of weights for user satisfaction and for the overall goal.

3. http://www.ibm.com/cloud-computing/bluemix/

For our experiments we used a typical web *bookstore* software system, developed using Java EE technology. The service computes various user requests at the application server and further SQL commands (select, insert, update) on the data server. In the initial topology, the service requires four cloud virtual machines (VMs) for different components of the application: the database server (MySql), two web application servers (Tomcat), while a fourth VM was acting as a load balancer (Apache 2) to distribute the incoming web requests between the application servers. In the context of our experiments, the web application is a transparent layer used only for the economic analysis; the user requests are generated as clicks which pass through the application without any delay directly to the back-end web service as service requests and the responses are returned to the end-users with a delay equal to the measured response time.

To automate the deployment process and management of the topology (adding/removing instances, changing configuration of instances, monitoring the instances), we have used the Hogna platform [13]. In order to capture the performance of the web application under different workloads and different settings for the infrastructure, we used the OPERA model [14]. OPERA gave us access to such metrics as response time, arrival rate and CPU utilization for applications deployed in cloud.

To emulate the user interaction with the application, we developed a workload generator where a number of users would issue requests to the application. We start with a small number of users which are increased to higher levels and stay there for a period of time before they decrease to their initial levels for a short period. The workload is exactly the same in terms of data points, number of users and requests for all experiments in the paper so that the results are unbiased and comparable. Eventually, we generate about 750 sets of requests for each experiment.

The first set contains four experiments, one for a different overall goal:

- **Exp1:** optimize over service profit ($w_S = 1, w_A = 0, w_U = 0$);
- **Exp2:** optimize over application profit ($w_S = 0, w_A = 1, w_U = 0$);
- **Exp3:** optimize over user satisfaction ($w_S = 0, w_A = 0, w_U = 1$);
- **Exp4:** optimize over ecosystem welfare ($w_S = 0.33, w_A = 0.33, w_U = 0.34$).

The second set contains a number of sensitivity tests:

- **Exp201-205:** 5 repetitions of Exp2 to assess the robustness of NSGA-II and its ability to produce consistent results;
- **Exp401-411:** 11 experiments with various weights for overall goal (as extensions of Exp4), where $w_S, w_A, w_U < 1$;
- **Exp501-507:** 7 experiments with various coefficients for the user satisfaction factors.

For all experiments, the parameters of the optimization problem are maintained the same, as shown in Table 1. For some of the economic parameters, we consulted similar services, including Amazon AWS, Google APIs and so on. We drew representative values for technical parameters, like response time and VM capacity, based on our experience and some trial experiments. To solve the multi-objective optimization problem, we used the 'MCO' (Multiple Criteria Optimization Algorithms and Related Functions)[4] package for the R project for statistical computing. MCO employs the NSGA-II genetic algorithm to solve the multi-objective problems. We set up the algorithm so that it produces 100 solutions every time it is invoked. Other than the population size, we specify no other parameters, such as crossover and mutation probabilities, and we use the defaults provided by the package. While all experiments share an identical workload generator (i.e., the same number of users per data point), the solutions produced by the genetic algorithm may not be identical between experiments or even between observations. However, the general results for all 750 data points are expected to be similar and thus comparable between the experiments.

TABLE 1
Summary of parameter values used in the experiments

| Variables | Value | Unit |
|---|---|---|
| $RPM$ | 0.7 | \$ |
| $H$ | 8.50E-05 | \$/KB |
| $p_i$ | 2.64E-04 | \$/sec |
| $d_u$ | 21 | req/ sec |
| $d_l$ | 15 | req/ sec |
| $n$ | 10 | KB |
| $p_n$ | 3.50E-06 | \$/KB |
| $\gamma_u$ | 10 | ads/ page |
| $\gamma_l$ | 1 | ads/ page |
| $R_u$ | 200 | ms |
| $R_l$ | 45 | ms |

## 4.1 Goal Experiments

Figures 4 to 7 show the results for the first set of experiments. In every figure, the top plot shows the number of VMs (purple) deployed for each observation of the experiment and the average CPU utilization (blue) of the topology. The second plot depicts the incoming traffic for the web system in terms of requests per second (blue) and the system's response time (green) in milliseconds. The third plot shows the other two decision variables of the optimization problem, service price (teal) and number of ads (orange). The last plot shows the user satisfaction (red), the service profit (teal) and the application profit (orange) per observation. Table 2 summarizes the results of the four experiments in terms of averages and totals of performance and economic parameters.

As it is naturally anticipated, a goal that optimizes a single objective will try to increase the decision variable that has a positive impact on the object and conversely restrain the variable with the negative impact. Therefore, when we optimize for service profit (Exp1, Figure 4), the price is driven to its highest possible value, which drives the client profit down. Similarly, when optimizing for client profit (Exp2, Figure 6), the number of ads is increased, which drives the user satisfaction down.

