Performance Trade-offs of Auto Scaling Schemes for NFV with Reliability Requirements

Jesus Perez-Valero[a,*], Jaime Garcia-Reinoso[b], Albert Banchs[a], Pablo Serrano[a], Jorge Ortin[c], Xavier Costa-Perez[d,e,f]

[a]*Departamento de Ingenieria Telematica, Universidad Carlos III de Madrid,* 28911*, Spain*
[b]*Automatics Department, Universidad de Alcala, Spain*
[c]*Centro Universitario de la Defensa, Academia General Militar, 50090 Zaragoza, Spain*
[d]*NEC Laboratories Europe, 69115 Heidelberg, Germany*
[e]*i2cat, 08034 Barcelona, Spain*
[f]*ICREA, 08010 Barcelona, Spain*

Abstract

A major concern for future mobile networks is to meet extremely stringent reliability requirements without incurring very high energy consumption. To this aim, we need auto-scaling techniques to switch on/off servers on demand, ensuring that they are active when required. One key challenge is to keep the right number of active servers while accounting for both the fallibility of servers and their non-zero boot up time: if this number is too small we risk service disruption, but if it is too large resources will be wasted. In this paper, we analyze this challenge by assessing the performance of different strategies to support an energy-efficient highly-reliable service, which range from deploying few blade (i.e., reliable) servers to deploying many nano (i.e., unreliable) servers. Our main take-away message is two-fold: (1) a server farm of nano servers can provide a more energy efficient operation than a farm of blade servers, but (2) it involves a very dynamic operation both in terms of task migrations and (de)activations, which challenges the endurance of the hardware.

Keywords: B5G, NFV, Reliability, Trade-offs

1. Introduction

To support services such as autonomous driving or industrial automation, the 3rd Generation Partnership Project (3GPP) introduced in 5G the ultra-reliable low-latency communication (URLLC) service, with reliability levels that typically span between $1 - 10^{-4}$ (four nines) and $1 - 10^{-6}$ (six nines). In 5G, the concept of network slicing [1] allows the instantiation of isolated, independent and service-based logical networks over a common infrastructure. For instance, URLLC services are deployed over URLLC slices. To be more precise, network slicing is supported via the Network Function Virtualization (NFV) paradigm, which facilitates the virtualization and interconnection of network functions over commodity servers with cloud techniques.

*Corresponding author
Email addresses: jesperez@pa.uc3m.es (Jesus Perez-Valero), jaime.garciareinoso@uah.es (Jaime Garcia-Reinoso), banchs@it.uc3m.es (Albert Banchs), pablo@it.uc3m.es (Pablo Serrano), jortin@unizar.es (Jorge Ortin), xavier.costa@neclab.eu (Xavier Costa-Perez)

Thus, a service in 5G is implemented as an ordered sequence of virtual network functions, running along other slices on top of a common physical infrastructure. Furthermore, in the case of URLLC services, such physical infrastructure is usually deployed at the edge of the network, where resources are typically scarce. Hence, selecting the most appropriate hardware to cope with the stringent requirements imposed by 5G services, while reducing the capital an operational expenditures, is key for network operators.

To reach the reliability levels required by URLLC services with virtualization techniques, resources need be scaled on demand to (*i*) guarantee their availability, while (*ii*) making an efficient use of the resources. While there are many cloud computing methods to match demand and active resources, existing approaches cannot guarantee the reliability levels considered in 5G or 6G (see e.g. [2] for a recent survey). To achieve these one cannot neglect that (1) servers are fallible, i.e., they can occasionally (albeit very infrequently) go down, and (2) servers take non-zero time to boot up.

On the one hand, redundancy can be used to address fallibility, i.e., keep enough additional resources activated so in case the health of a server degrades, the tasks can be seamlessly migrated to a different machine without disruption. This seamless "live migration" is already supported with existing technologies, both for functions involving state [3] and for those supporting a stateless operation (which makes the migration simpler) [4].

On the other hand, highly reliable servers involve boot up times in the order of minutes as they include the so called "power-on self test" [5]. Because of this, server farms need to be configured to anticipate the traffic demand and trigger the activation of additional resources with enough time in advance, such that requests do not have to wait before being served. Finally, another (and perhaps more important) challenge is how to design the farm: is it better to rely on very reliable servers or to rely on smaller and less reliable servers? The former have very low failure probabilities (and hence need less redundancy) but involve longer boot up times (thus requiring more anticipation).

In this paper, we analyze the above issues in a realistic setting with real traces and different server designs. We describe the challenge of providing a stringent reliability service under two real-life constraints: the need to scale resources to ensure an energy efficient operation, and the fallibility of the resources. Our main contributions and take-away messages are as follows:

- We conduct trace-driven simulations with traces from real-world systems to analyze the impact of the server configuration on performance.
- We show that when properly configured, 5G reliability levels can be achieved with both very reliable and less reliable servers.
- We find out that by employing less reliable servers, we can achieve substantial energy savings, at the price of increasing task migrations and server switching rates.
- We illustrate the ways in which distinct (and sometimes conflicting) variables interact under different scenarios, unveiling the various trade-offs that appear between these variables.

Our paper provides insights about the various trade-offs and performance issues when different server configurations are used to provide a very reliable service, taking into account variables typically overlooked such as the lifetime of the servers. In this way, our contribution paves the way for several new lines of research, such as the design of server farms targeting high reliability, or the minimization of costs for a given service level agreement. The rest of this paper is organized as follows. Section 2 presents the most important challenges operators face in their

data centers and, more specifically, the trade-offs between resources and reliability. Section 3 introduces the server farm model, the real-life traces considered, the different deployment configurations, and the performance figures of interest. In Section 4, we analyze the impact of server configuration on the performance figures, summarizing the main results of the paper. In Section 5 we present the related work, while Section 6 concludes the paper presenting the future work.

2. Challenges when scaling resources while ensuring reliability

One of the chief concerns of operators nowadays is the electricity bill of network operation. The softwarization of the network functions has exacerbated the energy consumption of data centers, since virtualized servers consume more energy than physical servers [6]. A well-known technique to reduce the consumption of server farms is the use of scaling techniques to adapt the amount of active resources to the demand. These involve the (de)activation of resources as needed, which for the case of the activation involves a non-negligible non-zero time, i.e., the start-up time. This variable has little impact with traditional scaling schemes that guarantee "coarse" performance figures (e.g., average queueing delay), but it cannot be neglected when dealing with the high reliability requirements of several 5G applications. For instance, in the 3GPP vision for URLLC, reliability requirements for a service range from $1 - 10^{-4}$ (factory automation) to $1 - 10^{-6}$ (process automation), and industrial sources aim for even more stringent values (see e.g. [7] and references therein). For these applications, the scaling algorithm has to anticipate the activation of resources to prevent that an incoming request has to wait until a resource has booted up (note that to achieve such levels of reliability, all composing elements of a network slice have to provide even higher guarantees).

