Energy-Aware Adaptive Scaling of Server Farms for NFV with Reliability Requirements

Jesus Perez-Valero, Albert Banchs, *Senior Member, IEEE*, Pablo Serrano, *Senior Member, IEEE*, Jorge Ortín, Jaime Garcia-Reinoso, and Xavier Costa-Pérez, *Senior Member, IEEE*

Abstract—Auto-scaling techniques aim to keep the right number of active servers for the current load: if this number is too small we risk service disruption, but if it is too large we waste resources. Despite the interest in the efficient operation of this type of systems, no prior work has addressed auto-scaling techniques for Network Function Virtualization (NFV) with stringent reliability requirements such as those envisioned in 5G (5 or 6 nines). To achieve such levels of reliability, we need to account for both the activation delay until servers become available (i.e., the wake-up or activation time) and the fallible nature of servers (which may fail with some probability). In this paper, we build on control theory to design an auto-scaling technique for a server farm for NFV that guarantees certain reliability while minimizing the number of active resources. We show that the use of well-established tools from control theory results in convergence times much shorter than those obtained with state-of-the-art reinforcement learning techniques. This shows that, despite the current trend to apply machine learning to all sorts of networking problems, there may be some cases where other techniques (such as control theory) can be more suitable.

Index Terms—Adaptive scaling, control theory, NFV, reinforcement learning

1 Introduction

With the deployment of 5G/6G, mobile services will be commonly implemented as network slices, i.e., as interconnected virtual network functions (VNFs) hosted by cloud servers. To make an efficient use of the resources, VNFs need to be scaled up and down based on the load of the VNFs [1], [2]. When scaling up resources, these need to be activated in advance, anticipating traffic demands. This is because the activation of additional resources, possibly involving the deployment of network instances, is not immediate and thus resources would not be available when needed unless they are activated in advance. Following this, relevant standard specifications by ETSI [3] and 3GPP [4] recommend that the activation of resources in such settings is performed in advance. When activating resources in advance, however, it is important to make sure that no more resources than needed are turned on, to minimize energy consumption. The design of scaling policies is particularly challenging for the case of 5G/6G services, where reliability requirements can be very stringent with levels of 5 or 6 nines (99.999% or 99.9999%, respectively) [5], [6], much beyond those attained by traditional scaling techniques [7].

- *J. Pérez-Valero, P. Serrano, and A. Banchs are with the Department of Telematic Engineering, Univ. Carlos III de Madrid, 28911 Leganés, Spain. E-mail: jesperez@pa.uc3m.es, {banchs, pablo}@it.uc3m.es,*
- *A. Banchs is also with Institute IMDEA Networks, 28912 Leganés, Spain,*
- *J. Ortin is with Centro Universitario de la Defensa, Academia General Militar, 50090 Zaragoza, Spain. E-mail: jortin@unizar.es,*
- *J. Garcia-Reinoso is with the Department of Automatic, Univ. of Alcalá, 28871 Alcalá, Spain. E-mail: jaime.garciareinoso@uah.es,*
- *X. Costa-Pérez is with i2cat, ICREA and NEC Laboratories Europe. E-mail: xavier.costa@ieee.org*

To achieve the levels of reliability required by 5G/6G services, scaling has to take into account not only the delay to activate a server but also the fact that the servers themselves are *fallible*, i.e., they can occasionally (albeit infrequently) go down. Indeed, while server failures have no perceptible impact when reliability requirements are not very strict, they cannot be ignored when pursuing such levels of reliability. This has an impact both on the developed reliability models, which need to account for server failures, as well as on the design of the scaling algorithms, which need to provide extremely high reliability.

When designing an algorithm to scale resources for 5G/6G, the challenge lies in achieving a good trade-off between reliability (activating sufficient resources in advance) and resource efficiency (not activating more resources than needed). Indeed, if too many resources are activated unnecessarily, this will result in a waste of resources. As a matter of fact, data centers running VNFs consume a very substantial amount of energy [8], more than physical servers [9], and significantly contribute to the electricity bill of network operation, which is one of the chief concerns of network operators nowadays [10]. Hence, in a Network Function Virtualization (NFV) context, it is of utmost importance to switch off or bring to a low-power state as many servers as possible, minimizing the number of active servers. The optimal trade-off between reliability and efficiency highly depends on the current conditions. For instance, for low arrival rates, we do not need very large safety margins when activating new servers; however, if arrival rates are high, we need larger margins or risk filling up all available servers. Thus, we need an *adaptive scaling algorithm* that dynamically adjusts the operation of the system based on its behavior, to meet the desired reliability level while minimizing energy consumption.

When designing such an adaptive algorithm, *artificial*

intelligence techniques may seem a natural choice. However, even though artificial intelligence techniques such as reinforcement learning have indeed proved very effective in many areas (see e.g. the recent surveys [11], [12]), we argue that for the specific problem addressed in this paper, where we need to adapt as quickly as possible to changing conditions of the network or risk to operate under non-desired conditions for a substantial amount of time, artificial intelligence techniques may be too slow to converge. In contrast to artificial intelligence techniques, control theory allows the design of algorithms that converge as quickly as possible to the desired point of operation while ensuring that the system does not turn unstable. In this paper, we leverage control theory to design a solution called A3S (Adaptive Algorithm for Auto Scaling), that dynamically adjusts the system parameters of an NFV system to provide an optimal trade-off between reliability and energy consumption. Our performance results show that our solution is much faster than a widely used reinforcement learning technique that has been designed for a similar goal to the one addressed in this paper.

Contributions

We propose A3S, a closed-loop system for adaptive scaling of server farms in the context of NFV which, in contrast to previous proposals, is capable of providing very high levels of reliability as required by 5G/6G systems. The proposed system leverages control theory to automatically adjust its configuration to provide an optimal trade-off between reliability and power consumption, bringing as many servers as possible to sleep. We thoroughly evaluate the performance of A3S via simulation. Our performance analysis shows that the proposed system quickly converges to the desired point of operation while keeping stable. A comparison against a well-established reinforcement learning (RL) algorithm shows that control theory is much quicker to adapt and hence much more suitable for this kind of problem. We further show that our technique is capable of adapting to real traffic, while a solution derived analytically may fail when dealing with real traffic.

The key novelties of our work are as follows (see Section 5 for more details): (*i*) to address this kind of problems, previous works have built on complex analytical models that are only valid for static conditions; (*ii*) the impact of fallible servers is neglected by most works in the literature, which consider only non-zero activation times; and (*iii*) in contrast to "agnostic" artificial intelligence techniques (e.g., Q-learning), which are too slow to adapt, our control theoretic design exploits the structure of the system and thus is much faster.

The rest of the paper is structured as follows. In Section 2 we present our system model and formally expose the problem that we aim at solving in this paper, which involves finding an optimal trade-off between reliability and power consumption. In Section 3 we design a closed-loop system along with an adaptive algorithm based on control theory to solve this problem. Section 4 evaluates the performance of the proposed approach, showing that (*i*) it meets the reliability target, (*ii*) it minimizes power consumption, (*iii*) it adapts quickly to changing conditions, and (*iv*) it outperforms a machine learning approach. Section 5 provides a detailed review of related work, highlighting the novelties of our approach, and Section 6 presents some concluding remarks.