In another observation, all overall goals seem to maintain good performance in terms of average CPU utilization and

4. https://cran.r-project.org/web/packages/mco/mco.pdf

TABLE 2
Goal experiment results

| | Exp1 | Exp2 | Exp3 | Exp4 |
|---|---|---|---|---|
| **Sum of VMs** | 3650 | 3609 | 3700 | 3621 |
| **CPU** | 62.13 | 62.84 | 61.18 | 62.56 |
| **response time** | 67.63 | 70.45 | 67.04 | 68.74 |
| **ads** | 1.08 | 2.74 | 1.00 | 2.67 |
| **service price** | 5.00e-04 | 1.07e-04 | 4.82e-04 | 4.68e-04 |
| **service profit** | 23.42 | 2.97 | 22.49 | 21.76 |
| **client profit** | 8.91 | 89.69 | 6.90 | 68.57 |
| **user satisfaction** | 92.24 | 82.12 | 92.88 | 83.03 |

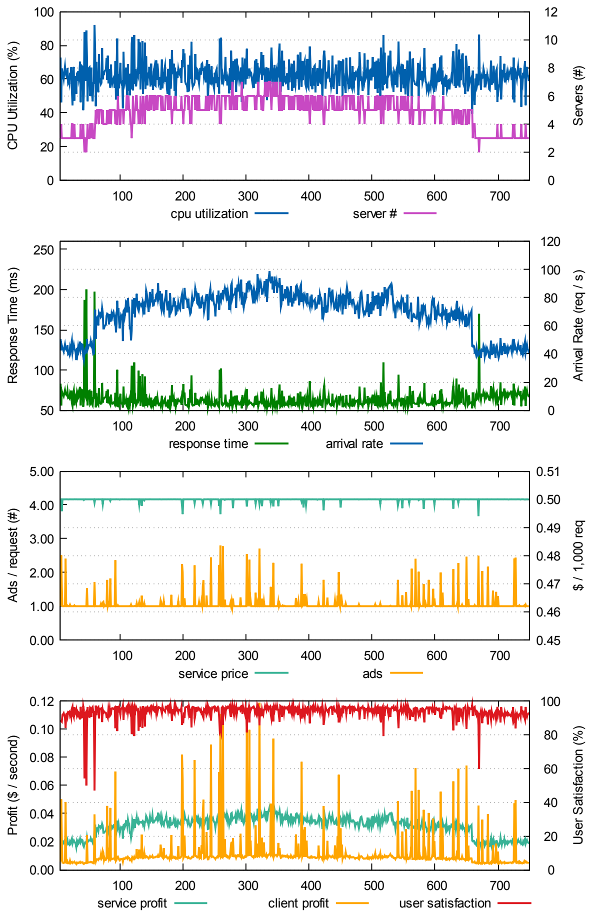

Fig. 4. Multi-objective optimization for service profit (Exp1).

response time. In retrospect, we found that this was due to the optimization of number of VMs before actually running the optimizer and when setting the bounds. The bounds are set purely based on incoming traffic and the capacity of the VMs. When these are set the optimizer seems to prefer to get closer to the upper bound as this will have a small effect on service profit (as VMs are quite inexpensive compared to the revenue generated by the served requests) and it is vital for maintaining a low response time for a high user satisfaction. Indirectly, this also seems to be the reason why the service profit and the user satisfaction goals seem to yield very similar results; with a guaranteed response time, the decision variables that maximize the two individual objectives are independent with each other, and therefore, the algorithm prefers solutions that maximize both goals in the expense of the client profit. Overall, in all experiments

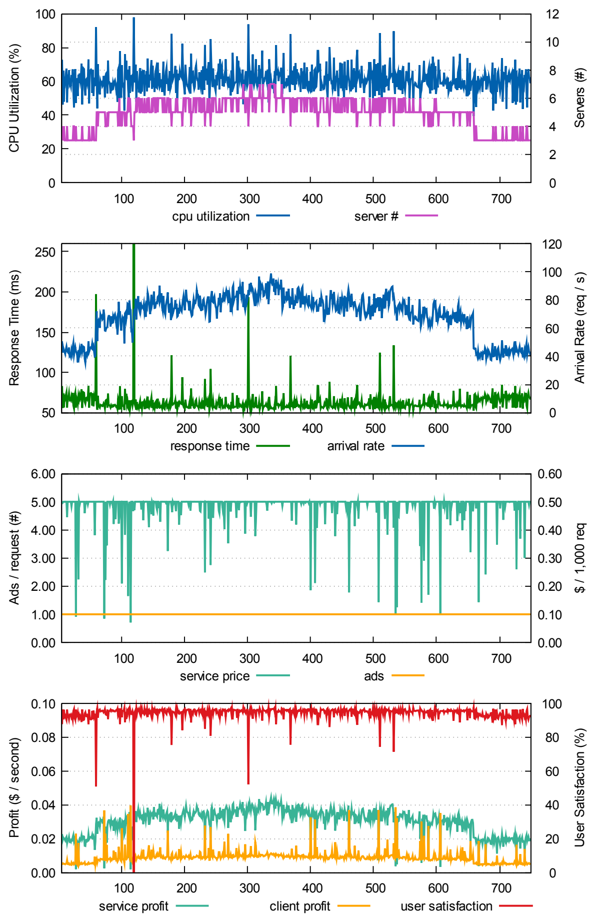

Fig. 5. Multi-objective optimization for user satisfaction (Exp3).