Another key parameter is the granularity of the resource: on the one hand, to ensure an efficient operation, the granularity should be high (i.e., small quantum of resources), to accurately match supply and demand, which would lead to frequent resource (de)activations and eventually fatigue the electronics; on the other hand, a large resource quantum would reduce this fatigue, but worsen the overall efficiency. While traditional data centers used powerful, high capacity servers (i.e., small granularity), the recent trend of deploying data centers using small or even nano servers (e.g., the Raspberry Pi [8]) has changed this assumption.

The use of small or nano servers motivates analyzing another key aspect, namely, the server fallibility. In fact, the fallibility cannot be ignored even with very reliable servers, since e.g. an availability of 99.999% requires a downtime shorter than 315 sec per year. To tackle this fallibility, redundant servers need to be preemptively activated so a task can be moved from a failing server into a new one.

In what follows, we analyze the trade-offs when providing a highly reliable service under different configurations of the server farm. These configurations consider different capacity per server, different start up times, and different mean time to failure (MTTFs).

3. System Model

3.1. Server farm description

The system we consider in this paper is a server farm used to deploy virtual network function(s) to instantiate URLLC network slices in 5G (we assume a scenario where the management and orchestration function [9] receives requests to deploy URLLC network services, where some virtual network functions have to be deployed in a particular server farm, close to the user). We

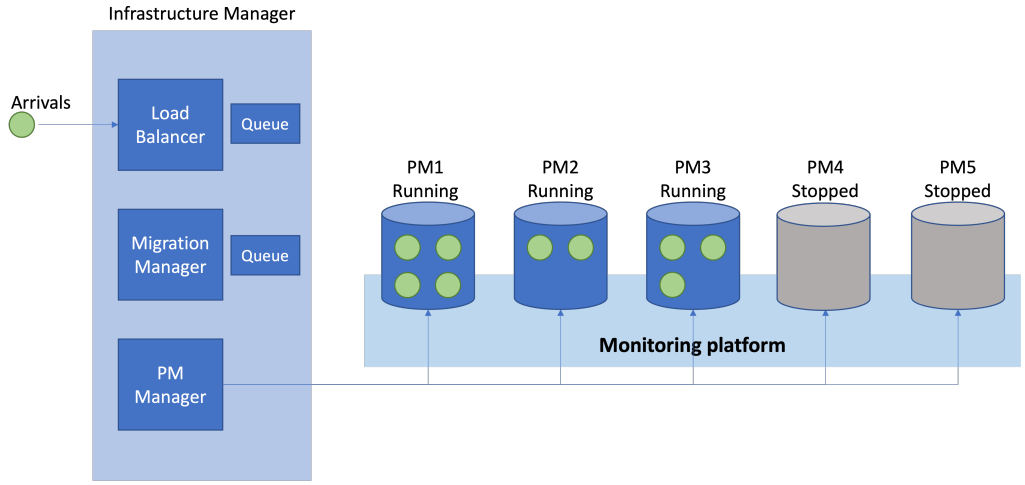

Figure 1: Server farm considered.

depict in Fig. 1 the system, which is composed by an infrastructure manager (IM) and a number of physical machines (PMs). The IM acts as: (1) load balancer, in charge of accommodating tasks arriving at the system, or holding them until there are enough available resources; (2) migration manager, keeping a backup of the status of the tasks to support their migration across PMs; and (3) PM manager, powering on and off the PMs following the policy described next. Regarding the capacity of a single PM, we assume that all tasks can be provided with the required computational resources as long as their number does not exceed a given threshold (the server capacity).

To handle these tasks, a subset of the servers is running (servers PM1, PM2, and PM3 in Fig. 1), while the remaining servers are activated when needed. The IM decides the number of servers that should be active at a given point following a simple occupation-based threshold policy:

if either the CPU occupation, defined as the sum of the CPU requested by all active tasks over the total active CPU capacity, or the RAM occupation, defined as the sum of the RAM requested by all active tasks over the total active RAM capacity, is above a given upper threshold (denoted as T_{on}), a new PM is activated; if with the deactivation of a PM both the occupation of CPU and the occupation of RAM would still fall below this threshold, tasks are moved accordingly (with seamless migrations) and a PM is deactivated. Given that our focus is to illustrate the different performance trade-offs, for simplicity we will assume a single threshold to trigger the (de)activation of servers, as otherwise the space parameter would grow unnecessarily.

Upon a task arrival, requesting a certain amount of CPU and RAM, the IM decides which of the active PMs with enough resources should handle the task.

If there are no PMs available to handle an arriving task, the IM requests the activation of a PM, holding the task in a queue until there are enough resources to serve it. We consider this as a service disruption, as the task has to wait in the queue.

Since the PMs are fallible, an active PM can crash at any point in time. In case it was serving one or more tasks, we assume that as long as there are other active PMs with resources available, these tasks can be seamlessly migrated to a different PM without disruption. The

reasons for this are two-fold. First, following state-of-the-art technology, we consider that a server failure can be anticipated with enough time in advance (via monitoring tools such as e.g. Nagios[1] or DOCTOR[2]), to perform a seamless migration using life migration tools available with `OpenStack`,[3] `VMWare` [10], or containers [11]. Second, in case failures could not be anticipated, we assume that the tasks can be seamlessly migrated by periodically backing up their state and using technologies such as ACHO [3]. Even if the time to boot up a new VNF is strictly non-zero, this time is typically very small (less than a second for the case of containers) and negligible as compared vs. the time to boot up a physical machine.