2 System Model

In the following, we present the model of the system proposed in this paper: first, we describe the server farm behavior as a function of the activation thresholds; then, we propose our policy for setting such thresholds; finally, we expose the problem addressed by this paper.

2.1 Server farm

The system we consider in this paper is a server farm for NFV. The server farm implements dynamic power management (DPM), where idle servers are switched into one of the multiple low-power states available to save energy [13].

The system is composed of an infrastructure manager and S servers. There are tasks arriving at the system which need to be served by the servers. To handle these tasks, a subset of the S servers is running, while the remaining servers are in one of the available sleep modes to save power, and can be activated when needed. More specifically, each server can be in one of the following states: (*i*) running (i.e., active), a state in which it can serve tasks; (*ii*) in a low-power sleep mode; (*iii*) switching from a sleep mode to the active mode (or vice versa), when the infrastructure manager decides that more (less) resources are needed; or (*iv*) booting-up, after a server crash (we assume that a server can always restart after a crash). The infrastructure manager decides the number of servers that should be active at a given point in time as well as the low-power sleep mode employed. Since there is a clear trade-off between power consumption and the delay to switch between a low-power state into the active state, selecting the proper sleep state in NFV is crucial for the performance of the system.

We denote the average task arrival rate to the system by . Upon arrival, the infrastructure manager decides which of the active servers with enough resources should handle the task, with the aim of balancing the load across servers. Alternatively, if there are no servers available to handle an arriving task, the infrastructure manager requests the activation of a server in sleep-mode (if any), holding the task in a queue until there are enough resources to serve it. A task involves the allocation of computational resources for the duration of the corresponding session, e.g., to run the corresponding virtual machine. We assume that one server can serve up to T simultaneous tasks and denote by 1= the average service time of a task. We assume that servers have a finite lifetime, and denote by 1= the average lifetime of a server until it crashes, i.e., its mean time to failure (MTTF). We also assume a non-zero boot time with average 1= , which is also the recovery time after a crash. In case of a hardware failure that does not allow rebooting the server, we assume that a new server is booted up instead. Eventually, when the number of active tasks in the system is sufficiently low, we can switch servers to one of the available sleep modes. We assume that the time required to put a server in sleep mode is negligible, and that the average time to activate a server from the sleep mode is also 1= .

The infrastructure manager keeps track of the system status, and after detecting a failure (using tools such as e.g. DOCTOR[1]) it triggers the relocation of the affected tasks with e.g. life migration tools available with `OpenStack`,[2] `VMWare` [14], or containers [15]. If there are not enough resources available right after a crash, the task is placed in a system queue until a server is available, thus causing a service disruption. Similarly, if a new task arrives at the system and there are no resources to serve it, this is also considered a service disruption. Hereafter, we consider any task suffering a service disruption as a 'failure' and refer to the percentage of tasks suffering a failure as 'failure probability.'

In addition to the failure probability, the other key metric is power consumption. We assume that the average power consumed by the server farm is given by four terms:

$$! = !_s + !_a + !_m + !_z \quad (1)$$

where the term $!_s$ corresponds to the average power consumed when serving the tasks, the term $!_a$ represents the power consumption due to the activation of the servers, the term $!_m$ represents the extra power consumption due to task migrations, and the term $!_z$ corresponds to the average power consumed by servers in sleep mode.

To compute $!_s$, we follow traditional power consumption models in the literature [8], [13], that characterize the power consumption behavior with the following two terms: (i) the idle power consumption P_{idle}, and (ii) a term that is proportional to the utilization P_{load}. Assuming that the average load per server is t, and the average number of active servers is N_s, $!_s$ can be expressed as

$$!_s = N_s P_{idle} + N_s \frac{t}{T} P_{load}. \quad (2)$$

To compute $!_a$, we assume that during its activation time (which lasts 1=), a server operates at its peak power consumption rate (i.e., $P_{idle} + P_{load}$), hence the energy consumption of one activation is:

$$e_a = (1=)(P_{idle} + P_{load}). \quad (3)$$

Since this is the energy consumption of one activation, to compute the power $!_a$ (i.e., energy over time), we multiply e_a by the server activation rate (i.e., the rate at which servers are powered on over time), which we denote by . In this way, $!_a$ can be computed as

$$!_a = (P_{idle} + P_{load})1= : \quad (4)$$

To compute $!_m$, we first compute the energy cost of a migration, following the measurements reported in [16] that characterize the energy cost of reading and writing from a server, denoted es E_r and E_w, respectively. Given an average migration of $_m$ tasks over time, $!_m$ can be computed as:

$$!_m = _m(E_r + E_w). \quad (5)$$

Finally, the $!_z$ can be computed as:

$$!_z = (S \quad N_s)P_{sleep} \quad (6)$$

where P_{sleep} is the power consumption in the sleep mode.

1. https://wiki.opnfv.org/display/doctor
2. https://docs.openstack.org/nova/pike/admin/live-migration-usage.html

2.2 Threshold-based (de)activation policy

The policy followed for the management of the resources is as follows. At least one server is kept active at all times (even when the system is empty) to avoid any delay in arriving tasks. For the rest of the servers, we employ a threshold-based policy that decides whether to activate an additional server based on the number of tasks being served and the number of active servers. We denote by s_m the activation threshold for the m^{th} server and proceed as follows. On the one hand, if the number of tasks in the system reaches s_m, we trigger the activation of the m^{th} server. On the other hand, when the number of tasks falls below s_m, tasks are seamlessly moved (using e.g. [14] or [15], as discussed in the previous section) to free one server, which is then switched into the sleeping mode.

The setting of the thresholds $fs_m g$ is a critical choice for the performance of the system. The highest possible threshold value to serve all current tasks is $s_m = (m \quad 1)T$, which corresponds to the minimum number of active servers needed to serve all the tasks present in the system (i.e., given that each server can serve up to T tasks, $(m \quad 1)T$ tasks fill $m \quad 1$ severs and at that point we need to activate the m^{th} server). Note, however, that if we want to ensure that an arriving task will not have to wait for a server to boot up, we need to choose lower threshold values. Indeed, with $s_m = (m \quad 1)T$, when task $(m \quad 1)T + 1$ arrives at the system, it will have to wait if the m^{th} server has not finished its boot up before the task can be served. Yet, if we set the thresholds unnecessarily low, we will have more servers running than those actually needed, paying a price in power consumption. Thus, when setting the activation thresholds there is a trade-off between reliability and power consumption.

2.3 Problem description

In this paper, we address the problem of optimally scaling an NFV server farm by appropriately setting the activation thresholds of our threshold-based policy. As 5G/6G services typically have Service Level Agreements (SLAs) that involve a certain level of reliability, the proposed scaling policy needs to satisfy such reliability requirements. These requirements are expressed in terms of a target failure probability that cannot be exceeded, i.e., the probability that a task is disrupted cannot exceed a certain target T_f. For 5G/6G services such as URLLC (ultra-reliable and low latency communications), such target failure probability can be as low as 10^{-5} or less, while for other services such as eMBB (enhanced mobile broadband) it does not need to be so stringent. As long as the target failure probability is met, it is desirable to save as much power as possible, to minimize the cost of running the network. Following this, the goal of this paper is to find the optimal values of the activation thresholds that (i) meet the reliability requirements of the services being provided, while (ii) minimizing power consumption as long as such reliability requirements are satisfied.