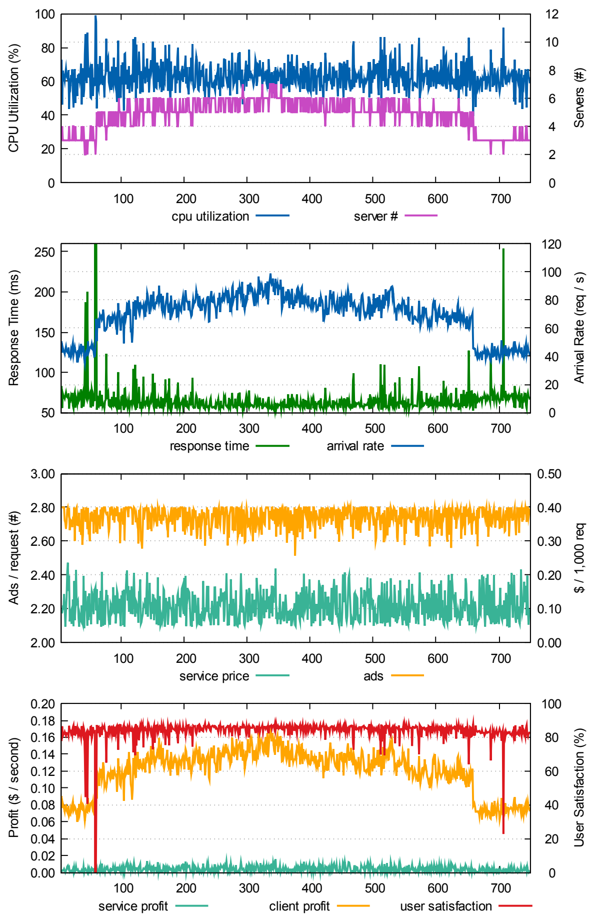

Fig. 6. Multi-objective optimization for client profit (Exp2).

average CPU utilization was maintained around 62% and average response time between 67 and 70 milliseconds, very close to the minimum. In all experiments, the number of VMs reached up to 7 machines. As it can be seen in the figures, this number fluctuates a lot with a rate of $\pm 1$ machine, which is due to the arrival rate being very close to the capacity of the machines and causing a very frequent change. In practice, this can be controlled either by not permitting VM removals so easily or by allowing the system to rest for a few samples after a scaling change has occurred. In addition, we can mitigate this phenomenon by having a VM in the reserve and simple starting and stopping it in order to avoid going through the creation and removal process, which is more time consuming. With respect to the total number of VMs (i.e., the sum of VMs that have appeared for an entire experiment) used in the experiments, we see that when the objective is to optimize the user satisfaction more VMs are employed. This is because response time becomes important as it is a factor in user satisfaction. This was the reason why Exp3 (Figure 5) had the best relative CPU utilization and response time among all experiments.

Optimizing the client profit implies a preference towards solutions with low service price and high number of ads, which both have a negative effect to service profit and user satisfaction. In case of Exp2 (Figure 6), if the service provider could study the data, it could have been seen that the client has a large profit margin in which the service provider could tap, without further affecting the user satisfaction. In an eventual increase of the service price, the client could not respond with a further increase in advertising due to the lower bound of user satisfaction. This is in part what happens in Exp4 (Figure 7), where all goals are given an equal weight. As a result, the service provider now shares in to the client's profits in an effort to increase the overall ecosystem welfare. This discussion shows that the proposed method, apart from an adaptation strategy and technique, can also be used as a consultation tool for stakeholders to identify opportunities for collaborations and increase their individual profit and value. Overall, in the setting of Exp4 (Figure 7), each stakeholder is in a better position than the worst they could find themselves if any other of their partners would try to maximize their goal unilaterally. To this end, such a setting can be considered optimal in a collaborative environment.

### 4.2 Sensitivity tests

In the first sensitivity test, the goal was to evaluate the robustness of NSGA-II. Due to the probabilistic nature of genetic algorithms, the generated solutions may be different from execution to execution of the optimizer. For this reason, we ran Exp2 5 times with exactly the same static input and investigated if the different solution sets affect the final decisions of the proposed method. As it can be seen in Table 3, the performance metrics, the decision variables and the objective values differ very little between the five runs. Although not identical, they are close enough so as