We next illustrate the operation with a toy example illustrated in Fig. 1, where the server farm is composed of five PMs, each supporting up to four tasks, and one infrastructure manager (IM). For the sake of clarity, we assume homogeneous tasks, all requesting one fourth of the CPU and RAM offered by each server. As depicted, at the beginning three PMs are running, two are stopped, and there is a total of 9 tasks (represented as green circles inside PMs). When a new task arrives to the system, the IM will select the most appropriate PM to process it (e.g., PM2) and will place it accordingly. Given the change in the load, assuming $T_{on} = 10/12$, the IM will also trigger the booting of PM4. In the meantime, and before PM4 is fully operational, the monitoring platform may alert the PM manager about the incipient crash of PM1 (e.g., memory issues), triggering the migration of the four tasks served by this PM. However, since PM4 is still booting up, only two out of the four orphan tasks can be seamlessly migrated to the two available slots at PM2 and PM3. As discussed above, this migration is immediate, thanks to light-weight migration tools [11, 3] or the use of stateless VNFs [12, 4]. The other two orphan tasks will be placed in the migration manager (MM) queue, causing a service disruption. Only after PM4 has completed its boot up, these two tasks can be migrated from the MM queue to the new active PM.

3.2. Input traces

To emulate the arrival time, service time, and CPU/RAM requirements of tasks, we rely on the Workflow Trace Archive (WTA) [13] to generate different input traces for our simulator. More specifically, we consider three traces:

- Very bursty traffic: obtained from the first `csv` file from Google trace [14] of the Workflow Trace Archive (WTA). Each task is specified with an arrival timestamp and a runtime, which ranges between 0.2 minutes and 199.8 minutes. It expands a period of approx. 29 days with approximately 90 000 tasks. The burstiness index of this trace, computed as suggested in [15], is 0.9299. Because when the burstiness value equals 1 the signal is considered the most bursty one, we denominate this trace as very bursty traffic.

- Bursty traffic: this is obtained by removing those entries with the same timestamp from the previous trace. Although these simultaneous arrivals might correspond to sudden spikes of simultaneous arrivals, they might also occur due to clock drifts, since according to the trace description "timestamps are approximate, although reasonably accurate". The burstiness index of this trace equals 0.6960, which means that it is still bursty.

[1] `https://www.nagios.org` (online; accessed 30-Aug-2023)

[2] `https://wiki.opnfv.org/display/doctor` (online; accessed 30-Aug-2023)

[3] `https://docs.openstack.org/nova/pike/admin/live-migration-usage.html` (online; accessed 30-Aug-2023)

- Smoothened traffic: based on the above trace, we compute the average arrival rate and service time, and generate a process with Poisson arrivals with the same rate and exponential runtimes with the same average service time. The burstiness of this trace equals 0, so we can consider this trace neutral or smooth.

We note that the above traces do not correspond to URLLC slices: since we could not find any open dataset for this traffic type, we had to rely on the above dataset that corresponds to processing tasks in a cloud server. To map these traces into an URLLC scenario, we interpret the arrival rate λ as the rate at which requests for new URLLC slices arrive into the system, while we interpret the service time $1/\mu$ as the duration of such slices. For the three traces the traffic parameters are relatively similar, with the average arrival rate being 2.1 tasks/min, and the average service time being 36.9 min/task, which results in the traffic load of 78.3 Erlangs. With respect to the CPU and RAM requested, the three traces have similar values, so only the statistical values of the very bursty traffic is shown in Table 1. These values are normalized with respect to the capacity of the Google servers. Although these tasks may correspond to data analysis tasks (and not e.g. an autonomous traffic service), they can also be used to illustrate the challenge to provide a highly reliable service in the context of 5G. For instance, we can assume that the duration of a task corresponds to the duration of the traffic flow from an autonomous guided vehicle that requires a timely and reliable processing to feed the actuators (see e.g. [16]).

	Min.	Max.	Mean	Std. Dev.
CPU	0.001	0.25	0.0322	0.0259
RAM	$1.56 \cdot 10^{-5}$	0.5088	0.02611	0.0236

Table 1: CPU and RAM requirements

3.3. Performance figures

In our analysis we focus on the following performance figures:

- **Failure probability**: probability that a task is ever disrupted. A disruption can occur upon arrival, if resources were not activated with enough anticipation, or during service, if after a PM crash there are no available resources to accommodate the tasks from that PM.
- **Migration probability**: probability that a task is seamless migrated from a PM to a different PM. While these migrations cause no disruption to the task, still they have a certain overhead and should be minimized.
- **Switch on rate**: defined as the number of times a server is switched on over time. The larger this number, the more stress to the electronics of the servers.
- **Power consumption**: the rate at which energy is consumed. We next detail its computation.

We assume that the average power consumed by the server farm is given by three terms:

$$\omega = \omega_s + \omega_a + \omega_m \tag{1}$$

where the term ω_s corresponds to the power consumption caused by the serving of the tasks, the term ω_a represents the power consumption due to the activation of the servers, and the term ω_m represents the extra power consumption due to task migrations.

We assume that a deactivated PM consumes zero power, while following the usual power consumption models [17, 18], the consumption of an active PM is given by two terms: one fixed and one proportional to its load. Following this, given an average number of active servers N_a, and an average load ρ the power consumption ω_s is given by

$$\omega_s = N_a \left(P_{\text{idle}} + \rho \cdot (P_{\text{peak}} - P_{\text{idle}}) \right) \tag{2}$$

where the terms P_{idle} and P_{peak} correspond to the power consumption terms of idle and peak power consumption, respectively.

To compute ω_a, we assume that during its activation, a server reaches its peak power rate. In this way, if we denote the server activation rate as Λ and the activation time $1/\alpha$, ω_a can be computed as

$$\omega_a = \Lambda P_{\text{peak}} 1/\alpha \tag{3}$$

To compute ω_m, we first compute the energy cost of a migration, following the measurements reported in [19] that characterize the energy cost of reading and writing from a server, denoted as E_r and E_w, respectively. Given an average migration of λ_m tasks over time, ω_m can be computed as:

$$\omega_m = \lambda_m (E_r + E_w) \tag{4}$$

we assume that the migration of a task requires the transmission and reception of all the RAM requested by such task, which is done with an efficiency of $E_w = 1.9$ MB/J and $E_r = 2$ MB/J, respectively [19].

3.4. Deployment configurations

To analyze the impact of the server configuration on performance, we consider three reference deployments, each one characterized by a different server type with a different lifetime. We denote the deployment with the most reliable servers as "carrier grade," the deployment with the least reliable servers as "consumer grade," and the third deployment as "enterprise grade." We next detail their configuration.