The setting of the activation thresholds to meet the above goals depends on the system conditions. For instance, when the rate of task arrivals is low and not bursty, we do not need to activate servers with a long time in advance since the likelihood that many tasks arrive before the server has

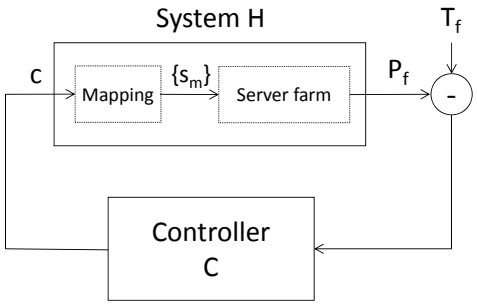

Fig. 1: Closed-loop system.

been activated will be low. Instead, in case of bursty and/or high-rate arrivals, we will need to have a much larger safety margin or risk not being able to serve arriving tasks without delay. Therefore, a static policy for setting the thresholds will not be effective in optimizing performance, and we need to adjust their setting based on the current conditions.

The rest of this paper is devoted to designing an adaptive algorithm that sets the activation thresholds based on the observed behavior of the system to provide an optimal trade-off between reliability and power consumption.

3 A3S System design

To address the problem exposed above, in the following we build A3S (Adaptive Algorithm for Auto Scaling), a closed-loop system that aims at bringing the system to the optimal point of operation using an adaptive algorithm that dynamically adjusts the activation thresholds.

3.1 Closed-loop system

Our system design is based on the closed-loop from control theory illustrated in Fig. 1. This closed-loop system is composed of the server farm itself that takes as input the thresholds $\{s_m\}$ and provides as output the failure probability, given by the percentage of tasks whose service has been disrupted measured over a time interval; this is the system's output signal P_f. This signal is compared to the reference signal, which is given by the target failure probability T_f: the difference between the output and the reference signals ($P_f - T_f$), referred to as the error signal, is fed into the controller. The controller produces as output the control signal c, which serves to derive the thresholds $\{s_m\}$ through a mapping function (described next), thus closing the loop. We denote the controller by C and the system composed by the server farm and the mapping function for the thresholds as H.

To measure the failure probability, the system proceeds as follows. We slot the time in periods comprising a given number of tasks, hereafter denoted as "slots", and measure the number of tasks that have failed over one slot, denoted by N_f. Note that there is a trade-off when selecting the slot length: an overly small number of tasks would result in a lot of noise when measuring the failure probability, while an overly large number would result in a system that takes too much time to react. As a compromise between these two extremes, we have set the slot length to $10/T_f$ tasks, which corresponds to an average of 10 failures per slot and thus provides a fairly reliable measurement.

The measured failure probability over a given slot t, $P_f(t)$ is given by the number of failed tasks in this slot, $N_f(t)$, over the total number of tasks (i.e., $10/T_f$), which gives

$$P_f(t) = N_f(t)T_f/10. \quad (7)$$

At each time slot, the controller takes as input the error $e(t)$ defined as the difference between the failure probability and the target one measured in this time slot, i.e.,

$$e(t) = P_f(t) - T_f; \quad (8)$$

and provides as output a value $c(t+1)$ that serves to configure the thresholds $\{s_m\}(t+1)$ for the next time slot.

As explained earlier, the challenge with an auto-scaling system is the configuration of the thresholds $\{s_m\}$; the goal when configuring these parameters is to minimize power consumption while guaranteeing that the failure probability stays below a given target T_f. As there is an inherent trade-off between power consumption and failure probability, in order to minimize power consumption we need to operate close to the failure probability target, as otherwise, we would be consuming more power than needed to reach this target.

Our solution is based on the following design criterion: to minimize energy consumption when operating at the target failure probability T_f, we set all thresholds $\{s_m\}$ at the same distance from their highest possible value, which is given by $(m - 1)T$. The intuition behind this criterion is that if some thresholds were more aggressive than others, the most conservative thresholds would need to be overly small to compensate for the more aggressive ones, and this would lead to wasting a large amount of power.

Next, we explain the mapping of the control signal to the thresholds and then present the adaptive algorithm itself.

3.2 Control signal mapping

The mapping of the control signal c to the thresholds is performed as follows. As explained above, we aim at setting all thresholds more or less at the same distance from the highest possible value, since this policy is effective in minimizing power consumption as corroborated by our performance evaluation results. Furthermore, to provide the highest possible level of granularity, we enforce that a difference of one unit in c only modifies one threshold by one unit (which is the minimum possible change).

As explained in Section II.B, the largest possible value for the thresholds is given by $s_m = (m - 1)T$, which corresponds to the case where we launch the m^{th} server only when the previous $m - 1$ servers are full.[3] We update the thresholds such that, at a given point in time, all the thresholds are at the same distance from such values, with differences of one unit at most due to rounding. Given that rounding imposes that some thresholds are one unit larger than others, we set the thresholds such that the lower ones take the smaller values.

3. For instance, with $T = 3$ machines per server we have, e.g., $s_3 = 6$, meaning that the third server is powered when there are 6 tasks in the system filling the first two servers.

When mapping the control signal c to the thresholds s_m, every time the control signal increases by one unit, we reduce one of the activation thresholds by one unit. More specifically, we define the mapping between the c and the s_m as follows:

With $c = 0$, we set all thresholds at the highest possible value, i.e., $s_m = (m \quad 1)T;\ 8\, m$.
For $c = 1$ we decrease s_2 by one[4] and leave all other thresholds unchanged, i.e., $s_2 = T \quad 1$ while $s_m = (m \quad 1)T$ if $m > 2$;
For $c = 2$, we decrease both s_2 and s_3 by one and leave all other thresholds unchanged, i.e. $s_2 = T \quad 1$, $s_3 = 2T \quad 1$ and $s_m = (m \quad 1)T$ if $m > 3$;

After we have decreased all thresholds by one unit (for $c = S \quad 1$), in the next increment of c (i.e., $c = S$) we decrease s_2 by two units and the remaining thresholds by one unit, i.e., $s_2 = T \quad 2$ and $s_m = (m \quad 1)T \quad 1$ if $m > 2$; we continue with the rest of the thresholds in the same way.

Following the above, the values of the thresholds s_m can be computed using the quotient (denoted as q) and the remainder (denoted as r) of c divided by $S \quad 1$, i.e.,

$$q = \frac{c}{S \quad 1}\ ; \quad r = c \quad (S \quad 1)\ \frac{c}{S \quad 1}\ ; \tag{9}$$

since each time c exceeds $S \quad 1$ all thresholds are decremented by one; in case m is equal to or smaller than $r + 1$, the threshold should be decremented once more. Following this,

$$s_m = \begin{cases} (m \quad 1)T \quad q \quad 1; & \text{if } m \quad r + 1; \\ (m \quad 1)T \quad q; & \text{otherwise.} \end{cases} \tag{10}$$

To illustrate the mapping, we give an example of a scenario with $S = 4$ servers with a per-server capacity of $T = 3$ virtual machines. In this example, the mapping of the first values of c to $\{s_m\}$ is illustrated in Table 1.

c	s_1	s_2	s_3	s_4
0	0	3	6	9
1	0	**2**	6	9
2	0	2	**5**	9
3	0	2	5	**8**
4	0	**1**	5	8

TABLE 1: Mapping of c to $\{s_m\}$ for $S = 4$ and $T = 3$.