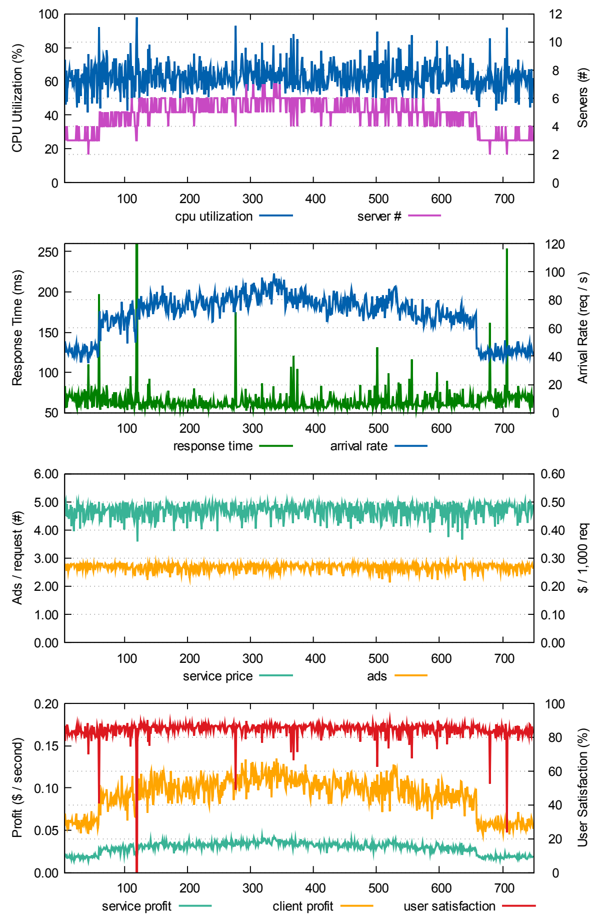

Fig. 7. Multi-objective optimization for ecosystem welfare (Exp4).

not to affect the decision making process due to this small variation. By extension, this implies that the algorithm will maintain the same behavior between runs of the optimizer within the same experiment (i.e., for different samples).

The second sensitivity test was designed to investigate the impact of selecting objective weights for the overall goal. We designed 11 iterations of the same experiment with different weight combinations. In practice, we steadily fed the end-user weight, as from the previous experiments we observed that this was the objective which was not maximized when other objectives where given priority. Therefore, we wanted to see which combination of mixed weights would alleviate this situation. Table 4 presents all weight combinations and the corresponding objective values, while Figure 8 graphically shows the progression of the objectives for all the experiments.

The results of this test show that when the weight of the end-user objective ranges between 0 and 0.73, user satisfaction varies minimally. After this point, user satisfaction greatly increases for another two weights (0.74 and 0.75) and then plateaus. While the exact point of this behaviour depends on the parameters of the rest of the problem, this observation implies that there is a tipping point with respect to the user satisfaction, which can be taken advantage by the client mainly to temporarily achieve high profits and then return in high and stable user satisfaction.

In the third sensitivity test, we optimize user satisfaction under different coefficients for its factors. In practice, we

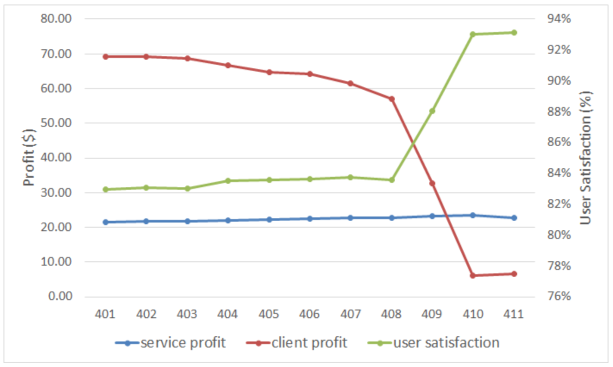

Fig. 8. Objective values for different weight mixes.

want to see how the results may be affected if we assign different importance to each of the factors. Table 5 summarizes the results of the test. We have tried 7 mixes for the coefficients. Given that we optimize for user satisfaction, the number of ads stay at its lowest level. In this respect, user satisfaction is affected by ads in a constant manner, leaving response time to possibly vary. In the first 6 experiments, where ads are assigned a non-zero weight, we observe little difference in all indices. In the last experiment, where only response time is considered as a factor for user satisfaction, we see a sudden drop in the service profits. On one hand, this is due to the fact that the optimizer, in an effort to minimize response time, assigns about 100 total VMs more than in the other experiments, which increases the service costs. On the other hand, since ads are now irrelevant for the user satisfaction, the genetic algorithm creates more solutions that increase the client's profits by having reduced service prices.

## 5 Related Work

Boehm [15] outlines the challenges around software engineering economics decisions and also discusses how software-economics principles can be applied to improve software design, development and evolution. He defines software engineering fundamentally as an activity of decision making over time with limited resources and usually in the face of significant uncertainties. Uncertainties pose a crucial challenge in software development that can lead to failure of systems. Uncertainties can arise from inaccurate estimation. For example, cost-estimation models developed for traditional development processes no longer apply to modern architectural styles and development processes, such as the ones around web software systems and cloud computing.

Boehm and Sullivan [16] put forward a utilitarian view for software evolution, according to which the system in order to create value for any involved party, it must create value for all whose contributions are critical to the project's success. Failure to satisfy any of those critical contributors will mean overall failure of the project thus failing to satisfy any of the involved parties. Through our work, we emphasize the importance of the utilitarian approach.

