Elastic Admission Control for Federated Cloud Services

Kleopatra Konstanteli, Tommaso Cucinotta, Konstantinos Psychas, and Theodora Varvarigou

Abstract—This paper presents a technique for admission control of a set of horizontally scalable services, and their optimal placement, into a federated Cloud environment. In the proposed model, the focus is on hosting elastic services whose resource requirements may dynamically grow and shrink, depending on the dynamically varying number of users and patterns of requests. The request may also be partially accommodated in federated external providers, if needed or more convenient. In finding the optimum allocation, the presented mechanism uses a probabilistic optimization model, which takes into account eco-efficiency and cost, as well as affinity and anti-affinity rules possibly in place for the components that comprise the services. In addition to modelling and solving the exact optimization problem, we also introduce a heuristic solver that exhibits a reduced complexity and solving time. We show evaluation results for the proposed technique under various scenarios.

Index Terms—Admission control, elasticity, cloud computing, optimum resource allocation, cloud federation

1 Introduction

Nowadays, more and more distributed applications and services are being provided in the Cloud as a composition of components. For example, a web-based application typically includes three main components (a.k.a., *tiers*): a web server, an application server and a database back-end. Each component is deployed as a set of virtualized servers, i.e., Virtual Machines (VMs), referred to in the following also as component *replicas*, that are activated each time a request arrives from the end users, where a load-balancing logic usually ensures that the workload is spread as evenly as possible across the VMs of a component (see Figure 1).

The use of virtualization techniques allows for the seamless allocation and relocation of service components inside the Cloud. It also makes it easier to perform the process of *horizontal elasticity*, i.e., adding/removing extra VMs for each component during runtime to maintain a certain level of performance for the overall service when there are variations in the workload. For example, in a web-based service as mentioned above, each tier may be designed to be horizontally scalable to meet the computing requirements imposed by large and variable workloads. Furthermore, the potential use of federation agreements among Cloud providers allows for accepting a service with the possibility to hand it over (entirely or partially) to a federated provider (see Figure 2). This pushes forward the idealistic view of infinitely available resources for a Cloud Provider (CP).

In this context, a challenging problem is the one of how to properly design an intelligent resource management logic, that considers the multi-faceted aspects of the VM placement process, including possible quality of service (QoS) requirements to be met, uncertainty and variability in workload demand, and CP own business objectives. At admission control time, and in order to provide strong performance guarantees, the CP must consider not only the basic resource requirements but also the extra ones that may be needed to be added at runtime, defined as *elastic* requirements. In many cases,

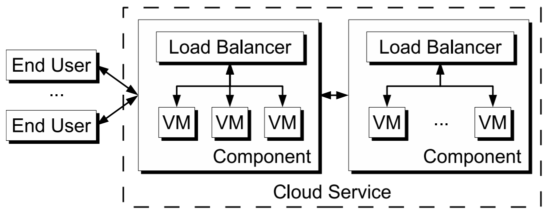


Figure 1. A sample service composed of multiple horizontally scalable components, each of them deployed as a set of Virtual Machines (VMs).

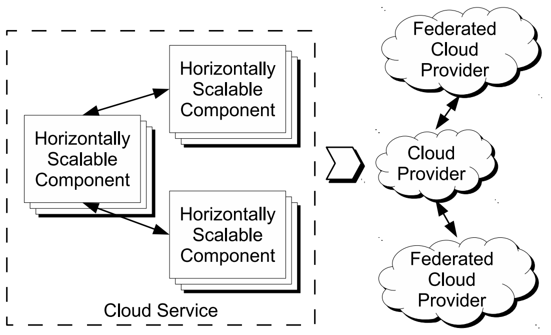

Horizontally Scalable Component
Horizontally Scalable Component
Horizontally Scalable Component
Cloud Service
Federated Cloud Provider
Cloud Provider
Federated Cloud Provider

Figure 2. Placement of a distributed cloud service onto a Cloud Provider with federation agreements.

• K. Konstanteli, K. Psychas and T. Varvarigou are with the Department of Electrical and Computer Engineering, National Technical University of Athens, Greece.
E-mail: {kkonst, kpsychas, dora}@mail.ntua.gr
• T. Cucinotta is with Bell Laboratories, Alcatel-Lucent, Dublin, Ireland.
E-mail: tommaso.cucinotta@alcatel-lucent.com

these may be quite large compared to the basic requirements. For example, given a service with a high expected variability in the number of users, the number of VMs that may be needed may significantly vary over time to meet the agreed quality of service (QoS) level. Therefore, the elastic requirements play a significant role in the cost for hosting the service. Still, with a probabilistic knowledge of the workload variability and flexible SLA models allowing for probabilistic guarantees, the CP and its clients may achieve an agreement that satisfies both parties to a sufficient extent. Also, when allocating a set of services within the infrastructure, a CP needs to optimize on proper metrics that express the goodness of the found allocation. Clearly, from the CP's perspective, such metrics are significant for the cost of each allocation solution. However, apart from the cost, nowadays a CP's business policy may include other factors such as how eco-efficient a given host is as compared to others [1].

Additionally to federated Clouds, the problem addressed in this paper is also relevant in the context of hybrid Clouds. An example from the telecommunications industry is the one of a network operator willing to deploy within its private cloud a virtualized network function [2], such as an IMS (IP Multimedia Subsystem) Core, which is composed of various components, e.g., Proxy, Interrogating and Serving Call Session Control Function (P-CSCF, I-CSCF, S-CSCF), and Home Subscribers Server (HSS). Said virtualized components are designed to be horizontally scalable to cope with variability in the number of subscribers, as well as in the workload imposed by a set of subscribers. The network operator might be willing to deploy a subset of the functions onto external Clouds to cope with particularly high peaks of demand that cannot be hosted internally.

In this paper, we tackle the problem of optimum allocation of distributed services in a federated cloud environment focusing on elastic workload requirements and incorporating a probabilistic approach in terms of availability guarantees. The resulting model constitutes a probabilistic admission control test that strives to optimize the allocation of the services within a CP (own or federated) infrastructure by considering its business level objectives (e.g., eco-efficiency and cost), while also sustaining a given QoS level for the clients and adhering to any affinity or anti-affinity constraints imposed on the services. The relative significance of optimization objectives can be regulated by the CP, and the proposed model can be easily extented to cover more objectives.

The output of the optimization problem includes the allocation pattern, i.e., the selected hosts and subnets for hosting the services, and the maximum amount of computational, networking and storage capacity that is allocated to each service component, either locally or in any of the federated Clouds. A heuristic solver is proposed for the presented problem that exhibits reduced computation times, when compared to an optimum solver (e.g., from tens or hundreds of seconds down to tens or hundreds of milliseconds, for the configurations shown in Section 6.1). This allows for applying the solver at run-time to scale up and down the allocated resources according to the demand, if needed.

This paper is organized as follows. Section 2 gives an overview of the related work, followed by the description of the problem under study in Section 3. This is formalized as a probabilistic admission control problem in Section 4, whereas Section 5 presents an approximate heuristic solver that yields high quality results at much higher speeds than those of an exact solver, as demonstrated in Section 6. An indicative case study that validates the proposed approach is presented in Section 7. Finally, conclusions are drawn in Section 8.

2 Related work

Several works that address the problem of optimal allocation of services in Cloud systems have appeared in recent years. Van den Bossche et al. [3], examine this problem in a multi-provider hybrid Cloud setting against deadline-constrained applications. A mixed integer optimization problem is formulated with the objective to minimize the cost of outsourcing tasks from data centers to external Clouds, while maximizing the internal utilization of the data centers. Macuzzo et al. [4], addressed the problem of optimal allocation for maximizing the revenues of Cloud providers by minimizing the amount of consumed electricity. Another green approach to resource allocation in Clouds can be found in the work of Xiao et al. [5]. This work introduces the concept of "skewness" as a way to measure the unevenness of the Cloud servers' utilization and formulate heuristics that try to minimize it while saving energy at the same time.

Sheng et al. [6] propose a method for proportionally scaling the resources among running tasks according to their demand, aiming to reduce their execution times. This technique targets self-organizing clouds that consist of desktop computers connected over the Internet in a best effort manner, and neglects SLA-violation aspects. The latter is the focus of the work of Eyraud-Dubois et al. [7], where SLA violations are considered when maximizing the resource allocation ratio. Service level objectives are also considered by Padala et al. [8], where the authors present a model for migrating VMs from overloaded nodes to available ones to avoid violations.

Chang et al. [9], formulated a resource allocation problem in which later tasks can reuse resources released by earlier tasks, and solved it with an approximation algorithm that can yield close to optimum solutions in polynomial time. In [10], the same authors propose an allocation problem for vTelco applications, where arbitrary latency expressions are used to model the end-to-end latency requirements of services to be placed. Another work focusing on placement of Telecom services can be found in An et al. [11], where trade-offs between centralized versus distributed cloud architectures are investigated. Alicherry et al. [12], tackled the problem of optimal placement of VMs in Clouds for minimizing latency. Complexity is reduced by recurring to a hierarchical split of the placement problem into the 2 reduced-complexity sub-problems of choosing the data centers in which to place, then choosing the specific racks and servers, and applying a partitioning of the application graph to be placed. Jiang et al. [13], focused on the problem of joint VM placement and route selection in the context of a multi-path enabled data center where traffic engineering techniques can be leveraged to arbitrarily route individual VM traffic. Also, the authors address migration of a minimum number of VMs for re-optimizing the infrastructure at each new VM acceptance or termination. Agarwal et al. [14], made a comparison of various algorithms for placing VMs in centralized vs distributed clouds, analyzing also the impact on user's latency for accessing the placed services. However, fluctuation of workload is neglected in the above works. We address it with a probabilistic approach.