Since the control signal can actually take any real value and not just non-negative integers, when doing this mapping we do as follows:

If c does not take an integer value, we use the closest integer to compute (9).
If c is so large that with the above expression some thresholds would take negative values, we set all thresholds to 0.
If c is negative, we set all thresholds to their highest possible value (as we do for $c = 0$).

4. Note that, since the first server is always active, s_1 is fixed to 0.

3.3 Adaptive algorithm

The control signal c needs to be dynamically adjusted through an adaptive algorithm to bring the error signal to 0. There are well-established controllers that have been designed for this purpose. In this paper, we choose the widely used Proportional-Integral (PI) controller. Some of the advantages of this controller are its simplicity and the fact that it guarantees zero error in the steady-state. With this controller, the control signal $c(t)$ at time slot t is computed based on the error signal in the previous time slots, $e(t')$ for $t' \in \{0, \ldots, t \quad 1\}$, as follows:

$$c(t) = K_p e(t \quad 1) + K_i \sum_{t'=0}^{t \quad 2} e(t'). \tag{11}$$

where K_p and K_i are the so called proportional and integral parameters of the controller. The control signal can be computed recursively as:

$$c(t) = K_p e(t \quad 1) + c(t \quad 1) + (K_i \quad K_p) e(t \quad 2). \tag{12}$$

For the setting of the K_p and K_i, we follow the same approach as in [17], [18]:

$$K_p = \frac{0.4}{\hat{H}}; \tag{13}$$

$$K_i = \frac{0.4}{\hat{H} \quad 0.85 \quad 2}. \tag{14}$$

where $\hat{H}$ is an upper bound for the module of the linearized transfer function of the system. The system takes the control signal c as input and provides the failure probability P_f as output. The linearized transfer function of the system, denoted by H, corresponds to the ratio between the variation of the failure probability, P_f, and the variation of the control signal, c, at the desired point of operation, i.e., $H = P_f = c$. To determine a bound for $|H|$, we apply the following rule of thumb. Let us assume that the system is operating at the desired failure probability, T_f, and we increase by one unit all thresholds $\{s_m;\ m \in \{2, \ldots, S\}\}$, which corresponds to increasing the control signal by $S \quad 1$ units, i.e., $c = S \quad 1$. Then, since larger c means that servers are switched on earlier, the failure probability will decrease; specifically, it will go from its value at the desired point of operation, T_f, to a smaller value. This yields the following bound for the variation of the failure probability, $|P_f| < T_f$, from which we obtain the upper bound $\hat{H} = T_f/(S \quad 1)$.

3.4 Multiple controller instances

A single instance of the control algorithm is well suited for scenarios with stationary traffic: with the proper setting of K_p and K_i, the control signal is guaranteed to arrive at the steady state after a transient period. However, real-life traffic is sometimes characterized by non-stationarity, with e.g. strong weekly correlations (weekdays vs. weekend), or even daily correlations (office vs. non-office hours, day vs. night). For these *cyclostationary* processes, instead of employing a single configuration of the activation thresholds for the whole input process, it is better to have a different threshold configuration for each period of the input process

with the same statistical properties, computed with a different control instance. To this end, the approach that we take in this paper is to run different instances of our algorithm, employing the same instance for the periods with similar statistical properties.

Fig. 2: Different instances for different periods of the day.

We illustrate the above approach in Fig. 2. The gure represents a cyclostationary process where the load (straight line) varies over time with a periodicity of one day. By running three different instances of the control algorithm (each instance is represented with a different color), the system can converge to three different values of $c(t)$: a relatively large value (magenta) for high load, a medium value (green) for an average load, and a small value (red) for a low load. Using this approach, we can thus achieve a more ef cient operation than with a single instance, since in each period the system is optimized to current traf c conditions to achieve the desired T_f while minimizing the power consumption.

To implement multiple algorithm instances running in parallel, we proceed as follows. In the transition from one period to another, the system freezes the algorithm instance corresponding to the previous period and reactivates the instance for the next period, recovering the state of that instance from the time when it was frozen.

4 Performance evaluation

To evaluate the performance of the proposed adaptive algorithm, A3S, we perform simulation experiments using a discrete event simulator written in C++. In Sections 4.1 to 4.4, we consider that tasks arrive following a Poisson process at a rate , service times follow an exponential random variable with average $1/ = 1$ hour, lifetimes also follow an exponential random variable with average $1/$, and boot up and activation times are constant (which we have con rmed with real-life measurements). In Sections 4.5 and 4.6 we rely on real traces for the arrival and service times. Experiments are repeated until the con dence intervals (not depicted for clarity) fall below 1% of the average. We further assume that the migration of a task requires the transmission and reception of 1 MB of information, which is done with an ef ciency of 1.9 MBpJ and 2 MBpJ, respectively [16]. This results in a migration cost of 1.02 J/task.

Throughout our experiments, we consider three con gurations for the server farm, each of them able to process up to $S \quad T = 128$ simultaneous tasks with different server types and power management policies:

Nano servers This deployment is composed of $S = 32$ nano servers such as, e.g., the Raspberry Pi, each supporting up to $T = 4$ simultaneous tasks. The time to boot up one server is $1/ = 30$ seconds and the average lifetime of a server is $1/ = 24$ hours. Following the measurements in [19], we take $P_{idle} = 3$ W and $P_{load} = 2$ W.[5]

Rack servers This deployment is composed of $S = 8$ servers such as, e.g., Intel Core i7-4470 with one socket, 4 cores, 8MB L3 cache and 16 GB memory, each of them supporting up to $T = 16$ simultaneous tasks. The time to boot up one server is $1/ = 2$ minutes and the average lifetime of a server is $1/ = 1$ week. In this con guration, we assume no sophisticated power management technique, and therefore the servers are either completely powered off or active. Following the numbers reported in [13], we take $P_{idle} = 30$ W, and $P_{load} = 40$ W.

Rack servers with ACPI This is the same deployment as before but assuming a power management based on ACPI. More speci cally, we assume that when unused, servers can be moved immediately to the lowest-consuming S_4 state, and brought back to the active mode after a wake time (shorter than the boot time). While this technique consumes more energy than directly switching off servers, it precludes frequent de/activations which might affect the lifetime of the server. Following [20], we take $P_{sleep} = 49$ W and a wake time of 48 s.

4.1 Failure probability

We start our evaluation by assessing the effectiveness of A3S to drive the system to the desired point of operation. To this end, we consider the con gurations described above, the target failure probabilities $T_f = f10^{-3}; 10^{-5}g$, and different values of the arrival rate . For each setting, we evaluate via simulation the failure probability P_f provided by A3S after more than 10^5 slots, and compare it against the target T_f. To model the service time, we consider three different random variables with the same average: an exponential distribution (short-tailed), a log-normal distribution with parameters $= 1$ and $= 2:85$ (long-tailed), and a uniform distribution between zero and twice the average (no tail). The results, depicted in Fig. 3, show that in all cases the resulting P_f values perfectly match the desired T_f, which con rms the effectiveness of A3S in providing the target reliability levels.