Carrier grade: this farm is composed by servers that are modeled after a Dell Power Edge server equipped with 32 GB of memory. According to the Power Consumption Database (TPCDB) [20], this type of server consumes 270 W at its peak power and 150 W while in idle mode (note that these numbers are in line with those of the least consuming servers analyzed in [18]). According to our own measurements, repeated 10 times, a server boots in $1/\alpha = 3$ minutes, while following [21], we assume that the average server mean time to failure (MTTF) is 768 h (i.e., 32 days). Following the usual assumptions in the literature [22], we assume that this time follows an exponential random variable.

Enterprise grade: for this deployment we assume a server with a smaller form factor, such as an Intel NUC8i5BEH. Although they are usually provided with 4 GB of RAM memory, we assume that it is extended to 8 GB or RAM. Following the TPCDB, it consumes 78 W at its peak power and 5.1 W while in idle, and according to our own measurements it boots in 8 seconds. Since we could not find any publicly available data on its MTTF, for simplicity we assume that it is half of that of a carrier-grade server, i.e., 16 days (note that we will analyze the impact of the MTTF during our performance evaluation).

Consumer grade: finally, we assume a third type of server deployment, composed of Raspberry Pi 4 Model B. According to our measurements repeated 10 times, it boots in 18 seconds, and following the same reasoning as above, we assume a MTTF that is half the one of a mini PC server, i.e., 8 days. Finally, the server consumes 4.6 W while idle and 7.6 W at full operation, according to the TPCDB.

With the above, we have defined the three server types that will constitute the server farms. We next address the dimensioning of these farms, i.e., the computation of the total number of servers that will be used to serve the traces, with the objective being that the three types provide relatively similar reliability. To this aim, we proceed as follows. We set a target reliability level of $1 - 10^{-5}$, i.e., a middle value between the two discussed requirements for URLLC according to the 3GPP vision. We first assume that all servers are always active, i.e., set the activation threshold to zero, since this is the most reliable configuration. Then, we start with a configuration where the total farm capacity (number of servers times capacity per server) is strictly above the 78.3 Erlangs of input traffic, and compute the total number of failures after 300 simulation runs. If the resulting reliability is below than the target value $1 - 10^{-5}$, we increase the number of servers by one and repeat the process.

We provide in Table 2 the resulting values of the total number of servers for the very bursty input traffic[4]. We also provide in the table a summary of the parameters that characterize each server type, an estimation of the relative total cost of each server farm (based on prices obtained during early 2022), and the maximum peak power consumption. It is worth remarking that the Consumer deployment has the smallest cost and peak power consumption, while providing the same failure-free performance as the other two deployments. Comparing these, the Carrier deployment duplicates the cost of the Enterprise deployment, while its peak power consumption is less than half of the latter.

	Carrier	Enterprise	Consumer
RAM Capacity (GB)	16	8	8
CPU	Intel Xeon Silver 4310	Intel Core i5-8259U	BCM2711
CPU Capacity (Relative)	424	46	1
Power consumption (P_{idle})	150 W	5.1 W	4.6 W
Power consumption (P_{peak})	270 W	78 W	7.6 W
Boot time ($1/\alpha$)	3 min	8 s	18 s
Mean time to failure (MTTF)	32 days	16 days	8 days
Total # servers (N)	13	104	208
Total cost (relative)	22.1x	11.4x	1
Peak power consumption	3.5 kW	8.1 kW	1.6 kW

Table 2: Configurations considered

[4]The CPU capacity is based on the CPU mark values available on https://www.cpubenchmark.net/cpu_list.php (online; accessed 30-Aug-2023)

4. Performance evaluation

In this section we assess the performance of the three server farms considered for different configurations of the activation threshold T_{on}, considering the four metrics described in the previous section. Our motivation is to characterize the different trade-offs in performance for each of the server types, to help understand the (dis)advantages of each configuration over the others. To this aim, we have written a discrete event simulator in C++ which closely follows the behavior of the server farm, which is then fed by the trace files described above. We have implemented two different mechanisms to select the most appropriate PM to accommodate a given task, among all PMs with enough resources: the most loaded server or the server with more available resources. Because the results are similar, we only present results using the latter approach. A detailed explanation of the simulation, including its source code, is publicly available.[5] Each result is computed as the average of at least 50 simulation runs, as we repeat the simulations (using different random seeds) until at least 20 failures are detected to ensure a certain accuracy of the results. In all cases the relative length of the resulting confidence intervals is less than 1%, and therefore we do not plot them in the figures for readability reasons.

4.1. Power consumption versus failure probability

We start by analyzing the different trade-offs between performance and resource consumption that each server configuration provides. To this aim, we perform a sweep in the activation threshold T_{on} from 0 (most consuming) to 100% (least consuming) in steps of 5%. For each value of T_{on}, we compute the corresponding failure probability (denoted as P_f) and average power consumption (denoted as ω) during the whole simulation, following the methodology described above. We depict all the resulting pairs (ω, P_f) in Fig. 2, where we use lines to connect these pairs (note that the leftmost values correspond to the $T_{on} = 0$ configuration and, given our methodology to dimension the server farm described in Section 3.4, they are all below 10^{-5}) and we use different linestyles to represent different tracefiles: dots and lines for very bursty, straight line for bursty, and dashed for smoothened. We also use different colors for different server configurations: blue for carrier, orange for enterprise, and green for consumer.

The results illustrate that there are different trade-offs between power consumption and reliability, which depend on the server farm configuration and the burstiness of the input load. Regarding the impact of the server farm, the results show that both the enterprise-grade and the consumer-grade servers can provide very high levels of reliability at the smallest power consumption, while the carrier-grade configuration demands more power for all levels of reliability.

Regarding the impact of the trace burstiness, the results are qualitatively similar across the last two farm configurations, where the burstiness does not affect the results. On the other hand, the carrier-grade configuration is more sensitive to the burstiness.

The reason for this slope difference in the carrier-grade configuration is as follows. On the one hand, the smoothened tracefile does not contain traffic spikes that trigger the activation of servers, and therefore the consumption is practically the same regardless of the configuration of the activation thresholds, since the scaling algorithm can easily adapt to the mild variations in the traffic rate.