Zhang et al. [15] investigated on probabilistic allocation of distributed services. In our prior work, the problem of optimum allocation of real-time workflows with probabilistic deadline guarantees was tackled [16]. In that work the main focus was on the probabilistic framework allowing the provider to overbook resources in the various time frames of each advance reservation request, knowing the probabilities of actual usage/activation of those services by the users. In the present paper, instead, the focus is on the problem of hosting elastic services whose workload may dynamically grow and shrink.

A way to estimate the probability that an end-to-end deadline is respected for a given composition of services is the one to build probabilistic models for the performance achieved by a composition of distributed services. For example, Zheng et al. [17], among others, investigated on mathematical models to compute the probability density function of the response-time of service compositions under various compositional patterns. Also, Zhang et al. [18] modelled probabilistic interactions among multiple components of multi-tier cloud applications recurring to Markov Chains. However, in the present paper we focus on variability of the workload for services whose composition is deterministically fixed.

Further works exist in the area of VM allocation for Cloud, Grid and Distributed computing, as witnessed by the interesting comparison among 18 algorithms carried out by Mills et al. [19], or the interesting architectural concepts highlighted by Xu et al. [20], where also a VM placement algorithm is introduced based on stable matching theory, or in the work of Maurer et al. [21], where a knowledge management based approach is proposed for dynamic optimization of cloud infrastructures.

Compared to our prior preliminary work [22] on VM allocation for elastic cloud services, this paper proposes major enhancements to the way the problem is formulated, introduces a novel heuristic to solve the problem, and provides evaluation results validating the technique.

3 Background and Notation

The problem under study is largely inspired by the OPTIMIS [23] and S(o)OS[1] EU Projects: a CP is a business entity that owns a set of physical hosts with potentially heterogeneous characteristics in terms of processing speed, architecture, and underlying network capabilities, and establishes SLAs with clients for hosting distributed services over a period of time. Each service is composed of components that are horizontally scalable, i.e., they can be deployed as a number of VMs spread across different cores, processors, hosts and subnets.

The CP books in advance physical resources for hosting the virtualized components of the service. Each component is characterized by specific resource requirements, as explained in Section 3.2. The VMs of the components can be associated with affinity and anti-affinity rules, which offer the client the ability to automatically keep VMs together on the same host, network or cloud, or apart on different physical elements, depending on the specific performance and reliability requirements of the encapsulated components. For example, affinity rules on host or network level allow for respecting tight response-time constraints and reduce the workload imposed on physical network elements. Anti-affinity rules are useful when components should run on separated hardware to achieve high resiliency or for security/privacy purposes. For example, best practises require that primary and secondary DNS servers are kept apart on different hosts or subnets. In other cases, the VMs of a service are meant to serve users that are geographically dispersed. Replicas[2] should be placed close to the users. These geographically dispersed resources could be owned by the same CP or acquired through the establishment of a federation with another CP.

According to the expected usage of the service, lower and upper limits to the requirements of each component can be specified, corresponding to the basic and elastic resources that may be needed at runtime.

3.1 Cloud federation model

In case a service cannot be fully accommodated by the CP, it may want to explore the option of federating with other CPs (unless forbidden by affinity rules), before rejecting the service. Also, there may be cases whereby the CP is forced to federate, i.e., there are anti-affinity rules in place specifying that only part of the service and/or component should be admitted into the CP's cloud, whereas the other part should be allocated on another CP's cloud. In any case, the CP may consider to federate either entire components or part of the components, i.e., allocate some of their required VM instances locally, and consider federating the rest of them.

1. More information is available at: http://www.soos-project.eu/ and http://www.optimis-project.eu/.

2. The term 'replica' is used henceforth to refer to the replicated VMs encapsulating a specific horizontally scalable component of the service under examination.

It is assumed that the CP has already established contractual agreements with one or more CPs following the Federated Shared Pool Option-Enabled Policy (FSPO), following the federation model of Toosi et al. [24]. This policy allows the CP to overcome the resources limitation that may occur at negotiation time by acquiring resources from other CPs. Resource prices are included in the agreement and are dependant on the VM characteristics. Usually, when a resource is rented in the context of a federation, its selling price is lower compared to the one that the CPs offer their own customers directly. This price list may periodically be updated according to the terms of the agreement, and thus there is no need to query the CPs for such information during the negotiation process. To be always cost-effective, the federation cost is calculated using the cheapest available resources in the federated shared pool.

Consequently, if the CP is not able to serve a request locally, then part of it is outsourced to another CP at the price defined in the pre-established federation agreement. Thus, the federation cost that is awarded to the CP with whom the federation is established, can be calculated at admission control time based on its pre-agreed price, and is subtracted from the actual gain of the original CP. The latter is based on the pricing scheme of the original CP, as advertised to its clients. This way, the establishment of federation remains transparent to the customers and is not reflected in the cost. Finally, in case none of the federated CPs can at that time provide the required resources for any reason, this will lead the CP that has received the original request to reject it or make a counter offer depending on its business policy.

3.2 Performance Model

In this work, it is assumed that the computing, networking and storage capabilities of each physical host, as well as those of the services, may be expressed in terms of a single performance metric. For example, across a set of hosts with similar capabilities in terms of the Instruction-Set Architecture (ISA) of the CPUs, the computing capabilities may be approximated in terms of instructions per second that each host can process, accounting for the different clock speeds and number of CPUs and cores available in each one of them. For example, a host with a single 1 GHz CPU would have a computing capability of 10^9, while a quad-core 3 GHz host would have a computing capability of 12×10^9. Similarly, we would roughly say that a service under given operating performance conditions (i.e. exhibited response-time within the desired range, under a workload following the expected pattern) might require 3×10^9 instructions per second. This would imply that the service should be hosted either replicated over 3 of the mentioned single-CPU machines, or simply as a single instance occupying one of the CPUs of the mentioned quad-core system.

Alternatively, the performance metric might be defined in terms of the performance achieved by a given benchmark (e.g. LINPACK[3]) that is relevant for the kind of applications that may potentially be hosted. A more precise performance model could consider a vector of metrics, e.g., as coming out of a number of heterogeneous benchmarks (e.g. linear algebra, graphics, integer and floating-point operations, etc.). However, in the present work, a single metric is used for the sake of simplicity, and its values are supposed to vary in a range of positive real numbers. As implied by the just mentioned example, we assume an ideal model of software scalability, in which *each service can arbitrarily be decomposed in a number of possibly imbalanced replicas, running over possibly heterogeneous hosts*. In Cloud environments, this is possible thanks to virtualization technologies by which more and more VMs hosting replicas of a service can be instantiated. Clearly, the whole performance of a service is given by the sum of the performance of the decomposed replicas.

Note that this assumption is easily verified in typical Cloud computing services in which both vertical and horizontal scalability may be applied to deal with a large number of users potentially accessing the services. In such a case, the multitude of replicas each service instantiates serves requests on behalf of different (groups of) users, operating essentially in isolation from the others. When allocating multi-resource workloads we assume that the splits in the various resources capacities vary proportionally to each other, i.e., when hosting half of the computational workload of a component in a replica, we expect half of its networking traffic to be generated by that replica as well, as detailed in Section 4. Also, differently from a prior version of this work [22], here we extend the performance model to consider communication and synchronization overheads due to the splitting (detailed in Section 4.2). Furthermore, we assume that any additional overheads due to the interference among VMs allocated on the same hosts are already accounted in the abstract resource requirements of the service. This seems reasonable in this context as we deal with horizontally scalable services that can span across various physical nodes, when needed (as opposed to many small services that can be consolidated on the same host). However, investigations on how to extend the model with a more informed inter-VM interference overhead model are reserved for future work. Note also that the impact of the inter-VM temporal interferences can be kept under control using proper soft real-time schedulers in the hypervisor, as done in our prior works [25], [26].

3.3 Resources Topology

The provider's resources may generally be considered as an interconnection of (potentially heterogeneous) networks that interconnect (potentially heterogeneous) computing nodes. For example, various LANs enclosing

3. More information is available at: http://www.netlib.org/linpack/.

multi-processor computing nodes are interconnected by means of one or more WANs. Therefore, the network topology is characterized by:

- A set of *computing nodes*, or *hosts*: $\mathcal{H} = \{1, \ldots, N_{\mathcal{H}}\}$. Each host $j \in \mathcal{H}$ is characterized by an available computing capacity $U_j \in \mathbb{R}^+$, which expresses the value of a given system-wide reference performance metrics (see Section 3.2), and an available storage capacity $M_j \in \mathbb{N}$, expressed in bytes.
- A set of available *subnets*: $\mathcal{N} = \{1, \ldots, N_{\mathcal{N}}\}$. Each subnet $n \in \mathcal{N}$ is characterized by a maximum aggregate bandwidth $W_n \in \mathbb{R}^+$, expressed in bytes/s.
- The network *topology*, specifying which hosts $\mathcal{H}_n \subset \mathcal{H}$ are connected to each subnet $n \in \mathcal{N}$.