4.2 Power consumption

We next assess the ef ciency resulting from A3S in terms of power consumption. To this end, we repeat the same experiment as in the previous section and measure the average power ! consumed by the server farm when using A3S. To

5. Note that this type of servers do not allow for different sleep modes, so the only way to deactivate a server is to switch it off.

Fig. 3: Failure probability vs. for two target values, 10^{-3} (top) and 10^{-5} (bottom), and different service times.

Fig. 4: Power consumption vs. . Lines represent the consumption of A3S. Points give the consumption resulting from performing an exhaustive search and selecting the best configuration.

determine if the algorithm is effective in minimizing power consumption, we compare it against the result of performing an exhaustive search over the $\{s_m\}$ configuration space and selecting the setting that meets the target T_f while minimizing the power consumption. Note that, while such an exhaustive search is unfeasible in a practical setting, it provides a benchmark for the optimal performance. Results are provided in Fig. 4 for the same settings as before.

According to the results, there are very small differences between the power consumption resulting from A3S (lines) and that of the exhaustive search configuration (points). Indeed, while the exhaustive search provides slightly smaller consumption, the difference is well below 1% in almost all cases. This confirms that A3S is effective in minimizing power consumption. As expected, power consumption increases with . With rack servers, this increase is steeper given the relatively higher value of P_{load} as compared to P_{idle}. Along the same lines, the more stringent the target failure probability T_f, the higher power consumption, as this requires on average a higher number of active resources. Note that we do not provide the power consumption provided by the RL algorithm since this is very similar to the one obtained using A3S.

For the set of parameters considered, it is more energy efficient to support a service with nano servers than with rack servers. This is somehow counter-intuitive as the nano servers are less reliable and hence one may expect that they are less suitable to support very low failure probabilities. However, their low power consumption allows them to efficiently provide highly reliable services by activating a sufficient number of servers. The use of ACPI results in an even larger power consumption, despite the faster activation times, due to the extra energy consumption in the sleeping mode.

4.3 Controller setting

The objective of the setting of the K_p and K_i parameters provided in Section 3.3 is to achieve a proper trade-off between stability and speed of reaction to changes. To verify this, we analyze the evolution of the controlled signal $c(t)$ over time for the "rack servers" configuration with a target failure probability of $T_f = 10^{-3}$ and a load of $= 0.2$ tasks/min. Assuming as initial conditions $c(0) = 0$, i.e., the thresholds at their highest values, we depict in Fig. 5 the instantaneous value of $c(t)$ after each slot, for three configurations of the controller parameters $\{K_p; K_i\}$, namely:

- The proposed configuration given by (13) and (14), denoted as '$K_p; K_i$' (middle subplot).
- A configuration of these parameters ten times larger than the proposed values, denoted as 'K_p 10; K_i 10' (top subplot).
- A configuration ten times smaller, denoted as 'K_p=10, K_i=10' (bottom subplot).

The figure confirms that the proposed configuration provides a very good trade-off between speed of reaction to changes and stability. An overly large setting of these parameters (top subplot) turns the system unstable, with strong oscillations across the desired point of operation of the system that brings the activation thresholds from very small values to very large ones. On the other hand, an overly small setting (bottom subplot) leads to very long convergence times; in the figure, the 'K_p=10; K_i=10' configuration does not reach the steady state. In contrast to these two settings, with the proposed configuration the control signal $c(t)$ reaches the desired point of operation fairly quickly (marked with a vertical dashed line in the figure) and has minor variations around the desired point of operation. We have repeated the experiment for different configurations of and , with the results being very similar. Based on this, we conclude that our control theoretic analysis is effective in providing a good configuration for $\{K_p; K_i\}$.

We have repeated the above experiment for $T_f = 10^{-5}$. We have observed that in that case, convergence time (i.e.,

Fig. 5: Behavior of the control signal for different con gurations of the controller parameters.

the time until $c(t)$ reaches a stable value) is 54 slots, while for $T_f = 10^{-3}$ convergence time was of 40 slots (as can be seen in Fig. 5). Given that the convergence time is very similar in both cases when measured in slots, this means that convergence is about 100 times more slower for $T_f = 10^{-5}$ than for $T_f = 10^{-3}$, as the slot duration of the former is 100 times longer. This is in line with intuition: since A3S learns from failures and with $T_f = 10^{-5}$ there are 100 time less failures, it is to be expected that in that case it takes 100 more time to learn.

In sum, the presented results show that A3S converges very quickly in terms of controller iterations, as it reaches steady-state just in a few tens of iterations, where an iteration of the controller corresponds to a slot. In terms of time, however, convergence becomes slower as the target failure probability decreases. We argue that this is unavoidable: since an adaptive algorithm necessarily learns from failures, it will need more time to learn as the failure rate decreases.

4.4 Comparison vs. reinforcement learning

We next compare the performance of A3S against a state-of-the-art technique based on reinforcement learning (RL). To this end, we adapt the approach proposed in [21], which uses Q-learning to minimize the available resources while ensuring a given SLA. The state s_t of the system is de ned by the number of tasks n and the number of active servers m, and the action a_t is chosen from three possibilities: switch on a server, switch off a server, or keep the current number of active servers. At each update step , which we set to $1=2$ (half the inter-arrival time), we observe the current state s_t and choose action a_t, resulting in a penalty p_{t+1}. The Q-values for the state and actions are updated following the usual rule:

$$Q(s_t; a_t) = (1 \quad)Q(s_t; a_t) + \quad p_t + \min_{a_{t+1}} Q(s_{t+1}; a_{t+1}) \; ; \tag{15}$$

while the penalty is given by the a weighted sum of the number of failures during the update interval n_t and the power consumption over the same period $!_t$, i.e.,

$$p_t = \; n_t + (1 \quad)!_t : \tag{16}$$

The different parameters for the Q-learning approach are set as follows. The discount factor is set to 0.9, and the factor is set to 0.9 to provide the same failure probability as A3S (which we set to $T_f = 10^{-3}$).[6] To compute the Q-values we follow standard techniques in RL [22] and employ an -greedy strategy with an exploration parameter $= 0:1$, while the parameter is set to 10^{-4} to obtain a meaningful average of the values.

Next, we analyze the convergence time of the RL mechanism. To this aim, we focus on the "rack servers" scenario, with a load of $= 0:2$ task/min. We depict in Fig. 6 the fraction of Q-values that have converged over all the Q-values visited (discarding those with a very low number of visits). Given that the exploration period should be long enough to allow the Q-values to converge, we set it to 40 000 slots, since at this point more than 90% of the values have converged.[7] Fig. 7 depicts the failure probability P_f of the RL approach for this exploration period (using a moving average for ease of visualization) and compares it against A3S. The gure shows that RL takes much more time to converge: while A3S already meets the target failure probability after 40 slots, Q-learning only reaches T_f after 40 000 slots (i.e., when nishing the exploration phase).

The above shows that Q-learning is not an appropriate tool for the problem addressed in this paper, as it takes far too much time to converge and thus risks not meeting the failure probability target over long transients. Note that, while we have focused on the behavior of one particular Q-learning approach, we would expect that any other technique based on reinforcement learning would likely suffer from similar issues, as the system addressed by this paper is a fairly complex system relying on many states.