On the other hand, the very bursty tracefile has arrival bursts that require the activation of all servers to accommodate the requests. But when $T_{on} > 0$, these resources are not active when the

[5] https://github.com/jpvalero/Server-farm-simulator

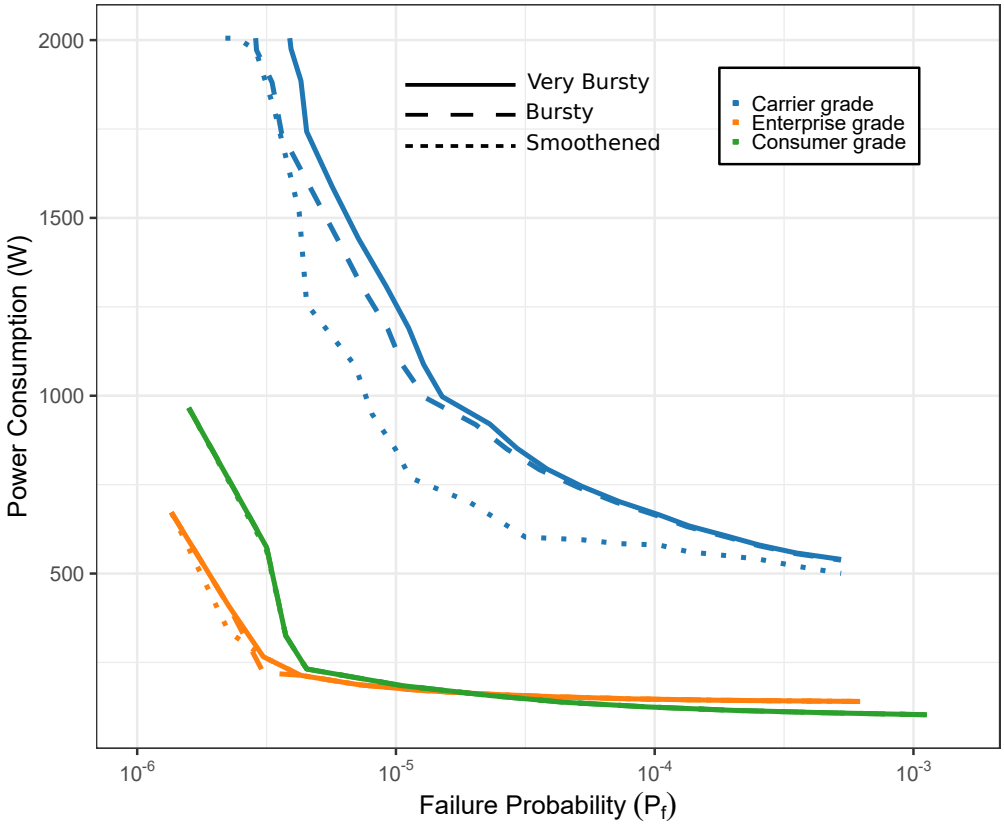

Figure 2: Power consumption vs. failure probability.

requests arrive, so they have to wait in the queue. This leads to a reduced power consumption but increases the failure probability, as illustrated by the slope of the line.

Finally, the slope of the bursty traffic sits between these two cases. For the rest of the paper, for simplicity we will focus on this tracefile.

4.2. Impact of threshold configuration

We next analyze the impact of the configuration of the activation threshold on performance. To this aim, we plot in Fig. 3 the failure probability P_f vs. configured activation threshold T_{on} for the same experiments as before. The figure confirms that, when the activation threshold is set to zero, all three deployments provide a failure probability that is below 10^{-5}. It also illustrates that the failure probability grows with the activation threshold, since the larger the value of T_{on}, the smaller the "slack" active servers have to accommodate either new arrivals or tasks from a failing server until a new server is powered on.

It is also worth remarking the semi-log behavior of P_f with T_{on} (note the log scale on the y-axis). This behavior can be related to that of an M/M/c queue [23], where the waiting probability (computed with the Erlang-C), i.e., the probability that a task is not immediately served, decreases in a semi-log manner with the number of resources. In our case, the failure probability

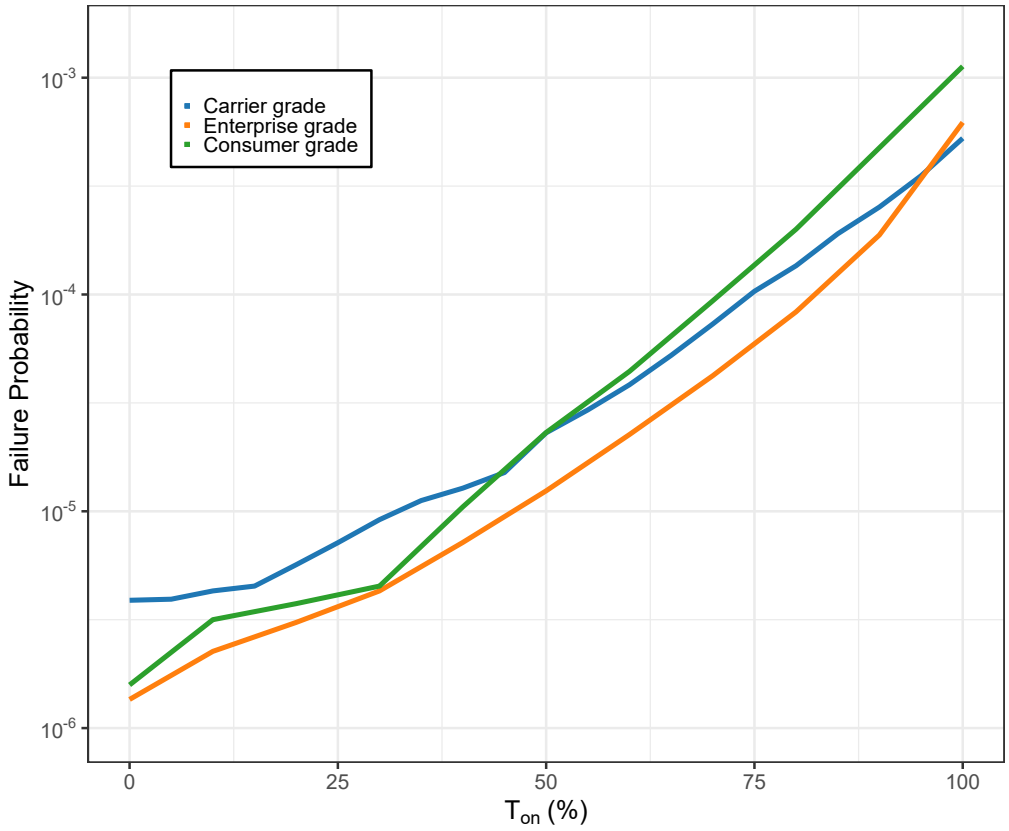

Figure 3: Failure Probability vs. activation threshold.

increases with T_{on}, since $1 - T_{on}$ corresponds to the relative number of immediately available resources at a given point in time.