Also, for each hosts pair $j_1, j_2 \in \mathcal{H}$, we denote by $P_{j_1, j_2} \subset \mathcal{N}$ the set of networks to be traversed from j_1 to j_2, according to the available routing information. Similarly to what done in [16], each host is associated in the model with a logical "loopback" subnet useful to model communications among different components possibly deployed on the same host. In order to properly deal with federation, we introduce in the physical topology notation a set of *special hosts* $\mathcal{H}_F \subset \mathcal{H}$ representing federated providers. Each special host $j \in \mathcal{H}_F$ is characterized by (big) aggregated computing and storage capacity limits U_j, M_j. If these are unknown, then the corresponding capacity equations below can be omitted. Also, each of them would be placed throughout the topology in a place that is representative of the network path necessary to reach the federated provider.

3.4 Services Notation

The following notation is used to refer to services:

- Set of *service instances* (referred to simply as *services* from here on): $\mathcal{S} = \{1, \ldots, N_S\}$.
- Each service $s \in \mathcal{S}$ is a linear workflow of m^s *components* (encapsulated inside VMs): $\mathcal{S}^s \triangleq \{\xi^s_1, \ldots, \xi^s_{m^s}\}$.

Each component $\xi^s_i \in \mathcal{S}^s$ has horizontal scalability capabilities, i.e., it can replicate as multiple VMs to be deployed on multiple hosts, and it is characterized by:

- minimum computing, network and storage requirements θ^s_i, b^s_i and ψ^s_i needed by ξ^s_i for basic operation;
- maximum computing, network and storage requirements Θ^s_i, B^s_i and Ψ^s_i, also called *elastic requirements*, that ξ^s_i may fruitfully exploit; maxima are useful for modeling services with intrinsic scale-up limitations, or for accounting for maximum per-customer quota of the provider (e.g., Amazon 20 VMs per availability zone), or also for accounting for budget restrictions of customers.

When splitting a component across multiple replicas, we may have a minimum amount of resources capacity below which it is not worth to have a replica, i.e.:

- minimum computing ϱ^s_i, network ν^s_i and storage ζ^s_i requirements allocatable to each instance; these easily map to the characteristics of the minimum VM "size" as offered by typical cloud providers.

For multi-resource components with non-negligible resource requirements, we may *optionally* assume them to be linearly interdependent with each other. Indeed, for web-based systems, it often happens that the processing, networking and disk access work-loads be proportionally dependent on the number of submitted requests per second. Some times the maximum memory occupation follows also the same dependency. Furthermore, services with significant network/disk traffic need a proportionate CPU allocation to properly handle it.

On a related note, the small, medium and large sizes in which Amazon "m1.*" instances can be rented, correspond to a nearly linear inter-dependence among CPU, RAM and disk capacity[4]. Introducing these optional constraints whenever possible allows us to reduce the number of independent variables, thus makes the presented problem simpler. However, these constraints can be left out for other cases in which such assumptions would not hold. Concerning storage requirements, we consider explicitly only one type of storage requirement, which may equally represent the amount of needed RAM or local HD storage. The model can easily be extended to consider both of them (not shown for the sake of brevity). The notation used throughout the paper is conveniently summarized in Appendix I.

3.5 Service Level Agreement Model

As already described in Section 3, the CP establishes SLAs with the clients for hosting their services over a period of time. The SLA for a given service $s \in \mathcal{S}$ carries the following parameters:

- The description of the service workflow $\mathcal{S}^s$, which must be complemented by the requirements of each component ξ^s_i: θ^s_i, Θ^s_i, b^s_i, B^s_i, ψ^s_i, and Ψ^s_i.
- A minimum probability ϕ^s that the required resources are actually available when the request arrives, namely that there are sufficient resources for the activation of the VMs when needed (useful when overbooking, see Section 4.5 for details).
- Gain $\mathcal{G}^s$ for the CP in case the service is accepted.
- Penalty P^s for the CP if the service fails to meet its QoS restrictions.

Furthermore, it is assumed that the CP has means to estimate the eco-efficiency E_j of each host $j \in \mathcal{H}$. The term eco-efficiency refers to consuming less energy to produce the same amount of useful output, and has a positive meaning, i.e. the higher its value, the more positive its effect is (interested readers may refer to [27] for further details on how to compute it).

4 Problem Formulation

Using the definitions in Section 3, we will now formalize the problem under study, hereinafter referred to as the

4. Compare the ECU, Memory and Storage columns of the table at: http://aws.amazon.com/ec2/instance-types/#instance-details.

Admission Control Optimization Problem (ACOP). Note that we introduce a wide set of possible constraints, but in some real-life problem instances, it may be possible that some of these are not needed and can be omitted.

4.1 Variables

First of all, let us introduce the unknown variables.

The locally allocated computing, network and storage capacity for the components on the hosts are denoted by $x^s_{i,j} \in \mathbb{R}^+$, $y^s_{i,j} \in \mathbb{R}^+$ and $z^s_{i,j} \in \mathbb{R}^+$ respectively, with $s \in \mathcal{S}$, $i \in S^s$, $j \in \mathcal{H}$. If a component ξ^s_i is not given any computing capacity on a host $j \in \mathcal{H}$, then $x^s_{i,j} = y^s_{i,j} = z^s_{i,j} = 0$. The allocated elastic computing, network and storage capacity for each component is $\sum_{j\in\mathcal{H}} x^s_{i,j} - \theta^s_i$, $\sum_{j\in\mathcal{H}} y^s_{i,j} - b^s_i$, and $\sum_{j\in\mathcal{H}} z^s_{i,j} - \psi^s_i$, respectively.

As components may be deployed on federated providers, we introduce, $\forall s \in \mathcal{S}$, $\forall i \in S^s$, the derivative variables $fx^s_i \in \mathbb{R}^+$, $fy^s_i \in \mathbb{R}^+$ and $fz^s_i \in \mathbb{R}^+$ representing the federated computing, network and storage capacity for the components: $fx^s_i = \sum_{j\in\mathcal{H}_F} x^s_{i,j}$, $fy^s_i = \sum_{j\in\mathcal{H}_F} y^s_{i,j}$, $fz^s_i = \sum_{j\in\mathcal{H}_F} z^s_{i,j}$. If a component ξ^s_i is not federated, then $fx^s_i = fy^s_i = fz^s_i = 0$. Thus, the overall accepted computing, network and storage capacity for a component are $x^s_i = \sum_{j\in\mathcal{H}} x^s_{i,j}$, $y^s_i = \sum_{j\in\mathcal{H}} y^s_{i,j}$, and $z^s_i = \sum_{j\in\mathcal{H}} z^s_{i,j}$, respectively.

Note that the capacity to be considered for federation is allocated roughly to the special hosts $\mathcal{H}_F \subset \mathcal{H}$ that represent the federated providers. These may use a similar admission control mechanism to the one described in this paper, or any other, to manage their resources. In order to accept a service by handing part of it over to a federated party, a CP has to consider trade-offs among availability, costs of its internal resources and extra costs imposed by the federated party, as it will be explained later in Section 4.3. Whenever the assumption of linear interdependency among computing, networking and storage requirements holds, some variables might be obtained as derivative of others, like in:

$$\begin{cases} y^s_{i,j} = & \alpha^s_i x^s_{i,j} + \beta^s_i \\ z^s_{i,j} = & \gamma^s_i x^s_{i,j} + \delta^s_i \end{cases} \tag{1}$$

for some component-dependent constants α^s_i, β^s_i, γ^s_i, δ^s_i that can be obtained by standard linear regression techniques. Note that the actual decomposition of each component ξ^s_i into VMs on the hosts $j \in \mathcal{H}$ where $x^s_{i,j} > 0$ is a lower-level detail that is not needed to be addressed in the formulated allocation problem. For example, a single (possibly multi-core) VM may be instantiated on each mentioned host for ξ^s_i, and allocation provided to various VMs may be guaranteed and isolated in a strong sense by employing proper real-time scheduling techniques [25].

4.2 Allocation Constraints

The overall computing, networking and storage capacity (allocated and federated) for each ξ^s_i should not exceed the limit defined by its maximum elastic requirements:

$$\begin{cases} \sum_{j\in\mathcal{H}} x^s_{i,j} \le \Theta^s_i \\ \sum_{j\in\mathcal{H}} y^s_{i,j} \le B^s_i \\ \sum_{j\in\mathcal{H}} z^s_{i,j} \le \Psi^s_i \end{cases} \quad \forall s \in \mathcal{S}, \forall i \in S^s. \tag{2}$$

The additional computing and storage requirements imposed on each host $j \in \mathcal{H}$ (including any federated provider $j \in \mathcal{H}_F$) cannot overcome the residual capacity:

$$\begin{cases} \sum_{s\in\mathcal{S}} \sum_{i\in S^s} x^s_{i,j} & \le & U_j \\ \sum_{s\in\mathcal{S}} \sum_{i\in S^s} z^s_{i,j} & \le & M_j \end{cases} \quad \forall j \in \mathcal{H} \tag{3}$$

Network constraints are formulated later in Eq. (7). Another possible upper constraint is the one restricting the overall expense for a customer, in cases in which the reserved resources have pre-specified unit costs. In such a case, the overall cost would easily be expressed as a linear combination of the $x^s_{i,j}$, $y^s_{i,j}$ and $z^s_{i,j}$ variables.