4.5 Stationary real life traces

The design of A3S does not rely on any assumption on the task and server models, and hence should work for any task and server behavior. To con rm this, we perform a similar experiment to the one of Section 4.2 but instead of exponential inter-arrivals and service times, we take the task arrival and service times from the "Work ow Trace Archive Google trace" [24]. In particular, we select three 24-h periods from those traces with traf c load $I = = = f30,$ 44, 64g Erlangs which, given the capacity of our system of $S \quad T = 128$, correspond to small, medium and high load, respectively. For each of these traces, we consider the

6. Note that Q-learning cannot be con gured a priori to provide a target T_f since there is no clear relation between the parameters and the resulting failure probability. In this paper, we have performed an exhaustive search on the con guration space for the sake of this comparison, but this approach is clearly impractical.

7. While in theory the exploration period should be such that the majority of Q-values have converged (as we have done here), in practice this period is typically set following a rule of thumb criterion (e.g., with trial and error). In any case, even when following a rule of thumb criterion, for systems with many states such as the one here, the exploration period needs to be set to very large values to ensure that the system behaves well in steady state (see e.g. [23]).

Load	A3S P_f	A3S ! (W)	Static P_f	Static ! (W)
Small	10^{-3}	516.1	$0.3 10^{-3}$	665.2
	10^{-5}	687.4	$0.8 10^{-5}$	710.8
Medium	10^{-3}	727.5	$0.4 10^{-3}$	773.5
	10^{-5}	880.8	$0.4 10^{-5}$	942.3
High	10^{-3}	1 020.3	$0.5 10^{-3}$	1 062.1
	10^{-5}	1 212.8	$0.8 10^{-5}$	1 232.5

TABLE 2: Performance with stationary real-life traf c

Fig. 6: Convergence of the Q-values for the Reinforcement-Learning (RL) technique: percentage of Q-values that have converged as a function of time.

Fig. 7: Comparison of A3S and RL: failure probability over time. The gure shows that A3S converges much more quickly than Reinforcement-Learning (RL).

target failure probabilities $T_f = f10^{-3}; 10^{-5}g$ and evaluate the resulting failure probability and power consumption provided by A3S for the `rack servers' con guration. As a benchmark, we consider the static optimal con guration provided by our previous work [25], which assumes Poisson arrivals and exponential service times. We summarize the obtained results in Table 2.

The results con rm the good performance of A3S under realistic traf c loads, as in all cases the obtained failure probability coincides with the target one. In contrast, the optimal con guration assuming Markovian arrivals and departures results overly conservative, leading to P_f smaller than the target T_f which results in wastage of resources. Furthermore, the approach of [25] requires estimating in advance the average arrival rate and service times, which limits its practicality. Based on these results, we conclude that A3S is well suited for stationary practical scenarios.

Fig. 8: A3S with multiple instances.

4.6 Non-stationary real life traces

Next, we analyze the performance of A3S under non-stationary real-life traf c, taking the multiple instance approach presented in Section 3.4. We take as input a trace from a large “Alibaba” production cluster [26], spanning a period between July and August of 2020, with an average arrival rate of = 9.2 tasks/min and an average service time of 1= = 61 min/task. The task arrival pattern, which is illustrated in Fig. 8 for one week, shows the “rough diurnal patterns” of the trace, with some distinction between weekdays and weekends.

Given the traf c load of the trace (I = 570 Erlangs), we have to redimension the server farms. To this aim, we increase the number of servers S to ensure that the number of resources available is enough to serve the traf c with no failures when all servers are active ($T_{on} = 0$),[8] which results in 75 rack servers or 300 nano servers. We then evaluate the following two con gurations of A3S in terms of the number of instances:

- Two instances: we run two instances of A3S, one for the 0-12 h period, and another one for the 12-0 h period. They are denoted as `1' and `2', respectively, in the rst row at the bottom of Fig. 8.
- Three instances: we run three instances of A3S, one instance for the weekends (instance `3' at the bottom row) and two instances for the weekdays like in the previous case (instances `1' and `2').

We assess the performance of each of the above multiple instance con gurations of A3S, as well as for the

8. By ensuring that no failures happen with all servers active, all failures produced during the execution of A3S are caused by the con guration of the activation thresholds.

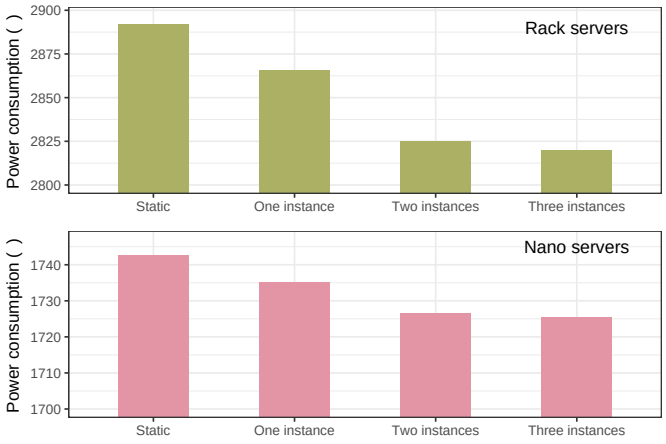

Fig. 9: Performance with non-stationary real-life traffic.

configuration with only one instance, for a target failure probability $T_f = 10^{-4}$, and provide the resulting average power consumption in Fig. 9. Like in the previous case, we also provide as a benchmark the performance of the static optimal configuration provided by [25]. Like in the previous case, the figure confirms that A3S provides a more efficient operation than a static configuration (which, in contrast to our approach, requires estimating the values of and). Furthermore, it also illustrates that the use of more instances of A3S can result in a more efficient operation, since they provide additional energy savings as compared to the use of one instance. We observe that there are some gains when moving from one to two instances, while the gains from moving to three instances are very small; this suggests that having two instances may be a good choice for A3S.

5 Related Work

5.1 NFV and reliability

The softwarization of networks introduces a great deal of flexibility to deciding where to instantiate a VNF, and how to interconnect them. This has motivated a lot of work on network function placement, see e.g. the survey [27], with the aim of maximizing resource efficiency while providing *mild* service guarantees (e.g., CPU availability, inter-VNF delays). For instance, [28] presents a mixed integer linear programming to minimize the power consumption while guaranteeing traffic constraints which optimizes the VNF locations. In contrast to these works, we propose an adaptive algorithm for auto-scaling that ensures strict service guarantees in terms of failure probability. For the particular challenge of providing a reliable service, [29] presents some heuristics for a routing optimization problem, where redundant VNFs are used to ensure an average reliability level. Another contribution also addressing the reliability of a softwarized deployment is that in [30], but in that case, the focus is on the understanding of the interconnection of virtual and physical resources and identifying when a single node failure results in large-scale network collapse. The scope and objective of these papers are different from ours: here, our adaptive algorithm targets a data center and provides hard guarantees on reliability.