4.3. Switch on frequency

We next analyze the switch on rate per server, i.e., the rate at which a single server is powered on. This performance figure is very relevant, since continuously switching on/off a server might dramatically reduce its lifetime and increase the maintenance costs. We depict in Fig. 4 the average number of activations per server for the three server farm configurations versus the resulting failure probability. According to the results, the most reliable configuration results in the smallest switch on rate (i.e., bottom left part of the figure), but the relation between these two variables is not straightforward. We next analyze this relationship in detail.

As mentioned, the switch on rate is almost zero for the smallest value of the failure probability. The reason is that this case corresponds to the $T_{on} = 0$ configuration (note the mapping between P_f and T_{on} of Fig. 3), i.e., all servers are always active. Under this configuration, the only activations are caused by the (infrequent) server crashes. Although having both the failure probability and the switch on rate at their minimum is obviously desirable, we note that this configuration results in the largest power consumption (see Fig. 2). Then, the switch on increases

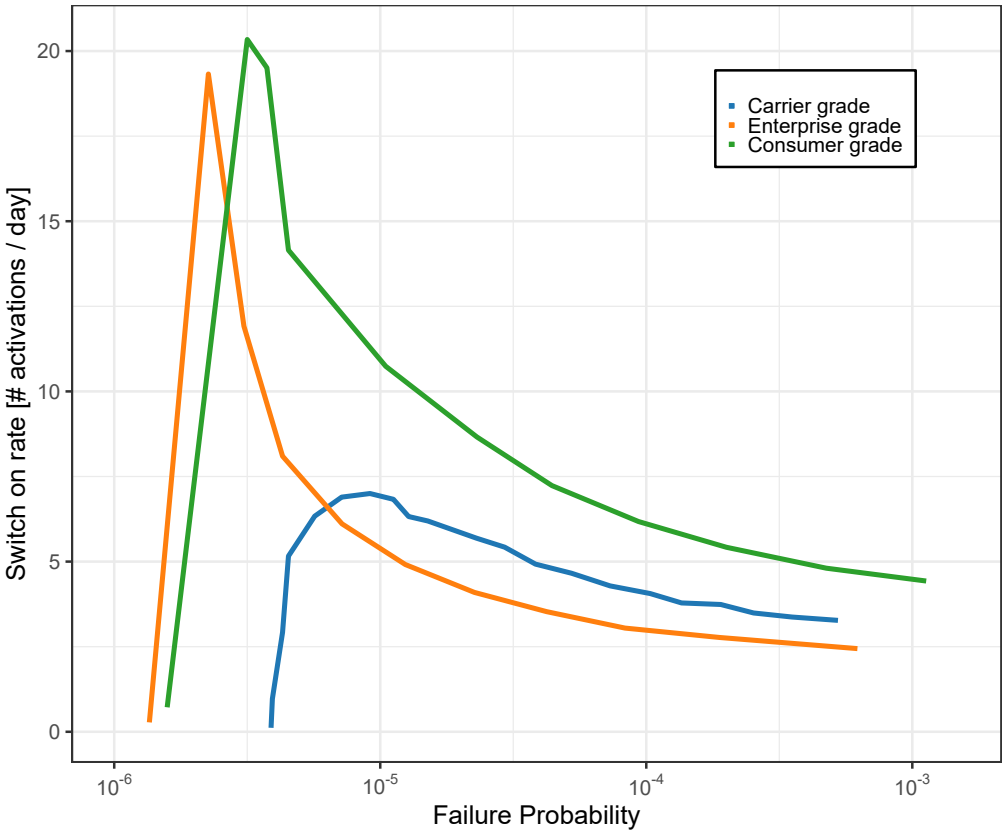

Figure 4: Switch on rate vs. failure probability.

as the failure probability increases. This is caused by a more dynamic operation of the system caused by the increase of T_{on}: there are more activations (and deactivations), following the load over time, but in certain cases the activations are not fast enough to accommodate the arrivals (hence the increase in P_f). However, although P_f increases with T_{on}, at some point the switch on rate starts to decrease, since for larger values of T_{on}, only for relatively large bursts of traffic the threshold is surpassed. For these cases, given the time to activate the resources, the requests are placed in a queue until additional resources are available —but it is likely that a request finishes its service before a server booted up, and therefore the activation of additional resources is no longer required. It is also worth noting the step-like behavior of the consumer grade deployment, caused by the small capacity per server, which results in a sever rounding of the configuration threshold T_{on}: for instance, with two consumer grade servers (i.e., a total capacity of four), there are only four actual activation thresholds.

Finally, it is also worth remarking the relative performance of the three configurations for larger failure probabilities, with the consumer-grade having the highest switching rate. This is due to the relatively shorter MTTF and the smaller capacity of these servers, which triggers a more dynamic behavior.

On the other hand, although the enterprise-grade configuration has a high switch on rate for

very small failure probabilities, this configuration drastically reduces the switch on rate for larger failure probabilities, going below the carrier-grade configuration. Since continuously switching on/off a server might dramatically reduce its lifetime and consistently increase the associated maintenance costs, the carrier-grade results the most desirable configuration under this scope.

4.4. Migration probability

Finally, we analyze the probability that a task experiences a seamless migration, i.e., it is moved from one PM to a different PM, either because of a server crash or because the need to empty a server prior to its deactivation. Although these migrations do not cause disruptions, they involve a certain complexity and associated resource consumption (e.g., power consumption migrating tasks ω_m), which motivates quantifying their prevalence. To this aim, we measure the number of times that a task is moved to a different server after a crash (n_c), and the number of times that a task is moved to a different server to prepare a deactivation (n_d) (please notice that a task could be migrated several times and due to any of these two events, so we could talk about a migration ratio instead of migration probability, but for the sake of simplicity, we will keep on using the term probability). We also add these numbers to count the total number of seamless migrations (n_t). By dividing n_c, n_d, and n_t over the total number of served tasks, we can estimate the probability that a task is migrated because of a server crash, because of a deactivation, and in total, respectively. We illustrate these probabilities in Fig. 5.