We introduce into the problem formulation some derivative Boolean variables that will shortly turn out to be useful: $\{\eta^s_{i,j}\}$, with a value of 1 if the component $i \in \mathcal{S}^s$ is given at least the minimum computing ϱ^s_i, networking ν^s_i and storage ζ^s_i requirements on host $j \in \mathcal{H}$ and 0 otherwise. These can be put in linear relationship with the $x^s_{i,j}$, $y^s_{i,j}$ and $z^s_{i,j}$ through the following constraints:

$$\begin{cases} x^s_{i,j} - \varrho^s_i & \ge & K(\eta^s_{i,j} - 1) \\ x^s_{i,j} & \le & K\eta^s_{i,j} \\ \mathit{similar\,for\,y\,and\,z} \end{cases} \quad \forall i \in S^s, \forall j \in \mathcal{H} \tag{4}$$

where K is a sufficiently large constant (refer to Appendix VI for details on its calculation). The above inequalities constrain the allocation variables $\{x^s_{i,j}, y^s_{i,j}, z^s_{i,j}\}$ to give enough capacity on each host j so as to meet the minimum splitting requirements $\{\varrho^s_i, \nu^s_i, \zeta^s_i\}$ for each component with $\eta^s_{i,j} = 1$, or alternatively they force them to be 0 with $\eta^s_{i,j} = 0$. For a proof, refer to Appendix IV.

Similarly, we introduce the derivative Booleans $\{\chi^s_{i,n}\}$ encoding whether or not for each component there is any replica allocated within each subnet:

$$\begin{cases} \sum_{j\in\mathcal{H}_n} x^s_{i,j} & \ge & K(\chi^s_{i,n} - 1) \\ \sum_{j\in\mathcal{H}_n} x^s_{i,j} & \le & K\chi^s_{i,n} \\ \mathit{similar\,for\,y\,and\,z} \end{cases} \quad \forall i \in S^s, \forall n \in \mathcal{N} \tag{5}$$

Finally, we introduce the derivative Booleans $\{\eta^s_i\}$ encoding whether for each component there is any allocated replica within the CP:

$$\begin{cases} \sum_{j\in\mathcal{H}} x^s_{i,j} - \theta^s_i & \ge & K(\eta^s_i - 1) \\ \sum_{j\in\mathcal{H}} x^s_{i,j} & \le & K\eta^s_i \\ \mathit{similar\,for\,y\,and\,z} \end{cases} \quad \forall i \in S^s, \forall j \in \mathcal{H} \tag{6}$$

The above equations also impose that the minimum overall allocations per component θ^s_i, b^s_i and ψ^s_i are respected, if the component is accepted. Finally, we introduce the derivative Boolean variables $\eta^s_{i_1,i_2,j_1,j_2} = \eta^s_{i_1,j_1} \wedge \eta^s_{i_2,j_2}$ encoding whether or not components i_1

and i_2 have, respectively, non-null deployed capacity on hosts j_1 and j_2. Appendix V shows how to introduce, via linear constraints, a Boolean variable forced to encode the logical AND between two or more Boolean variables. Then, the network capacity constraints can be stated as:

$$\sum_{\{j_1,\, j_2 \in \mathcal{H} | n \in P_{j_1,\, j_2}\}} \sum_{s \in \mathcal{S}} \sum_{i \in S^s \setminus \{\xi^s_{m^s}\}} \eta_{i,\, i+1,\, j_1,\, j_2} y^s_{i,\, j_1} \leq W_n, \ \forall n \in \mathcal{N} \tag{7}$$

This kind of constraint can also be used to model data flows in and out of the whole service workflow.

When horizontally scaling a service, it may happen that the scaled out version of the service needs additional resources as compared to the single-instance version, in order to operate correctly, due to additional synchronization and communication needed across its various instantiated replicas. A common model that accounts for such overheads is one [28] in which the overheads grow linearly with the number of replicas, but other models are possible as well [29]. The linear model may be considered by rewriting Eq. (4) after increasing the minimum required per-host resources ϱ^s_i, ν^s and ζ^s_i of a quantity that is proportional to the number of hosted replicas. All equations that serve as constraints, i.e., from (2) to (6), can be similarly reworked to account for synchronization and communication overheads (the exact formulation is omitted for the sake of brevity).

4.3 Partial admittance and federation

In case a service cannot be fully allocated within own premises, the CP may want to consider federating with other CPs, before rejecting the service. To model this, we introduce a further set of derivate variables:

- Booleans $\{\tau^s_i\}$ encoding whether for each component there is any capacity that should be federated:

$$\begin{cases} fx^s_i & \geq & K(\tau^s_i - 1) \\ fx^s_i & \leq & K \cdot \tau^s_i \end{cases} \quad \forall i \in S^s \tag{8}$$

- Booleans $\{\upsilon^s_i\}$ encoding whether for any component replica is allocated either within the CP or federated (i.e., υ^s_i is the logical OR between η^s_i and τ^s_i) :

$$\begin{cases} \upsilon^s_i & \leq & \eta^s_i + \tau^s_i \\ 2 \cdot \upsilon^s_i & \geq & \eta^s_i + \tau^s_i \end{cases} \quad \forall i \in S^s \tag{9}$$

  These are easily justified with the dual reasoning of the one reported in Appendix V.
- Ratio of the service $s \in \mathcal{S}$ that has been accepted whether locally or federated: $x^s = \frac{\sum_{i \in S^s} \upsilon^s_i}{m^s}$, where m^s is the number of components of service s.

When it comes to the cost of federation, it is assumed that it depends on the characteristics of the component, i.e. whether it is computational, network or storage intensive. To this direction, we assume that the federation cost of a component is a function of the requested resources $Fc(fx,\, fy,\, fz)$ and it can derive from a given price list that the CP uses to calculate the corresponding cost (please, refer to Appendix II for details).

4.4 Affinity and anti-affinity rules

Apart from the introduced allocation constraints, different types of services and components may require special treatment when it comes to their deployment. To this direction, an extra basic set of conditional affinity and anti-affinity constraints can be added to the allocation problem. For each service $s \in \mathcal{S}$, we allow for the specification of n^s_A affinity and anti-affinity constraints. Each (anti) affinity constraint $c \in \mathcal{C}^s_A = \{1, \ldots, n^s_A\}$ involves a subset $\sigma^s_c \subset \mathcal{S}^s$ of components, and is formally stated as described below.

In terms of affinity, the components ξ^s_i of the combination σ^s_c may be constrained to be allocated:

- *in the same physical node:*$\sum_{j \in \mathcal{H}} h^s_{\sigma^s_c,\, j} \leq 1$,
- *in the same subnet:* $\sum_{n \in \mathcal{N}} \delta^s_{\sigma^s_c,\, n} \leq 1$, where $\{h^s_{\sigma^s_c,\, j}\}$, $\{\delta^s_{\sigma^s_c,\, n}\}$ are Boolean variables that become 1, if host j or subnet n respectively is used in the allocation of at least one component i of combination σ^s_c.
- *in the same cloud:*$\forall i \in \sigma^s_c$, $\tau^s_i = \tau^s_{i+1}$.

The above rules can also cover only one component. In this case the rule affects the allocation of the instances of the specific component.

In terms of anti-affinity, the components ξ^s_i of the combination σ^s_c may be constrained to be allocated:

- *in different physical nodes:* $\forall j \in \mathcal{H}$, $\sum_{i \in \sigma^s_c} \eta^s_{i,\, j} \leq 1$,
- *in different subnets:* $\forall n \in \mathcal{N}$, $\sum_{i \in \sigma^s_c} \chi^s_{i,\, n} \leq 1$,
- *in different clouds:* $\sum_{i \in \sigma^s_c} \eta^s_i \leq 1$.

Anti-affinity rules can also be specified on the instance level of a given component, to regulate the distribution of its replicated VMs. Therefore, a component ξ^s_i must be replicated across at least:

- k^s_i *different nodes, if accepted:* $\sum_{j \in \mathcal{H}} \eta^s_{i,\, j} \geq k^s_i \eta^s_i$,
- l^s_i *different subnets, if accepted:* $\sum_{n \in \mathcal{N}} \chi^s_{i,\, n} \geq l^s_i \eta^s_i$.

4.5 Probabilistic Elasticity

In this section we propose a probabilistic approach to the problem of allocating extra resources for elasticity reasons. The basis of this approach lies in the existence of prediction models that are able to forecast resource usages using historical monitoring data. Indeed, there are several works in the literature, such as [30], that produce statistical information in the form of probability distributions of the resource requirements experienced at run-time by a service in virtualized environment. Given that such statistical knowledge is known, then it can be leveraged inside the problem to employ an over-allocation strategy ensuring the service can run flawlessly with at least a minimum probability ϕ^s (see Section 3.5). In the following, we focus on computing requirements only, but the line of reasoning extends easily to the consideration of network and disk requirements.