5.2 Analytical modeling of reliability

The seminal work of [31] analyzes the reliability of a blade server system. This is done with a high-level fault tree model that interconnects a number of lower-level Markov models, which serve to account for the fallibility of the various hardware modules such as e.g. the CPU or the memory. A similar approach is followed in [32], where authors analyze the reliability of a virtualized and non-virtualized system composed of two hosts. A related system is later considered in [33], where a *sensitivity analysis* is performed to identify the parameters that more critically affect reliability. A closer work to ours is [34], where a Markov chain is used to model a server farm with setup delays in terms of the response time and its power consumption. A related analysis in the context of 5G/6G networks is presented in [1], where thresholds are used to power up and down instances, and the performance is characterized in terms of power consumption and waiting time. None of these works takes into account the fallible nature of the servers, which is essential in order to meet the very stringent requirements of 5G/6G networks. In a previous work, we studied a similar system to the one analyzed in this paper in [25] by extending the analysis of [1], [34] to take into account fallible servers. Based on the analysis, [25] characterizes service reliability and derives an optimal policy for the configuration of the server farm. The key differences between [25] and this paper are: (*i*) the approach of [25] relies on strong assumptions on the arrival processes and service times, while our approach works for general arrival processes and service times, and (*ii*) the policy derived in [25] is static and cannot dynamically adapt to changing conditions, while here we design an adaptive algorithm based on control theory. Finally, other approaches such as parallel queues with vacations [35] do not fit our problem as they do not include a policy to (de)activate servers depending on the number of tasks.

5.3 Adaptive auto-scaling of server farms

The dynamic management of cloud resources to reduce consumption has received notable attention from the research community. For instance, in [36] authors assess the impact of different static algorithms to (de)activate resources and reallocate tasks in a data center focusing on energy consumption and service violations. In a follow-up paper [37], they propose to adapt the thresholds to the estimated conditions. A key difference between our work and these cloud techniques [7] is that they cannot provide high levels of reliability, such as the ones required in 5G/6G. Some approaches have been proposed that use of machine learning to tune the auto-scaling of server farms. The work in [21] considers a similar scenario to ours, with a global manager that switches on/off physical machines using a Q-learning approach. However, it targets a penalty function given by a weighted sum of power consumption and SLA violations and cannot easily be tuned to provide strict reliability guarantees. Furthermore, convergence times can be extremely large. Another Q-learning approach has been proposed in [38] for a similar purpose, although based on a more complex penalty function, which suffers from similar issues. Finally, control theory has been used in the past to perform energy-efficient resource allocation in cloud

computing systems (see [39] for a survey). However, the techniques proposed address different problems than the ones we focus on in this paper and none of them focuses on systems with fallible servers and very high-reliability requirements. For instance, [40] leverages control theory to perform load balancing and to select the CPU frequency.

6 Conclusions

In light of the very stringent reliability requirements of 5G/6G and the network operators' concerns on energy consumption, in NFV it is very critical to derive appropriate scaling techniques that power on resources in advance in order to provide the desired levels of reliability while keeping the overall power consumption as low as possible. Given the high variability of traffic load in mobile networks, such scaling techniques need to automatically adapt to the current network conditions. To that end, an adaptive algorithm is needed that, depending on the observed behavior, adjusts the scaling parameters to dynamically optimize performance. In this paper, we have addressed this problem by relying on control theory. We have designed a closed-loop system that measures the observed failure probability and increases/decreases the aggressiveness of scaling in order to drive the system to the target reliability, thus minimizing power consumption. The performance evaluation conducted for our approach shows that the proposed is effective in providing the desired reliability while minimizing power consumption. Equally importantly, the system reacts as quickly as possible to changing conditions, minimizing the impact of transients. This contrasts with the performance of a widely used reinforcement learning technique, whose reaction is much slower. This shows that, in spite of the high popularity of machine learning, other techniques that may be more suitable in problems where convergence times are critical.

Acknowledgements

This work has been partly funded by the European Commission through the H2020 project Hexa-X (Grant Agreement no. 101015956), by NEC Laboratories Europe Student Research Fellowship program of 2021, and by the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D SORUS project. The work of Jorge Ortin was funded in part by the University of Zaragoza through project UZ2022-IAR-08, in part by the Gobierno de Aragon through Research Group under Grant T31_20R, in part by the European Social Fund (ESF), and in part by Centro Universitario de la Defensa under Grant CUD-2023_14. The work of Jaime Garcia-Reinoso has been partly funded by the Spanish Ministry for Science and Innovation through the ADMINISTER (TED2021-131301B-I00) project.

References

[1] Y. Ren, T. Phung-Duc, J. Chen, and Z. Yu, "Dynamic Auto Scaling Algorithm (DASA) for 5G Mobile Networks," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM 2016)*, Washington DC, USA, Dec. 2016.

[2] M. Gramaglia, P. Serrano, A. Banchs, G. García, A. Garcia-Saavedra, and R. Perez, "The case for serverless mobile networking," in *Proceedings of IFIP Networking 2020 - Network Slicing workshop*, Paris, France (virtual conference), Jun. 2020.

[3] European Telecommunications Standards Institute (ETSI), "Network Functions Virtualisation (NFV) Release2; Management and Orchestration; Functional requirements specification MANO Functional Rqmts Spec." GS NFV-EVE 010, v. 2.4.1, 2019.

[4] 3rd Generation Partnership Project (3GPP), "Technical Specification Group Services and System Aspects; Management and orchestration; Provisioning," TS 28.531, v. 15.0.0, 2019.

[5] ——, "Service requirements for the 5G system," TS 22.261, v. 17.3.0, 2020.

[6] A. Gonzalez, P. Gronsund, K. Mahmood, B. Helvik, P. Heegaard, and G. Nencioni, "Service availability in the nfv virtualized evolved packet core," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[7] P. Singh, P. Gupta, K. Jyoti, and A. Nayyar, "Research on Auto-Scaling of Web Applications in Cloud: Survey, Trends and Future Directions," *Scalable Computing: Practice and Experience*, vol. 20, pp. 399–432, May 2019.

[8] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, and R. Subbiah, "Worth their watts? - an empirical study of datacenter servers," in *Proceedings of the Sixteenth International Symposium on High-Performance Computer Architecture (HPCA 2010)*, Bangalore, India, Jan. 2010.

[9] Y. Jin, Y. Wen, and Q. Chen, "Energy efficiency and server virtualization in data centers: An empirical investigation," in *Proceedings of IEEE INFOCOM Workshops 2012*, Orlando, FL, Mar. 2012, pp. 133–138.

[10] GSM Association. Energy Efficiency: An Overview, available at https://www.gsma.com/futurenetworks/wiki/energy-efficiency-2/. [Online]. Available: https://www.gsma.com/futurenetworks/wiki/energy-efficiency-2/

[11] C. Zhang, P. Patras, and H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2224–2287, Mar. 2019.

[12] Gutierrez-Estevez, D. M. et al., "Artificial Intelligence for Elastic Management and Orchestration of 5G Networks," *IEEE Wireless Communications*, vol. 26, no. 5, pp. 134–141, Aug. 2019.

[13] C. Gu, Z. Li, H. Huang, and X. Jia, "Energy efficient scheduling of servers with multi-sleep modes for cloud data center," *IEEE Transactions on Cloud Computing*, 2018.

[14] VMware vSphere vMotion, "Architecture, Performance and Best Practices in VMware vSphere 5: Performance Study," Technical White Paper, 2019.