Lyons et al. [6], [17] recognize the distinctive characteristics of web systems as business artifacts and study various business models to deliver and price web services. These business models and the strategic decisions around them

TABLE 3
Sensitivity test 1: NSGA-II robustness for Exp2.

| Experiment | CPU | response time | ads | service price | service profit | client profit | user satisfaction |
|---|---|---|---|---|---|---|---|
| **Exp201** | 62.85 | 70.45 | 2.74 | 1.07E-04 | 2.97 | 89.70 | 82.13% |
| **Exp202** | 62.88 | 69.49 | 2.74 | 1.06E-04 | 2.94 | 89.70 | 82.44% |
| **Exp203** | 62.85 | 68.94 | 2.73 | 1.02E-04 | 2.73 | 89.80 | 82.64% |
| **Exp204** | 62.95 | 69.31 | 2.74 | 1.05E-04 | 2.88 | 89.70 | 82.51% |
| **Exp205** | 62.95 | 70.94 | 2.74 | 1.04E-04 | 2.81 | 89.72 | 81.99% |
| **Average** | **62.90** | **69.83** | **2.74** | **1.05E-04** | **2.87** | **89.72** | **82.34%** |
| **St. Dev.** | **5.14E-02** | **8.34E-01** | **1.88E-03** | **1.94E-06** | **9.83E-02** | **4.21E-02** | **2.71E-03** |

TABLE 4
Sensitivity test 2: Objective weights.

| Experiment | 401 | 402 | 403 | 404 | 405 | 406 | 407 | 408 | 409 | 410 | 411 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{w_S}$ | 0.50 | 0.40 | 0.33 | 0.20 | 0.15 | 0.14 | 0.13 | 0.12 | 0.11 | 0.10 | 0.00 |
| $\mathbf{w_A}$ | 0.50 | 0.40 | 0.33 | 0.20 | 0.15 | 0.14 | 0.13 | 0.12 | 0.11 | 0.10 | 0.00 |
| $\mathbf{w_U}$ | 0.00 | 0.20 | 0.34 | 0.60 | 0.70 | 0.71 | 0.72 | 0.73 | 0.74 | 0.75 | 1.00 |
| **service profit** | 21.57 | 21.65 | 21.76 | 22.10 | 22.35 | 22.47 | 22.64 | 22.85 | 23.28 | 23.42 | 22.73 |
| **client profit** | 69.18 | 69.06 | 68.57 | 66.77 | 64.73 | 64.22 | 61.46 | 57.10 | 32.74 | 6.05 | 6.68 |
| **user satisfaction** | 82.97% | 83.10% | 83.04% | 83.51% | 83.57% | 83.63% | 83.77% | 83.57% | 88.04% | 92.99% | 93.13% |

TABLE 5
Sensitivity test 3: User satisfaction coefficients.

| Experiment | 501 | 502 | 503 | 504 | 505 | 506 | 507 |
|---|---|---|---|---|---|---|---|
| **a** | 1.00 | 0.75 | 0.67 | 0.50 | 0.33 | 0.25 | 0.00 |
| **b** | 0.00 | 0.25 | 0.33 | 0.50 | 0.67 | 0.75 | 1.00 |
| **Sum of VMs** | 3705 | 3715 | 3695 | 3707 | 3705 | 3700 | 3820 |
| **CPU** | 61.14 | 61.01 | 61.27 | 61.10 | 61.15 | 61.23 | 59.22 |
| **response time** | 66.42 | 66.43 | 66.87 | 66.40 | 66.51 | 66.57 | 64.33 |
| **ads** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| **service price** | 4.83E-04 | 4.81E-04 | 4.80E-04 | 4.79E-04 | 4.84E-04 | 4.85E-04 | 2.81E-04 |
| **service profit** | 22.54 | 22.40 | 22.34 | 22.24 | 22.56 | 22.60 | 12.06 |
| **client profit** | 6.86 | 7.01 | 7.06 | 7.17 | 6.84 | 6.80 | 17.33 |
| **user satisfaction** | 100.00% | 96.54% | 95.34% | 93.10% | 90.70% | 89.56% | 87.53% |

may dictate the relationship between providers and clients, the target market segments and competition. The authors discuss different pricing models for web services including the freemium model, pay-per-use and the ad-based model. In our work, we make use of some of this models, including pay-per-use for cloud resources and the ad model for web applications.

The relationship between a producing party (provider) and a consuming party (client) is a prevalent concept in many economic and business fields. More specifically, in the field of operations management, the relationship between the provider and the client is a special case of a supply-chain relationship, where we have the provider of an input interacting with a firm using that input in the production process [18]. In the field of marketing, the relationship is referred to as the channel of distribution [19]. In both of these fields, there is an external relationship between the upstream supplier/provider and the downstream producer/client.

Cloud economics is a prevalent aspect of cloud computing, given that this is its appeal [20]; the commoditization of computation resources. Cloud computing has been recognized as an economy of scale at least from the infrastructure provider's point of view [21], [22], with particular benefits for startups or small companies [23], [24] and in many cases cloud economics have been used as a decision criterion between private or public clouds [25]. Cloud economics, again mainly from a cost perspective, have also been considered for mitigating distributed denial-of-service attacks on cloud deployed web applications [12], [26], [27]. Economic models in cloud computing have also been employed for pricing cloud resources [28], service composition [29], and capacity sharing in federated clouds [30]. Armbrust et al. [2] claim that the "economy-of-scale" property is the main appeal for using a public cloud instead of setting up a private cloud, which is materialized in the "pay-as-you-go" pricing scheme for cloud consumers. The authors also discuss the economic aspects of scaling with respect to the effects of traffic fluctuation, underprovisioning and overprovisioning, considering also the long-term business effects (e.g., loss of clients).