We assumed that the CP has knowledge about the probability that a given component may use a capacity up to x^s_i. This can be formally described by the cumulative distribution function $\mathcal{F}^s_i(x^s_i) = P\left[X^s_i \leq x^s_i\right]$ of the real-valued random variables X^s_i, representing the computational capacity that a component i may require at runtime. In order to deal simultaneously with all the components of the service, this can be generalized as: $\mathcal{F}^s(x_1^s, .., x_{m^s,}^s)$, where m^s is the number of the components of service s. Then, in order for a service s to be admitted into the system, instead of reserving resources for the maximum amount of elasticity requirements deterministically, it is sufficient to guarantee that the probability for service s to find enough available resources when actually required, denoted from now on as Φ^s, be subject to:

$$\Phi^s \geq \phi^s. \tag{10}$$

This reduces the resources that need to be booked for elasticity reasons, by exploiting statistical information, as demonstrated in Section 7. Note that, if $\phi^s = 1$, then the deterministic case is obtained as a particular case of the probabilistic one, i.e., the model will operate deterministically if the client asks for 100% guarantees in the SLA.

In order to formalize Φ^s, we propose two alternate models. When resources fluctuation of individual service components are independent of one another, the overall service availability is lower-bounded by the product of the individual service availabilities: $\Phi = P\left[X_1^s \leq x_1^s, .., X_m^s \leq x_{m^s}^s\right] \geq \prod_{i \in \mathcal{S}^s}[1 - (1 - \mathcal{F}_i^s(x_i^s))v_i^s]$, where v_i^s is the Boolean variable that becomes 1 when the component i is accepted, either allocated within the CP or federated (see Section 4.3). In some particular cases, the components of the service are tightly dependent on one another, therefore the service availability may be approximated as the availability of the least available component:

$$\begin{aligned} \Phi^s &= P\left[X_1^s \leq x_1^s, .., X_m^s \leq x_{m^s}^s\right] \\ &= min_{i \in \mathcal{S}^s}[1 - (1 - \mathcal{F}_i^s(x_i^s))v_i^s]. \end{aligned} \tag{11}$$

This formulation will concretely be applied in Section 7.

4.6 Objective Function

As already explained, the elastic capacity may be quite large compared to the basic one and thus it plays a significant role in the cost of hosting the service. Clearly, from a provider perspective, a metric of the goodness of an allocation may be significant of the additional costs possibly needed to admit the services. In order to formalize this, we introduce the extra cost ω_j associated with turning on an unused host $j \in \mathcal{H}_{off} \subset \mathcal{H}$. Then, a simple term to consider is the total additional cost $\mathcal{C}$ associated with turning on unused hosts:

$$\mathcal{C} = \sum_{j \in \mathcal{H}_{off}} h_j \cdot \omega_j, \tag{12}$$

where h_j are Booleans encoding whether or not there's any capacity allocated on host j. They may be defined as logical OR for all i of the $\eta_{i,j}$ Booleans through a set of linear constraints, as shown in Appendix V. Furthermore, it is in the CP's best interest to extend the optimization goal so that the eco-efficiency of the hosts E_j (see Section 3.5) is considered. To this direction, the overall objective is complemented by the term $\mathcal{E}$ that expresses the eco-efficiency score of the hosts that are used to form the allocation pattern:

$$\mathcal{E} = \sum_{j \in \mathcal{H}_{off}} h_j \cdot E_j. \tag{13}$$

The cost of federation for each component needs also to be considered as follows: $\mathcal{CF}^s = \sum_{i \in \mathcal{S}^s} Fc(fx_i^s, fy_i^s, fz_i^s)$. Also, the probabilistic framework, as introduced in Section 4.5, implies that with a maximum probability of $\overline{\Phi^s} \triangleq 1 - \Phi^s$, an admitted service is not expected to find the needed extra resources available, leading to the necessity to pay the penalty P^s back to the client. Therefore, for each service that is admitted into the system ($x^s > 0$), the expected penalty $\mathcal{P}^s = \overline{\Phi^s} P^s$, should be subtracted from the immediate gain $\mathcal{G}^s$. By taking into account all the different objectives mentioned above, we obtain the following multi-objective function:

$$\max \textstyle\sum_{s \in \mathcal{S}} x^s(\mathcal{G}^s - w_{\mathcal{P}}\mathcal{P}^s - w_{\mathcal{C}}\mathcal{CF}^s) - w_{\mathcal{C}}\mathcal{C} + w_{\mathcal{E}}\mathcal{E}, \tag{14}$$

where $w_{\mathcal{P}}$, $w_{\mathcal{C}}$, and $w_{\mathcal{E}}$ are used as weights for adapting the heterogeneous quantities in the sum and configuring their relative importance. For example, a profit-driven policy can be expressed by setting the weight $w_{\mathcal{E}}$ equal to zero. In this way, the eco-efficiency aspect is not considered in the optimization process, and the optimal solution will be the cheapest one. Also, given that the CP has no means to calculate the eco-efficiency of its hosts, this aspect can be easily unplugged by the model by simply setting the value of $w_{\mathcal{E}}$ to zero. Different acceptance policies can be applied by configuring these weights, as demonstrated later on in Section 7.

5 ACOP Heuristic Solver

The ACOP problem formalized in Section 4 falls within the class of Mixed-Integer Linear Programming (MILP) optimization problems in its simplest deterministic variant, or within the one of Mixed-Integer Non-Linear Programming (MINLP) when probabilistic availability constraints are in place. In addition to its precise mathematical formulation that is solved by global solvers with great complexity, we developed an approximation algorithm, namely the ACOP heuristic, that can yield near-optimal results but at much higher speeds than the optimal MINLP solvers. The basic steps of this heuristic approach are the following:

1) Get all valid combinations of the components for each service by taking into consideration the affinity or anti-affinity rules that are in place.
2) For each valid combination, calculate the gain, and the total resource capacity that it uses.
3) Iterate over all combinations of each service and find a possible allocation, if one exists. Store the

combinations that maximize gain for a given computing capacity.
4) Compute the cost of the stored allocations.
5) Output the allocation that maximizes the gain minus the cost.

Each step is detailed in the following sections.

5.1 Valid combinations

First, we reduce the search space by assuming that the computing, network and storage capacities are discrete. In a typical case we can consider 3 discrete values of computing capacity for each component: one providing the maximum needed capacity Θ_i^s, one providing the minimum acceptable value θ_i^s, and one intermediate value. Network and storage capacities have 3 discrete values too. They may optionally depend linearly on computing capacity, as stated in Section 3.4, leading to a further reduction of the combinations of values of the problem variables satisfying the allocation constraints.

The notation Σ^s refers to the set of valid combinations of x, y and z variables, and σ^s to refer to an individual combination respectively. The valid combinations Σ^s should satisfy the constraints described in Section 4, i.e., the allocated computing, network and storage capacity should be enough to cover the minimum probability of availability ϕ^s. The number we need to examine can be reduced assuming that the availability of the service is determined by the minimum availability of all components (see Section 4.5). That means that if we increase the capacity above a point for a certain component availability might remain the same if it exceeds the overall minimum availability so far. Subsequently neither the gain will change while the cost might be higher. If we avoid examining those cases while iterating the total number of cases can be computed as follows.

Theorem 1. *For a given service s that consists of m^s components the number of possible sets of constraints is 3^{m^s}, and this number is reduced to at most $(3\varphi^2)^{m^s}$, where φ is the golden ratio, assuming that the availability of the service is determined by the minimum availability of all components (please, refer to Appendix III for details).*

When considering the option of federating part of the capacity, again the best assumption is that we can have three levels of federated capacity, minimum (possibly 0), maximum and an intermediate value which translates to 3^{m^s}possibilities as well. Thus, the overall number of constraints is at most $(3\varphi^2)^{m^s}$. In addition, we should consider the fact that some components might be federated so if there are m^s components, there are 2^{m^s}subsets of components that can be allocated. Considering that there can be both subsets and 3 different levels of computing, network and storage capacity, one can verify that the number of all valid combinations are equal to $\sum_{i=0}^{m^s}\binom{m^s}{i}\cdot(3\varphi^2)^i = (3\varphi^2+1)^{m^s}$. When the components are more than 6, the order of magnitude of the valid combinations grows significantly, drastically increasing the associated computational cost. However, the possible combinations are significantly reduced when (anti-)affinity constraints are active.

```
1:  function ACOP($S$,$H$)
2:      $c_{max} := min(\Theta_i^s, sum(U_j))$    ▷ initialization of maximum capacity as the minimum of total required and total available
3:      $gain[0..c_{max}] := 0$    ▷ array of gains
4:      $combs[0..c_{max}] := \{nil\}$    ▷ array with lists of combinations (initially empty)
5:      for all $s \in S$ do    ▷ for each service
6:          $\Sigma^s := get\_valid\_combinations(s)$
7:          for all $\sigma^s \in \Sigma^s$ do    ▷ for each valid combination
8:              for $c = 0..c_{max}$ do    ▷ for each component in a combination
9:                  if $(c \geq c_{\sigma^s}) \wedge (gain[c] \leq gain[c - c_{\sigma^s}]) \wedge (is\_alloc\_found(combs[c - c_{\sigma^s}] \cup \sigma^s, H)$ then
10:                     $\_gain[c] := gain[c - c_{\sigma^s}] + G^s$    ▷ update auxiliary arrays
11:                     $\_combs[c] := combs[c - c_{\sigma^s}] \cup \sigma^s$
12:                 end if
13:             end for
14:         end for
15:         $gain := \_gain$    ▷ update result
16:         $combs := \_combs$
17:     end for
18:     return (gain, combs)
19: end function
```

Figure 3. ACOP Heuristic solver algorithm

5.2 Calculation of objective

It is easy to verify that one part of the objective function, i.e. $G^s = \sum_{s\in\mathcal{S}} x^s(\mathcal{G}^s - w_{\mathcal{P}}\mathcal{P}^s - w_{\mathcal{C}}\mathcal{CF}^s)$, is independent of the allocation of the components and depends only on the given combination and the allocated part of the service. Variable x^s can be easily computed depending on whether each component is involved in the allocation or not. Also, given that $\mathcal{P}^s = \overline{\Phi^s}P^s$, we can compute this factor using Eq. (11), as the computing capacity is known for each component. Therefore we can compute the whole term as all other values are constant for each service. For each combination, we also need information about the (anti-)affinity rules and the total capacity of each component, so that no constraint is violated.