The figure illustrates that the migration probabilities are greater than the failure probabilities, a result that confirms the importance of seamless live migrations to support ultra reliable services. Regarding the migration probability due to crashes (top figure), the figure illustrates that they are practically independent of the activation threshold, except for very high values of T_{on}. The reason for this is that, when T_{on} is high, it is more likely that there are no additional resources to accommodate those tasks moved from a crashing server. In contrast, for small values of T_{on}, the probability that another server is available is practically one, and therefore the migration probability after a crash is approximately constant and given by the product of the total number of servers, the server crashing probability, and the average number of tasks per server.

The migration probability caused by a deactivation (middle figure) increases with T_{on}. The reason for this behavior is that although the switch on rate has a non-strict behavior with the activation threshold (see Fig. 4), the number of affected tasks increases with T_{on}. It is important to notice that the migration probability due to deactivations is two orders of magnitude greater than due to crashes for all three server configurations, so the probability of any type of migration (or total migration probability), illustrated in Fig. 5 (bottom) is almost exactly the same as this figure.

As mentioned above, these results confirm the need to implement seamless migration algorithms to ensure an efficient operation of resource on demand schemes while guarantee high reliability levels.

4.5. Impact of server lifetime on performance

Here we analyze the impact of one of our key parameters, namely, the server lifetime, on performance. More specifically, we are interested in understanding to what extent the ability to provide a reliable service, and its associated power consumption, depend on the quantitative value of the MTTF.

To address the above, we set as requirement a reliability guarantee of 99.9%, and focus on the carrier-grade deployment since it provides the most significant results. Then, we consider

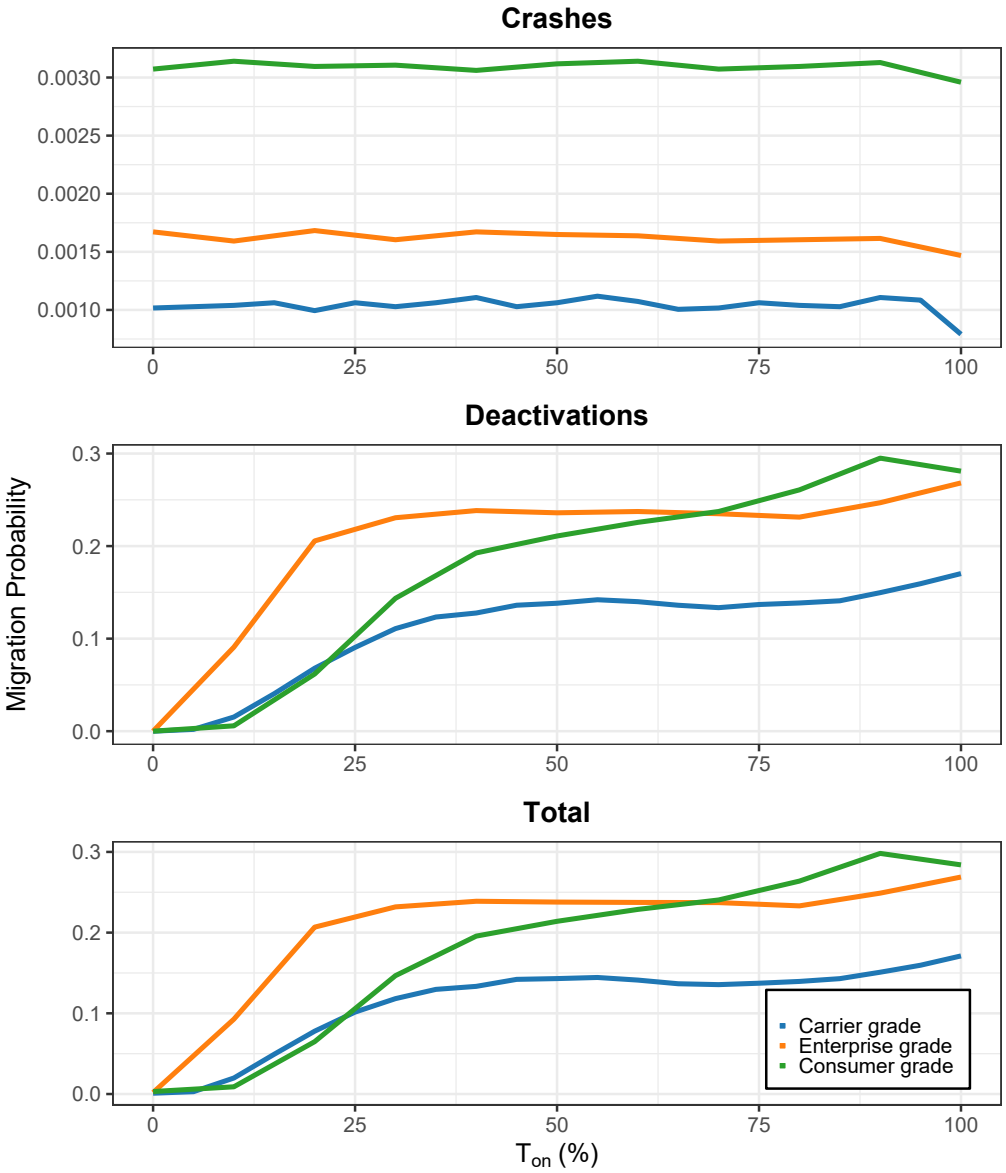

Figure 5: Migration probability vs. activation threshold T_{on}

the scenario with the carrier servers and vary the value of MTTF (originally set to 32 days), in increments or decrements of 10. For each considered value of MTTF, we perform an exhaustive search on T_{on} to find the configuration that minimizes the power consumption while guaranteeing the above reliability level. The results are depicted in Fig. 6, where we mark for each MTTF the value of T_{on} that optimized performance.