In cloud elasticity, a number of research works have considered the optimization of cost in scaling actions. Cost has been employed in scaling either as a goal to be minimized [31] or by considering its trade-offs with performance (i.e., sacrifice performance to minimize cost or vice versa) [32] or in the context of task scheduling where scaling occurs under budget and deadline constraints [33]. Nevertheless, there is no consideration for the revenue generation of the application in any of these works.

Chen et al. [9] also address the problem of scheduling service requests in cloud resources, but take into account the cost of resources, prices and revenue generation, and client satisfaction, where the last has a negative relationship with price. The approach is in the same vein as ours, by combining performance maintenance with profit optimization, but differs in a number of points. It adapts the service price and revenues solely based on client satisfaction, without any other constraint. In addition, it considers a static demand, while in our experiments we have considered dynamic and fluctuating incoming traffic. However, our definition of user satisfaction was inspired by this work.

Shtern et al. [34] consider both benefit and cost in a quality metric, called *cloud efficiency*, to evaluate how efficient applications use cloud resources. While the cost function is detailed based on information provided by the cloud provider or gathered by the application developer, the benefit function is rather simplistic and its details are generally left to the application developer. In our work, we present a more complete revenue function, which may be limited with respect to its scope, but it has the ability to monetize benefit and directly compare it with cost.

When service-oriented architectures gained prominence and popularity, their organization in ecosystems with multiple components and different stakeholders also attracted the attention of researchers [35], [36]. Additionally, researchers also recognized the business dimension of these ecosystems and how value flows between the partners [37]. More specifically, Fokaefs et al. [38], [39] have explored the impact of technical decisions on economic parameters of the software and how the latter can be used to complement these decisions and optimize the business position of the service provider. As far as this work is concerned, the structure of the web software ecosystem and the respective economic model were originally proposed in [40]. The difference in this work is that we have one more party, who can make decisions, the application provider and one party, who provide an additional optimization objective, the end-user. Additionally, in another previous work [1], we have proposed a similar hybrid (performance and economics) adaptation method which can both scale the cloud infrastructure of the web service and its price per request. The premise of the method is based on the economy-of-scale nature of cloud computing; it becomes cheaper on average to serve more requests as the total cost for VMs is spread out to more requests. The algorithm then adapts the service price accordingly. In either of the previous works, we addressed the problem only from the service provider's perspective without constraints from clients or end-users. In this work, the proposed method operates in an extended ecosystem and can adapt other stakeholders' parameters as well.

DevOps [4] can be considered as a novel development process or model or paradigm or even philosophy. Its primary goal is to bridge the gap between development and operation management. In this capacity, it recommends the use of tools and processes, as well as knowledge and skill sets, which would span across the entire lifecycle of the software. The concept existed, even before it was named DevOps [5]; developers would tinker with system administration tools and concepts to better understand how the software is to be deployed, while IT operators would occasionally merge with the development team to better understand the system's functionality and ensure higher quality. Thanks to virtualization technologies and the self-healing and self-adaptation capabilities of modern system, DevOps is the beginning of a new software development culture and a new breed of hybrid developers and IT specialists.

Another mission of the DevOps model is to shorten the release cycle of the software [41]. The goal is for every change to be incorporated seamlessly to the system in production, while high quality is ensured and maintained [42]. Netflix is already employing *chaos engineering* techniques [43], which, in the context of DevOps, enables concepts such as *continuous deployment* and *continuous delivery*.

[44] present their experience in migrating a monolithic mobile back end as a service (MBAAS) to a microservice architecture. Apart from the common pains and suffering of migrating a legacy system to a service-oriented architectures, they had some interesting experience with respect to the DevOps side of the process and the final system. First, they had to change the team organization from a horizontal structure (development, QA, operations) to more vertical teams with all layers responsible for the smaller services. Second, they report on the importance of monitoring the system and, finally, on the use of containers to bridge the gap between the development and the production phases.

## 6 Conclusion

In this work, we presented a novel adaptation method for cloud-deployed web software systems. The method assumes an extended software ecosystem with four types of independent stakeholders, cloud providers, web service providers, web application providers, and end-users, with individual interests and conflicting goals. These goals are materialized as objectives, service profit, client profit, and user satisfaction, in a multi-objective optimization problem. The three objectives constrain each other, which is why a global optimal solution may not be possible to be found. The output of the algorithm is a set of Pareto optimal solutions, on which we will have to apply an external optimization objective. Our experiments have shown that such an objective, which tries to optimize the welfare of the ecosystem as a whole, will lead the stakeholders to better positions than if any other of their partners tried to unilaterally optimize their individual objectives. Our experiments have also showed that our analysis can be used to identify opportunities for collaboration and in general assess the economic situation of the ecosystem. Further sensitivity tests have shown that our method can be trained in terms of weights and coefficients according to different goals and situations.

Our approach is novel in the sense that it proposes both traditional scaling actions to adapt a service's cloud infrastructure, as well as adaptive actions for economic parameters, like prices and advertisements. We believe that such a method, which combines the economic and technical dimensions of a cloud ecosystem, will eventually assist developers to achieve sustainable scaling, it will create stronger business bonds between stakeholders and will reduce the entrance barriers to software-centric provider-client ecosystems. In the near future, we intend to study and apply our method in a vertically extended ecosystems with multiple providers and clients, and see how any decisions, scaling and adaptive actions are affected in the presence of competition and collaboration.