5.3 Possible allocations

As already explained in Section 5.1, the computing capacity was discretized, i.e. $1\,GHz = 1$ unit of computing capacity. Using the notation in Section 3, the pseudocode in Figure 3 formalizes the ACOP approximation algorithm for discretised capacities.

The inputs of the algorithm are the set of services $\mathcal{S}$ and the set of hosts $\mathcal{H}$[5]. At lines 3 and 4 we define the structures that will store the best gains for a given capacity and their respective lists of combinations. Combinations will contain any information needed for allocation i.e. which components are included, what

5. Indexes s and j are related to the sets $\mathcal{S}$ and $\mathcal{H}$ respectively.

is their computing,network and storage capacity, and if there are any constraints. At line 6, the function *get_valid_combinations,* is not defined due to space constraints but its logic is explained in detail in Section 5.1. At lines 9 to 11 a dynamic programming approach is followed which resembles Knapsack. The purpose is to find the combinations of components that maximize gain for a given capacity with the additional constraint that no two combinations of the same service should appear. Therefore the auxiliary matrices are used. It is important to notice that the corresponding items of Knapsack algorithm are the combinations of our algorithm and not the components, while their total capacity is the corresponding item size. If the combination under examination fits in capacity, its gain $gain[c - c_{\sigma^s}] + G^s$ improves the current best gain $\_gain[c]$ and at the same time an allocation of a new set of combinations is found, then the auxiliary matrices are updated. At lines 15 and 16 when all combinations of a service are examined the actual gain and combination arrays are updated.

The ACOP heuristic does not find all possible allocations nor the most profitable ones in all cases. Even the problem of finding if a set of components fits in a Cloud is an NP-Hard problem and practically impossible to solve effectively. Instead, the problem of finding whether an allocation is possible or not, is reduced to a decision problem of bin packing in case affinity rules are active for every component (each component must be allocated in one host) and all hosts have the same capacity:

$$\begin{aligned} &\sum_{i \in S^s} \vartheta_i \cdot \eta^s_{i,j} \le U_j, && j \in \mathcal{H} \\ &\sum_{j \in \mathcal{H}} \eta^s_{i,j} = 1, && i \in \mathcal{S}^s \\ &\eta^s_{i,j} \in \{0,1\}, && i \in \mathcal{S}^s,\ j \in \mathcal{H} \end{aligned} \tag{15}$$

where ϑ_i is the total computing capacity of a component (which is a constant value between θ^s_i and Θ^s_i), $\eta^s_{i,j}$ is a Boolean variable indicating if component i is used in host j, and U_j is the capacity of hosts. This is of course a special case of input for our problem but that proves that there are instances of the problem that are difficult to solve, so an approximation method should be used. It should be noted that network and storage capacity are also considered as constraints of the actual algorithm and that makes the problem even more difficult.

At line 9, *is_alloc_found* function uses First Fit by taking into account any additional constraint. The complexity of this step highly depends on the constraints of components but in the worst case we have to check for all discrete computing capacities of all components of each service if they fit in any host (in case vms fit only in the last host examined). Thus the complexity using the already defined notation is $O\left(N_s \cdot m^s_{max} \cdot N_H \cdot \Theta^s_i\right)$. Finally, it is easy to show that the complexity of the algorithm is $O\left(c^s_{max} \cdot (3\varphi + 1)^{m^s_{max}} \cdot N_S\right)$ multiplied by the complexity of *is_alloc_found*.

5.4 Cost of the allocations

The cost of the allocation could be computed in the previous step of the algorithm in the process that checks if the allocation is possible. However, the combinations that are finally rejected are many more than those stored and computing the allocation cost for each one of them would add a significant overhead. Now the number of the costs that are computed correspond to c_{max}. The cost of allocation corresponds to the term $C = (-w_{\mathcal{C}}\mathcal{C} + w_{\mathcal{E}}\mathcal{E})$ of the objective function, where $\mathcal{C}$ and $\mathcal{E}$ are computed by Eqs. 12 and 13 respectively. The algorithm tries to maximize this term by selecting hosts with lower cost ($\mathcal{C}$) and higher eco efficiency ($\mathcal{E}$).

5.5 Result computation

In order to find the best result, we calculate the objective function for each allocation previously found. This is equal to $(G^s - C)$. The cost of hosting a service is generally lower than the gain, so the more computing capacity we allocate the more gain we obtain. However, an increase in capacity may cause minor changes in the availability and an insignificant increase in gain, thus the objective function is not necessarily greater when more capacity is allocated. Thus, the objective function needs to be computed for all possible values of the capacity. The complexity of doing that is proportional to c_{max}.

6 Evaluation

In this section, we present a detailed performance evaluation of the formal MINLP formulation of the ACOP problem, as formulated in Section 4, and its heuristic ad-hoc algorithm, discussed in Section 5. The formal MINLP optimization problem was modelled on the General Algebraic Modelling System (GAMS)[6]. For solving it, we used the Branch and Reduce Optimization Navigator (BARON) [31], which is a computational system for solving non-convex MINLP problems to global optimality. The reason for focusing on BARON was that other MINLP solvers for non-convex problems, either failed to solve the problem or converged much slower.

In the conducted experimentation, different model profiles were generated for different number of services, components and subnets. The simulated problem types included many of the features described in Section 4 but not all of them, as detailed below in the various cases. Correspondingly, the problem formulations have been simplified whenever applicable. All required inputs, such as the requirements of the components, the capacity of the hosts, etc., were generated so that they have constant values across all profiles for better comparison. The average running time for each model profile was computed across 5 repeated experiments to increase the accuracy. All results were obtained using BARON with GAMS v23.7 and Python v2.7.3, which was used for the

6. GAMS Development Corporation. General Algebraic Modeling System (GAMS). Available at http://www.gams.com/.

implementation of the ACOP heuristic solver, on AMD FX Six-Core 3.32 GHz processor with 8 GB of RAM. BARON was configured to use 1 GB of RAM with 1% accuracy and to halt after 500 seconds.

6.1 Performance evaluation

Figure 4 plots the CPU time consumed by the ACOP heuristic and BARON solvers to find the optimal solution for each of the examined profiles of different sizes. The number of hosts to be considered as candidates for allocation is plotted along the x-axis reaching a maximum of 100, whereas the CPU time consumed is plotted along the y-axis. In order to test the problem under a difficult input, anti-affinity rules on network and host level were used, whereas all other values were constant. The three lines that are plotted for each solver correspond to three different indicative profiles. For example, profile(4,3,10) considers 4 services with 3 components each and 10 subnets. Note that the maximum requirements, i.e., number of VMs to be considered for allocation for each component, was set to 5.

Although the results prove that the problem is tractable and that the BARON solver can yield the optimal solution, they indicate that the cost of an accurate solution may be considered not acceptable even for small to medium-sized problems. Thus, the unpredictable run times and the lack of scalability indicate that the usage of global MINLP solvers is unsuitable for on-line admission control. Note that for larger sized problems, the running times for BARON proved to be impractical (hundreds of minutes in some cases) and for this reason they are not reported here. Also, the usage of approximation techniques, such as setting the optimality threshold of the BARON solver to much higher values, severely impacts the quality of the solution without decreasing the consumed time to an acceptable value.

As it can be seen, the solutions obtained by the ACOP heuristic are approximately two to three orders of magnitude faster than those obtained by BARON for 2 out of 3 profiles. For the profile(1,6,3), there are some interesting conclusions that can be drawn. The increased number of components and the fact that there are no affinity rules in place generates a large number of allocations to be examined by the ACOP heuristic. In addition, the anti-affinity rules that are active result in more checks and increase the time needed by the ACOP heuristic to find a valid allocation. As a result the average performance of the ACOP heuristic is worse that BARON's in this profile, whereas BARON performs better than in other profiles. This may be due to the fact that BARON examines much less combinations and all other values are relatively small so the total number of equations is smaller and easier to solve. Another observation is that even though the ACOP heuristic performs worse on average on this specific profile, it still demonstrates better scalability as the number of the hosts increases and starts to outperform BARON as the "bottleneck" of

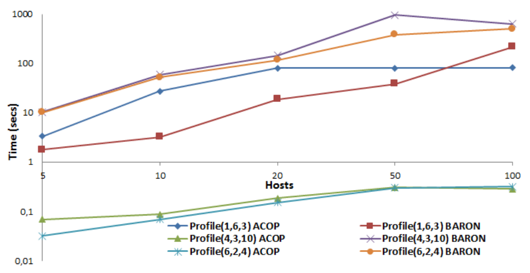


Figure 4. CPU times for ACOP heuristic and BARON solvers.