[15] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete Container State Migration," in *Proceedings of the 37th IEEE nternational Conference on Distributed Computing Systems (ICDCS 2017)*, Jun. 2017, pp. 2137–2142.

[16] J. A. Aroca, A. Chatzipapas, A. F. Anta, and V. Mancuso, "A measurement-based characterization of the energy consumption in data center servers," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2863–2877, 2015.

[17] P. Patras, A. Banchs, and P. Serrano, "A control theoretic scheme for efficient video transmission over ieee 802.11e edca wlans," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 8, no. 3, Aug. 2012. [Online]. Available: https://doi.org/10.1145/2240136.2240142

[18] A. Garcia-Saavedra, A. Banchs, P. Serrano, and J. Widmer, "Adaptive mechanism for distributed opportunistic scheduling," *IEEE Transactions on Wireless Communications*, vol. 14, no. 6, pp. 3494–3508, 2015.

[19] F. Jalali, R. Ayre, A. Vishwanath, K. Hinton, T. Alpcan, and R. Tucker, "Energy consumption comparison of nano and centralized data centers," in *Proceedings of ACM Greenmetrics*, Austin, TX, Jun. 2014.

[20] K. N. Khan, Z. Ou, M. Hirki, J. K. Nurminen, and T. Niemi, "How much power does your server consume? Estimating wall socket power using RAPL measurements," *Computer Science-Research and Development*, vol. 31, no. 4, pp. 207–214, Aug. 2016.

[21] S. Telenyk, E. Zharikov, and O. Rolik, "Modeling of the Data Center Resource Management Using Reinforcement Learning," in *Proceedings of the International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T 2018)*, Kharkiv, Ukraine, Oct. 2018, pp. 289–296.

[22] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, pp. 1–8.

[23] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Deepcog: Cognitive network management in sliced 5g networks with deep learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 280–288.

[24] Google, "Workflow Trace Archive Google trace [Data set], available at https://doi.org/10.5281/zenodo.3254540," Jun. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3254540

[25] J. Ortin, P. Serrano, J. Garcia-Reinoso, and A. Banchs, "Analysis of scaling policies for nfv providing 5g/6g reliability levels with fallible servers," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1287–1305, 2022.

[26] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters," in *19th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 22)*, 2022.

[27] Xin Li and Chen Qian, "A survey of network function placement," in *Proceedings of the 13th IEEE Annual Consumer Communications Networking Conference (CCNC 2016)*, Las Vegas, NV, Jan. 2016, pp. 948–953.

[28] A. N. Al-Quzweeni, A. Q. Lawey, T. E. H. Elgorashi, and J. M. H. Elmirghani, "Optimized Energy Aware 5G Network Function Virtualization," *IEEE Access*, vol. 7, pp. 44 939–44 958, Mar. 2019.

[29] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A Reliability-Aware Network Service Chain Provisioning With Delay Guarantees in NFV-Enabled Enterprise Datacenter Networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, p. 554–568, Sep. 2017.

[30] J. Liu, Z. Jiang, N. Kato, O. Akashi, and A. Takahara, "Reliability evaluation for nfv deployment of future mobile broadband networks," *IEEE Wireless Communications*, vol. 23, no. 3, p. 90–96, Jun. 2016.

[31] W. E. Smith, K. S. Trivedi, L. A. Tomek, and J. Ackaret, "Availability analysis of blade server systems," *IBM Systems Journal*, vol. 47, no. 4, pp. 621–640, 2008.

[32] D. S. Kim, F. Machida, and K. S. Trivedi, "Availability modeling and analysis of a virtualized system," in *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, 2009, pp. 365–371.

[33] R. d. S. Matos, P. R. M. Maciel, F. Machida, D. S. Kim, and K. S. Trivedi, "Sensitivity analysis of server virtualized system availability," *IEEE Transactions on Reliability*, vol. 61, no. 4, pp. 994–1006, 2012.

[34] A. Gandhi, M. Harchol-Balter, and I. Adan, "Server farms with setup costs," *Performance Evaluation*, vol. 67, no. 11, pp. 1123 – 1138, Nov. 2010.

[35] P. Wartenhorst, "N parallel queueing systems with server breakdown and repair," *European Journal of Operational Research*, vol. 82, no. 2, pp. 302–322, 1995.

[36] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 577–578.

[37] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, ser. MGC '10. New York, NY, USA: Association for Computing Machinery, 2010.

[38] S. Horovitz and Y. Arian, "Efficient cloud auto-scaling with sla objective using q-learning," in *Proceedings of the 6th IEEE International Conference on Future Internet of Things and Cloud (FiCloud 2018)*, Barcelona, Spain, 2018, pp. 85–92.

[39] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, S. U. Khan, and A. Zomaya, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," *Computing*, vol. 98, no. 7, pp. 751–774, Jun. 2014.

[40] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Power-efficient response time guarantees for virtualized enterprise servers," in *2008 Real-Time Systems Symposium*, 2008, pp. 303–312.

Jesús Pérez-Valero got his B.Sc. and M.Sc. degrees in 2019 and 2020, respectively, from the Universidad Politécnica de Cartagena. He is currently a Ph.D. candidate at the Universidad Carlos III of Madrid. His main research interests lie in the performance analysis and optimization of communication systems.

Albert Banchs has a double affiliation as Professor at the University Carlos III of Madrid and Deputy Director of the IMDEA Networks institute. Prof. Banchs has served in many TPCs and has also served in the editorial board of a number of journals, including IEEE Transactions in Wireless Communications and IEEE/ACM Transactions on Networking. Dr. Banchs has participated in many European projects and industry contracts.

Pablo Serrano (M'09, SM'16) is an Associate Professor at the University Carlos III de Madrid. His research interests lie in the analysis of wireless networks and the design of network protocols and systems. He has over 100 scientific papers in peer-reviewed international journals and conferences and has served on the TPC of many conferences. He currently serves as Editor for IEEE Open Journal of the Communication Society.

Jorge Ortín is an Associate Professor at the Centro Universitario de la Defensa Zaragoza, Spain. He received his MS degree in Telecommunications and his Ph.D. degree from the Universidad de Zaragoza in 2005 and 2011 respectively. In 2012 he joined the Universidad Carlos III of Madrid as a postdoc research fellow. From 2013, he works at Centro Universitario de la Defensa Zaragoza. His research interests focus on the optimization of communications systems.

Jaime Garcia-Reinoso (M'04) received the Telecommunications Engineering degree in 2000 from the University of Vigo, Spain and the Ph.D. in Telecommunications in 2003 from the University Carlos III of Madrid, Spain. He is an associate professor at University of Alcala, Spain since 2021 and he has published over 60 papers in top magazines and conferences. He has been involved in many international projects on next generation networks, 5G, SDN and NFV.

Xavier Costa-Perez Xavier Costa-Pérez is ICREA Research Professor, Scientific Director at the i2cat Research Center and Head of 5G Networks R&D at NEC Laboratories Europe. His research focuses on the transformation of society driven by the interplay of mobile networks and AI. Currently, he is serving as Associate Editor at IEEE Transactions on Mobile Computing, IEEE Transactions on Communications and Elsevier Computer Communications journals.