## Acknowledgments

This research was supported by IBM Centres for Advanced Studies (CAS), the Natural Sciences and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network and the Ontario Research Fund for Research Excellence under the Connected Vehicles and Smart Transportation(CVST) project.

## References

[1] M. Fokaefs, C. Barna, and M. Litoiu, "Economics-Driven Resource Scalability on the Cloud," in *11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Austin, TX, USA: IEEE, May 2016, pp. (129–139).

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[3] J. Silvestre, "economies and diseconomies of scale," in *The New Palgrave: A Dictionary of Economics*, J. Eatwell, M. Milgate, and P. Newman, Eds. Basingstoke: Palgrave Macmillan, 1987.

[4] M. Hüttermann, *DevOps for developers*. Apress, 2012.

[5] D. Spinellis, "Being a devops developer," *IEEE Software*, vol. 33, no. 3, pp. 4–5, 2016.

[6] K. Lyons, C. Playford, P. Messinger, R. Niu, and E. Stroulia, "Business Models in Emerging Online Services," in *Value Creation in E-Business Management*, ser. Lecture Notes in Business Information Processing, M. Nelson, M. Shaw, and T. Strader, Eds. Springer Berlin Heidelberg, 2009, vol. 36, pp. 44–55.

[7] M. Sahajwani, "3 Ways Apps Make Money," http://www.investopedia.com/financial-edge/0112/3-ways-apps-make-money.aspx, 2012, accessed: 2015-10-24.

[8] M. Litoiu and C. Barna, "A performance evaluation framework for web applications," *Journal of Software: Evolution and Process*, vol. 25, no. 8, pp. 871–890, 2013.

[9] J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya, "Tradeoffs between profit and customer satisfaction for service provisioning in the cloud," in *Proceedings of the 20th international symposium on High performance distributed computing*. ACM, 2011, pp. 229–238.

[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[11] IBM, "An architectural blueprint for autonomic computing," IBM, Tech. Rep., 2005. [Online]. Available: http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf

[12] C. Barna, M. Shtern, M. Smit, V. Tzerpos, and M. Litoiu, "Mitigating DoS attacks using performance model-driven adaptive algorithms," *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 1, pp. 3:1–3:26, Mar. 2014.

[13] C. Barna, H. Ghanbari, M. Litoiu, and M. Shtern, "Hogna: A platform for self-adaptive applications in cloud environments," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2015 IEEE/ACM 10th International Symposium on*, May 2015, pp. 83–87.

[14] M. Litoiu, "Optimization, Performance Evaluation and Resource Allocator (OPERA)," 2013. [Online]. Available: http://www.ceraslabs.com/technologies/opera

[15] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice Hall, 1981.

[16] B. W. Boehm and K. J. Sullivan, "Software Economics: A Roadmap," in *Proceedings of the Conference on the Future of Software Engineering*. ACM Press, 2000, pp. 319–343.

[17] K. Lyons, P. R. Messinger, R. H. Niu, and E. Stroulia, "A Tale of Two Pricing Systems for Services," *Inf. Syst. E-bus. Manag.*, vol. 10, no. 1, pp. 19–42, Mar. 2012.

[18] A. Nagurney, *Supply Chain Network Economics: Dynamics of Prices, Flows, and Profits*. Edward Elgar Publishing, 2006.

[19] S. C. Choi, "Price Competition in a Channel Structure with a Common Retailer," *Marketing Science*, vol. 10, no. 4, pp. 271–296, 1991.

[20] H. Erdogmus, "Cloud computing: does nirvana hide behind the nebula?" *Software, IEEE*, vol. 26, no. 2, pp. 4–6, 2009.

[21] R. L. Grossman, "The case for cloud computing," *IT professional*, vol. 11, no. 2, pp. 23–27, 2009.

[22] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, "Cloud computing: Distributed internet computing for it and scientific research," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 10–13, 2009.

[23] M. Dave and Y. S. Shishodia, "Cloud economics: Vital force in structuring the future of cloud computing," in *Computing for Sustainable Global Development (INDIACom), 2014 International Conference on*. IEEE, 2014, pp. 61–66.

[24] K. Rafique, A. W. Tareen, M. Saeed, J. Wu, and S. S. Qureshi, "Cloud computing economics opportunities and challenges," in *Broadband Network and Multimedia Technology (IC-BNMT), 2011 4th IEEE International Conference on*. IEEE, 2011, pp. 401–406.

[25] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam, "To move or not to move: The economics of cloud computing," in *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*. USENIX Association, 2011, pp. 5–5.

[26] J. Idziorek and M. Tannian, "Exploiting cloud utility models for profit and ruin," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 33–40.

[27] M. Naresh Kumar, P. Sujatha, V. Kalva, R. Nagori, A. K. Katukojwala, and M. Kumar, "Mitigating economic denial of sustainability (edos) in cloud computing using in-cloud scrubber service," in *Computational Intelligence and Communication Networks (CICN), 2012 Fourth International Conference on*. IEEE, 2012, pp. 535–539.