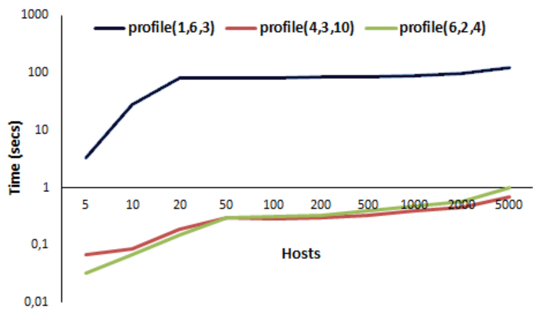


Figure 5. CPU times of ACOP heuristic up to 5000 hosts.

the algorithm does not depend on the number of hosts due to the usage of the First Fit algorithm.

The ACOP heuristic solver was then used to solve much larger problems, whereby, as already mentioned, BARON was proven to be impractical. Figure 5 plots the solution times of the ACOP heuristic solver with respect to the number of hosts, reaching a maximum of 5000 hosts. As shown, the running times scale with the number of hosts, subnets, services and components, with the latter having more negative impact on the solution time (profile(1,6,3)). Specifically, the number of the components dominate as a factor that affects execution time, whereas the number of the services becomes more significant as the number of the hosts increases. It is evident that the number of hosts does not have any significant impact once an allocation is found in a subset of hosts, with the execution times remaining relatively steady up to 5000 hosts.

6.2 Solution quality evaluation

Figure 6 plots the ratio of the value that the objective function is given using the BARON solver to the one obtained by the ACOP, with the optimality threshold for BARON set to 0.1%. The results show that, for profiles (4,3,10) and (6,2,4) the ACOP heuristic's solution quality increased with the number of hosts from 98% for 5 hosts, and maintained 100% from that point on. Interestingly, those 2 profiles are also the one's that the ACOP heuristic outperformed BARON by three orders of magnitude (see Figure 4). For profile (1,6,3), the ACOP heuristic's solution quality was only 60% of the optimal as hosts increased, dropping from about 90%, its initial value.

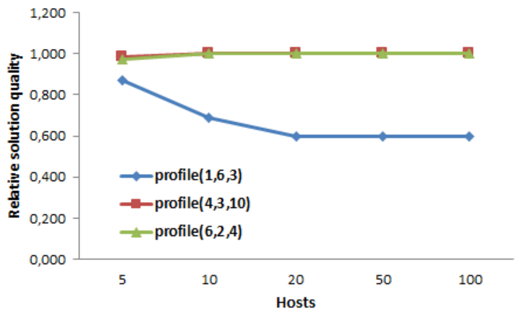


Figure 6. Solution quality of ACOP heuristic vs BARON.

Table 1
Hosts and services characteristics

$j \in \mathcal{H}$	κ_j	E_j	$n \in \mathcal{N}$
$j_1 - j_{10}$	2.5	0.5	n_1
$j_{11} - j_{20}$	3	1	n_2

S	s_1	s_2
$\mathcal{G}^s$	100	120
$\mathcal{P}^s$	10	12

Although BARON may perform better in specific cases in which the number of components is relatively large, this happens only when the number of hosts is kept very low, which is not realistic when it comes to Clouds that usually comprise thousands of hosts. Also, due to the inherent non-convexity of the presented problem, it is not easy to predict BARON's running times, which might end up much worse than expected, even in very similar cases. As it can be concluded, the combination of the much faster running times, and the high quality of its solutions, make the ACOP heuristic far more suitable for on-line admission control.

7 Case Study

In this section, we highlight the way the flexibility that is introduced by the probabilistic framework, as described in Section 4.5, is regulated by the weights of the different factors in the multi-objective function, and the way changing these weights results in different solution patterns in terms of cost, allocated capacity, selected hosts, etc. We consider a case study of 2 different subnets $\{n_1, n_2\}$ of the same capacity $(1\,Gb/s)$. Each subnet interconnects 10 unused hosts with different usage costs ω_j and eco-efficiency scores E_j[7], as shown in Table 1. The CPU cores of the hosts are considered homogeneous and of a capacity of $U_j = 2.0\,GHz$ for hosts j_1 to j_{18} and $1.0\,GHz$ for the remaining two hosts j_{19} and j_{20}. Thus, their total computing capacity sums up to $38\,GHz$.

Under these settings, we consider 2 services $\{s_1, s_2\}$ requesting admission, as shown in Table 1. Both services consist of 2 components $\{\xi_1^s, \xi_2^s\}$, that have the same basic and elastic computing capacity: $\theta_i^s = 1\,GHz$, and $\Theta_i^s = 10\,GHz$. Thus, the maximum computing capacity

7. The eco-efficiency scores were normalized between 0 and 1, i.e. in the simulation example, the value 0.5 means that the related hosts were 50% less eco-efficient compared to those that demonstrated maximum eco-efficiency, which was set to 1.

Table 2
Indicative cases under examination

Case	$w_\mathcal{E}$	ϕ^{s_1}
I	0	0.8
II	0	0.9
III	0	1

Case	$w_\mathcal{E}$	ϕ^{s_1}
IV*	0	1
V**	0	0.8
VI	1	0.8

* Affinity for s_1 components on cloud level
**Anti-affinity for s_1 components on network level

that can be accepted for deterministically admitting both services is $40\,GHz$, which exceeds the available one $(38\,GHz)$. For simplicity and without loss of generality, we consider that the cumulative probability distributions of the components capacity requirements $\mathcal{F}_i^s(x_i^s)$ (see Section 4.5) are independent and uniformly distributed in the interval $[\theta_i^s, \Theta_i^s]$ (i.e. probability Φ^s is given by Eq. (11)), and that their network and storage requirements are negligible compared to the available capacity.

Six indicative cases are examined with each of them having a different combination of availabilities and CP acceptance policies, as expressed by the different values of the weights in the objective function and (anti-)affinity rules in place (Table 2). The values of ϕ^s, which is the minimum probability that the required elastic capacity will be actually available when needed, is kept fixed to 0.8 for service s_2, whereas for service s_1 it takes different values, as shown in Table 2. The sum of the weights is equal to 1 in all cases. Table 3 summarizes the obtained solutions by the ACOP solver.

In Case I, the weights $w_\mathcal{C}$ and $w_\mathcal{P}$ are both set to 0.5, whereas $w_\mathcal{E}$ is set to 0, denoting a profit-driven CP. The requested availability for service s_1 is 0.8 and there are no (anti-)affinity rules in place. According to the output of the ACOP, the solution distributes the instances of two services on 19 hosts (host j_{16} remains unoccupied), whereas the allocated capacity is compressed to the minimum allowed by the probabilistic constraint of Eq. 10, with service s_2 being fully allocated since $\phi^{s_2} = 1$ (Table 3, Case I). Also, the host that remains unoccupied belongs to subnet n_2, which interconnects hosts of higher cost compared to subnet n_1.

For Case II, in which the requested availability for service s_1 is increased to 0.9, the allocation pattern now includes turning on all hosts for hosting more elastic capacity. Further increasing ϕ^{s_2} to 1 as in Case III, leads to the acceptance of the maximum amount of requirements for service s_1, with the extra capacity that could not be locally allocated, being federated onto another CP (Table 3, Cases II and III). It should be noted that similar effect can be achieved by increasing the penalty weight while the availability remains fixed. This means that the CP wants to decrease the risk of paying penalties to the clients, therefore more elastic capacity is offered.

Case IV, is similar to the previous one with the only difference being that the two components of service s_1 are now bound to each other with affinity on cloud level, i.e. all their instances must reside in the same

Table 3
Comparison of different cases

Case	I				II				III				IV				V				VI			
Services	s_1		s_2		s_1		s_2		s_1		s_2		s_1		s_2		s_1		s_2		s_1		s_2	
Components	i_1	i_2	i_1	i_2	i_1	i_2	i_1	i_2	i_1	i_2	i_1	i_2	i_1	i_2	i_1	i_2	i_1	i_2	i_1	i_2	i_1	i_2	i_1	i_2
Involved network	n_1	n_1	n_2	n_1,n_2	n_2	n_2	n_1	n_1	n_2	n_2	n_1	n_1	-	-	n_1	n_1	n_1	n_2	n_1	n_2	n_1	n_1	n_2	n_1,n_2
Allocated capacity	8	8	10	10	9	9	10	10	10	8	10	10	0	0	10	10	8	8	10	10	8	8	10	10
Federated capacity	0	0	0	0	0	0	0	0	2	0	0	0	10	10	0	0	0	0	0	0	0	0	0	0
Unoccupied hosts	j_{16}				-				-				$j_{11} - j_{20}$				-				j_{10}			
Gain-Cost $\mathcal{G} - C$	220-52				220-55				220-55				220-25				220-52.5				220-52.5			
Eco-efficiency $\mathcal{E}$	14				15				15				5				14.5				14.5			
Expected penalty $\mathcal{P}$	2				1				0				0				2				2			

Cloud. Interestingly, the allocation pattern now involves federating the entire capacity that was requested for s_1 (Table 3, Case IV). This is due to the fact that under deterministic admission, it is not possible to allocate s_1 in the underlying cloud while allocating s_2 at the same time. Although part of s_1 could be allocated locally, as demonstrated in Case III, the existence of the affinity rule forces the entire service to be federated. Case V shares the same input as Case I except that anti-affinity is applied on network level for service s_1, i.e., the components that comprise this service must be allocated in different subnets. Contrary to Case I whereby the deployment of service s_1 was contained within the same subnet, the activation of the specific anti-affinity rule forces the two components of service s_1 to be allocated on different subnets, even though the gain is decreased compared to the one behind Case I.

Case VI is similar to Case I with the only difference being that the weight related to the ecology factor $w_{\mathcal{E}}$ has increased from 0 to 1. Therefore, the eco-efficiency of the available hosts is now considered in the optimization process. Contrary to Case I, the optimal allocation pattern now involves turning on more hosts in subnet n_2 instead of n_1, and the overall eco-score of the allocation pattern is increased, whereas the gain is decreased (Table 3, Case VI). This is due to the fact that the hosts of subnet n_2, although more expensive and equivalent to the ones of n_1 in terms of computing power, they are characterized by a higher eco-efficiency score.

8 Conclusions

This paper proposed an approach to the problem of optimum allocation of services on virtualized physical resources, with the following key contributions.

Elasticity of resources. We focused on hosting elastic services whose resource requirements may dynamically grow and shrink, depending on the dynamically varying number of users and patterns of requests. The presented optimization model exploits statistical knowledge about the elastic workload requirements of the services in order to reduce the resources needed to maintain a given quality of level of service. In doing so, it also accounts for business level objectives such as cost and eco-efficiency.

Cloud federation. The presented approach allows for the establishment of federations among Cloud providers by considering the acceptance of a service with the possibility to outsource it (entirely or partially) to other Cloud providers. The model acknowledges any extra federation costs, communication overheads and affinity/anti-affinity rules and achieves to seamlessly handle federated resources similarly to local ones while maintaining its optimization goals across the Cloud federation.

Given the high complexity of the existing methods for solving the formulated MINLP non-convex problem, a heuristic solver was proposed that exhibits reduced computation times up to 3 orders of magnitude compared to the exact solver, while achieving a high solution quality, making the proposed method practical and suitable for on-line resource allocation in federated Clouds.

Acknowledgments

The research leading to these results has received funding from the European Commission's Seventh Framework Programme under grant agreement n.257115 OPTIMIS – Optimized Infrastructure Services and n.248465 S(o)OS – Service-oriented Operating Systems.

References

[1] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, May 2011.

[2] M. Chiosi et al. Network functions virtualisation - introductory white paper. In *Proc. of the SDN and OpenFlow World Congress*, Darmstadt, Germany, October 2012.

[3] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 228 –235, july 2010.

[4] M. Mazzucco, D. Dyachuk, and R. Deters. Maximizing cloud providers' revenues via energy aware allocation policies. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 131 –138, july 2010.

[5] Z. Xiao, W. Song, and Q. Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1107–1117, 2013.

[6] S. Di and C. Wang. Dynamic optimization of multiattribute resource allocation in self-organizing clouds. *Parallel and Distributed Systems, IEEE Transactions on*, 24(3):464–478, 2013.

[7] L. Eyraud-Dubois and H. Larcheveque. Optimizing resource allocation while handling SLA violations in cloud computing platforms. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 79–87, 2013.

[8] P. Padala, K. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 13–26, New York, NY, USA, 2009. ACM.

[9] F. Chang, J. Ren, and R. Viswanathan. Optimal resource allocation in clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 418 –425, july 2010.

[10] F. Chang, R. Viswanathan, and T. L. Wood. Placement in clouds for application-level latency requirements. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 327–335, 2012.

[11] X. An, F. Pianese, I. Widjaja, and U. Günay Acer. DMME: A Distributed LTE Mobility Management Entity. *Bell Labs Technical Journal*, 17(2):97–120, 2012.

[12] M. Alicherry and T. V. Lakshman. Network aware resource allocation in distributed clouds. In *INFOCOM*, pages 963–971, 2012.

[13] J.W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang. Joint VM placement and routing for data center traffic engineering. In *INFOCOM, 2012 Proceedings IEEE*, pages 2876–2880, 2012.

[14] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: automated data placement for geo-distributed cloud services. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 2–2, Berkeley, CA, USA, 2010. USENIX Association.

[15] Z. Zhang, H. Wang, L. Xiao, and L. Ruan. A statistical based resource allocation scheme in cloud. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 266 –273, dec. 2011.

[16] K. Konstanteli, T. Cucinotta, and T. Varvarigou. Optimum allocation of distributed service workflows with probabilistic real-time guarantees. *Springer Service Oriented Computing and Applications*, 4(4):229–243, 10 2010.

[17] H. Zheng, J. Yang, and W. Zhao. QoS probability distribution estimation for web services and service compositions. In *IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1 –8, December 2010.

[18] F. Zhang, J. Cao, H. Cai, J.J. Mulcahy, and C. Wu. Adaptive virtual machine provisioning in elastic multi-tier cloud platforms. In *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, pages 11–19, 2011.

[19] K. Mills, J. Filliben, and C. Dabrowski. Comparing VM-placement algorithms for on-demand clouds. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 91–98, Washington, DC, USA, 2011. IEEE Computer Society.

[20] H. Xu and B. Li. Anchor: A versatile and efficient framework for resource management in the cloud. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1066–1076, 2013.

[21] M. Maurer, I. Brandic, and R. Sakellariou. Enacting SLAs in clouds using rules. In *Proceedings of the 17th international conference on Parallel processing - Volume Part I*, Euro-Par'11, pages 455–466, Berlin, Heidelberg, 2011. Springer-Verlag.

[22] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou. Admission control for elastic cloud services. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 41–48, 2012.

[23] Ferrer et al. OPTIMIS: A holistic approach to cloud service provisioning. *Future Generation Comp. Syst.*, 28(1):66 – 77, 2012.

[24] A. N. Toosi, R. K. Thulasiram, and R. Buyya. Financial option market model for federated cloud environments. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 3–12, 2012.

[25] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari. Hierarchical multiprocessor CPU reservations for the Linux Kernel. In *Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009)*, Dublin, Ireland, June 2009.

[26] Cucinotta et al. Virtualised e-learning on the IRMOS real-time cloud. *Service Oriented Computing and Applications*, pages 1–16. 10.1007/s11761-011-0089-4.

[27] C. Liu, J. Huang, Q. Cao, S. Wan, and C. Xie. Evaluating energy and performance for server-class hardware configurations. In *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, pages 339–347, 2011.

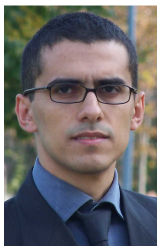
[28] O. Werner-Kytölä and W. F. Tichy. Self-tuning parallelism. In *Proceedings of the 8th International Conference on High-Performance Computing and Networking*, HPCN Europe 2000, pages 300–312, London, UK, 2000. Springer-Verlag.

[29] T. Cucinotta. Optimum scalability point for parallelisable real-time components. In *Proceedings of the International Workshop on Synthesis and Optimization Methods for Real-time and Embedded Systems (SOMRES 2011)*, Vienna, Austria, November 2011.

[30] S. Mallick. Virtualization based cloud capacity prediction. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 849 –852, july 2011.

[31] M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Math. Program.*, 103(2):225–249, June 2005.

Kleopatra Konstanteli received her diploma in Electrical and Computer Engineering in 2004 by the National Technical University of Athens (NTUA). In 2007 she received a Master in Techno-economical Systems (NTUA, University of Athens, and University of Piraeus), and her Ph.D. in the area of distributed computing from the school of Electrical and Computer Engineers of NTUA in 2011. She has been working as a research associate at NTUA since 2005. Her research interests are mainly focused in the field of distributed computing.



Tommaso Cucinotta is a member of technical staff at Bell Labs in Dublin, Ireland. He has a computer engineering degree from the University of Pisa (Italy), as well as a Ph.D. in computer engineering from the Scuola Superiore Sant' Anna of Pisa, where he spent more than 10 years researching on topics including computer security, smartcard-based authentication, soft real time scheduling, embedded systems, resource management, general purpose operating systems, real-time virtualization and cloud computing. He has published several scientific papers on the topics above and serves as reviewer for several international journals, conferences, and workshops in his areas of expertise. He also recently filed almost 20 patents on the above topics.

Konstantinos Psychas received his diploma in Electrical and Computer Engineering in 2012 by the National Technical University of Athens (NTUA). He has been a research associate at NTUA, working on EU funded research projects since then and until 2013. He is currently a Ph.D student at the Department of Electrical Engineering in Columbia University in New York where he works in a computational neuroscience project.

Prof. Theodora A. Varvarigou received the B. Tech degree from the National Technical University of Athens, Athens, Greece in 1988, the MS degrees in Electrical Engineering (1989) and in Computer Science (1991) from Stanford University, Stanford, California in 1989 and the Ph.D. degree from Stanford University as well in 1991. She worked at AT&T Bell Labs, Holmdel, New Jersey between 1991 and 1995. Between 1995 and 1997 she worked as an Assistant Professor at the Technical University of Crete, Chania, Greece. Since 1997 she was elected as an Assistant Professor while since 2007 she is a Professor at the National Technical University of Athens, and Director of the Postgraduate Course 'Engineering Economics Systems'. She has great experience in the area of distributed computing and embedded systems, and has published more than 150 papers in leading journals and conferences.