[28] B. Sharma, R. K. Thulasiram, P. Thulasiraman, S. K. Garg, and R. Buyya, "Pricing cloud compute commodities: a novel financial economic model," in *Proceedings of the 2012 12th IEEE/ACM international symposium on cluster, cloud and grid computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 451–457.

[29] Z. Ye, A. Bouguettaya, and X. Zhou, "Economic model-driven cloud service composition," *ACM Transactions on Internet Technology (TOIT)*, vol. 14, no. 2-3, p. 20, 2014.

[30] N. Samaan, "A novel economic sharing model in a federation of selfish cloud providers," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 1, pp. 12–21, 2014.

[31] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, 2012, pp. 644–651.

[32] B. Suleiman, "Elasticity economics of cloud-based applications," in *Services Computing (SCC), 2012 IEEE Ninth International Conference on*. IEEE, 2012, pp. 694–695.

[33] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 49.

[34] M. Shtern, M. Smit, B. Simmons, and M. Litoiu, "A runtime cloud efficiency software quality metric," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 416–419.

[35] A. P. Barros, M. Dumas, and P. D. Bruza, "The move to web service ecosystems," *BPTrends*, vol. 3, no. 3, 2005.

[36] A. P. Barros and M. Dumas, "The rise of web service ecosystems," *IT professional*, vol. 8, no. 5, pp. 31–37, 2006.

[37] N. S. Caswell, C. Nikolaou, J. Sairamesh, M. Bitsaki, G. Koutras, and G. Iacovidis, "Estimating value in service systems: A case study of a repair service system," *IBM Systems Journal*, vol. 47, no. 1, pp. 87–100, 2008.

[38] M. Fokaefs and E. Stroulia, "Software evolution in web-service ecosystems: A game-theoretic model," *Service Science*, vol. 8, no. 1, pp. 1–18, 2016.

[39] M. Fokaefs, E. Stroulia, P. Messinger, I. Mistrik, R. Bahsoon, R. Kazman, K. Sullivan, and Y. Zhang, "Software evolution in the presence of externalities: A game-theoretic approach," pp. 243–258, 2013.

[40] M. Fokaefs, C. Barna, and M. Litoiu, "An Economic Model for Scaling Cloud Applications," in *9th IEEE International Conference on Cloud Computing*. San Francisco, CA, USA: IEEE, June 2016.

[41] L. Zhu, L. Bass, and G. Champlin-Scharff, "Devops and its practices," *IEEE Software*, vol. 33, no. 3, pp. 32–34, 2016.

[42] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[43] 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.

[44] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices architecture enables devops: Migration to a cloud-native architecture,” *IEEE Software*, vol. 33, no. 3, pp. 42–52, 2016.

**Marios Fokaefs** Marios Fokaefs is a postdoctoral fellow with the Adaptive Software Research Lab in the Department of Electrical Engineering and Computer Science, York University. He holds a MSc and a PhD in Software Engineering from the Department of Computing Science, University of Alberta. His research revolves around the problems of software evolution and change management with particular focus on service-oriented architectures and cloud systems. Additionally, his research looks into the relationship of technical decisions with economic considerations in software systems. Marios Fokaefs is an IEEE Member and former IBM Student Fellow, while he has also served in the program committees of IEEE ICSME, IEEE ICWS, IEEE CLOUD, IEEE MESOCA, IEEE ICSOC and CASCON among others.

**Cornel Barna** Cornel Barna received his Ph.D. from York University in 2015 under the supervision of Prof. Litoiu. Currently he collaborates with Prof. Litoiu’s team at the Adaptive Software Research Lab on big data projects involving cloud computing and micro-services. Cornel’s research interests include cloud computing, adaptive systems, big data, modeling, computational optimization, heuristics and meta-heuristics.

**Marin Litoiu** Marin Litoiu is Associate Professor in the Department of Electrical Engineering and Computer Science and in the School of Information Technology, York University. He leads the Adaptive Software Research Lab and focuses on making large software systems more versatile, resilient, energy-efficient, self-healing and self-optimizing. His research won many awards including the IBM Canada CAS Research Project of the Year Award and the IBM CAS Faculty Fellow of the Year Award for his “impact on IBM people, processes and technology.” Prior to joining York University, Dr. Litoiu was a Research Staff member with the Centre for Advanced Studies in the IBM Toronto Lab where he led the research programs in software engineering and autonomic computing. He received the Canada NSERC Synergy Award for Innovation in recognition for these collaborative university/industry activities. He was also recipient of the IBM Outstanding Technical Contribution Award for his research vision on Cloud Computing. Dr. Litoiu is one of the founders of the SEAMS Symposium seriesACM/IEEE Software Engineering for Adaptive and Self-Managing Systems and the Chair of the SEAMS Steering Committee. He has been the General Chair of SEAMS in 2013, and also serves on the steering committees of IEEE MESOCA and CASCON. Dr. Litoiu is also member of IEEE CS Conference Advisory Committee and Vice-chair of IEEE Technical Council on Software Engineering.