According to the results, a MTTF value longer than the assumed value of 32 days does not

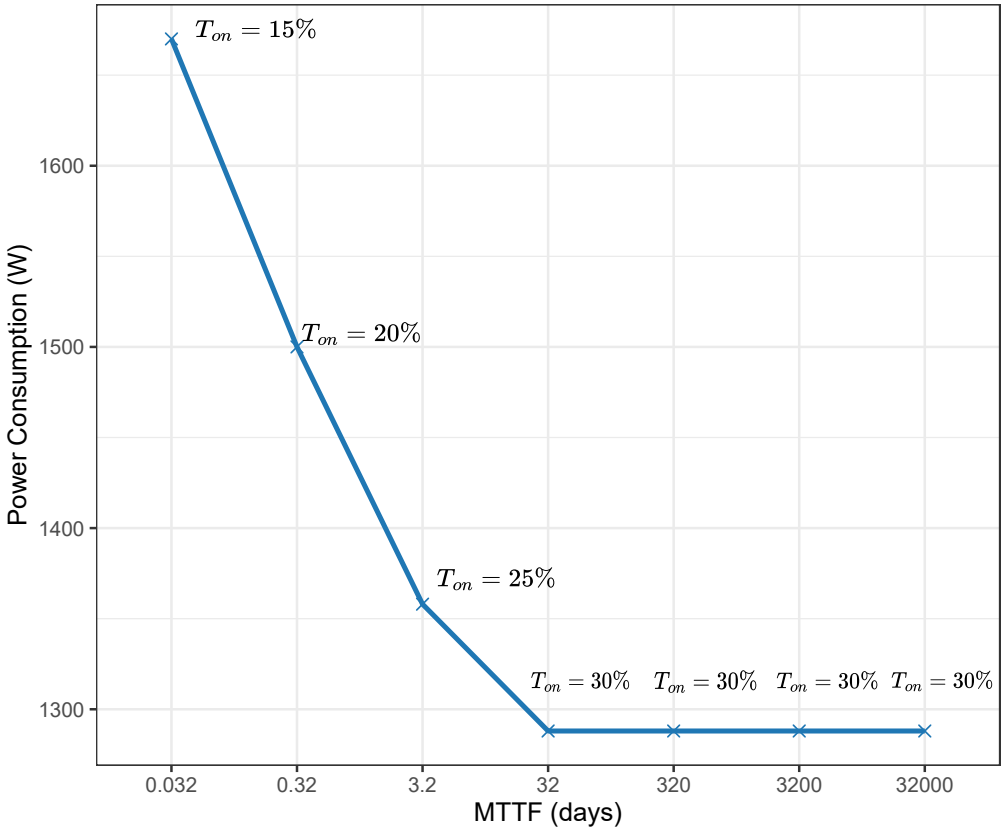

Figure 6: Power consumption vs. lifetime of carrier-grade servers

have any significant impact on the power consumption of the server farm, nor in the optimal value of T_{on}. It would be therefore of little interest to increase the expending on more reliable servers to improve the efficiency of the farm. Regarding decrements in the MTTF, it is worth remarking than with a 10× reduction on the MTTF (i.e., 3.2 days) the same reliability can be guaranteed but the power consumption increases as the T_{on} decreases.

A further reduction of the MTTF also results in a reduction of the T_{on}, but also in an increase of the power consumption of the server farm. Finally, as hinted above, it is worth mentioning that this sensitivity of the results to variations of the MTTF is even smaller for the other two server configurations.

4.6. Impact of replicas on performance

Finally, we are interested in analyzing the performance of a system model where all active PMs used to run tasks have running PMs counterparts, ready to accept all the orphan tasks after a server crash. These empty, but active PMs are denominated replicas. Although there are different degrees of replication, as we are focused on scenarios requiring a high reliability, we will test a 1:1 replica approach, where every active PM has a replica PM.

The differences between the approach using replicas and the previous approach with no replicas, in terms of failure probability and power consumption is the following. With regards to the failure probability, we assume that when a PM fails, its replica will not fail at the same time, or at least during the activation of a new replica for the new active PM. In other words, we assume that the probability of a PM and its replica failing at the same time is negligible. Thus, the failure probability is reduced to one component: the probability that there are not enough resources for an incoming task. On the other hand, the power consumption of the replicas approach is greater than the previous approach. More specifically, one replica has to be active for each active PM, so the P_{idle} is duplicated, and when the IM decides that a new PM is necessary, a replica server has to be activated too, so the ω_a is duplicated too. Following Eq. 1,2,3,4, we can define the power consumed by the replica approach ($\omega_{replica}$) as:

$$\begin{aligned}\omega_{replica} &= \omega_s + \omega_a + \omega_m \\ &= [N_a \cdot P_{idle} + N_a(P_{\text{idle}} + \rho \cdot (P_{\text{peak}} - P_{\text{idle}}))] + [2 \cdot \Lambda P_{\text{peak}} 1/\alpha] + [\lambda_m(E_r + E_w)] \\ &= [N_a \cdot P_{idle} + \Lambda P_{\text{peak}} 1/\alpha] + [N_a(P_{\text{idle}} + \rho \cdot (P_{\text{peak}} - P_{\text{idle}})) + \Lambda P_{\text{peak}} 1/\alpha + \lambda_m(E_r + E_w)] \\ &= [N_a \cdot P_{idle} + \Lambda P_{\text{peak}} 1/\alpha] + \omega_{no\_replica}\end{aligned}$$

We have modified our simulator to implement the replicas approach. Figure 7 presents a comparison between the approach with replicas and the previous approach where replicas were not used. These results are for the enterprise-grade configuration, showing that the replicas approach requires more power thant the previous approach, for a given failure probability. It is also important to notice that the lower the required failure probability, the higher the difference between the power used by the replicas approach. On the other hand, as the replicas approach has a failure probability with one single component, this is an interesting approach when a service requires a extremely high reliability, at the cost of increasing the power consumption.

5. Related work

Several previous proposals have developed algorithms to scale server farms, e.g., heuristics to adapt activation thresholds to the estimated load to reduce service violations [24], or schemes that take as input Service Level Agreements [25, 26] to guarantee coarse metrics such as bandwidth or CPU. Newer approaches follow a similar direction but relying on machine learning to switch on/off physical resources [27, 28]. These approaches, however, do not target the high reliability levels considered in 5G/6G.

Other approaches have tackled network function placement to improve some mild service guarantees: [29] uses linear programming to minimize the power consumption while guaranteeing traffic constrains, [30] presents some heuristics for a routing optimization problem, and [31] analyzes the decomposition mobile applications in several components and their best placement in the edge. In contrast to these works, we focus on a fine-grained performance figure, with the objective of understanding the different trade-offs when deploying a service, including the design of the server farm.

Finally, some analytical approaches have proposed tree models to characterize server reliability based on its physical components [32] or virtualized elements [33]. However, these models do not tackle the reliability of the server farm. Other approaches aim at improving reliability by replicating tasks and selecting the most appropriate server [34, 35], but they do not consider energy consumption

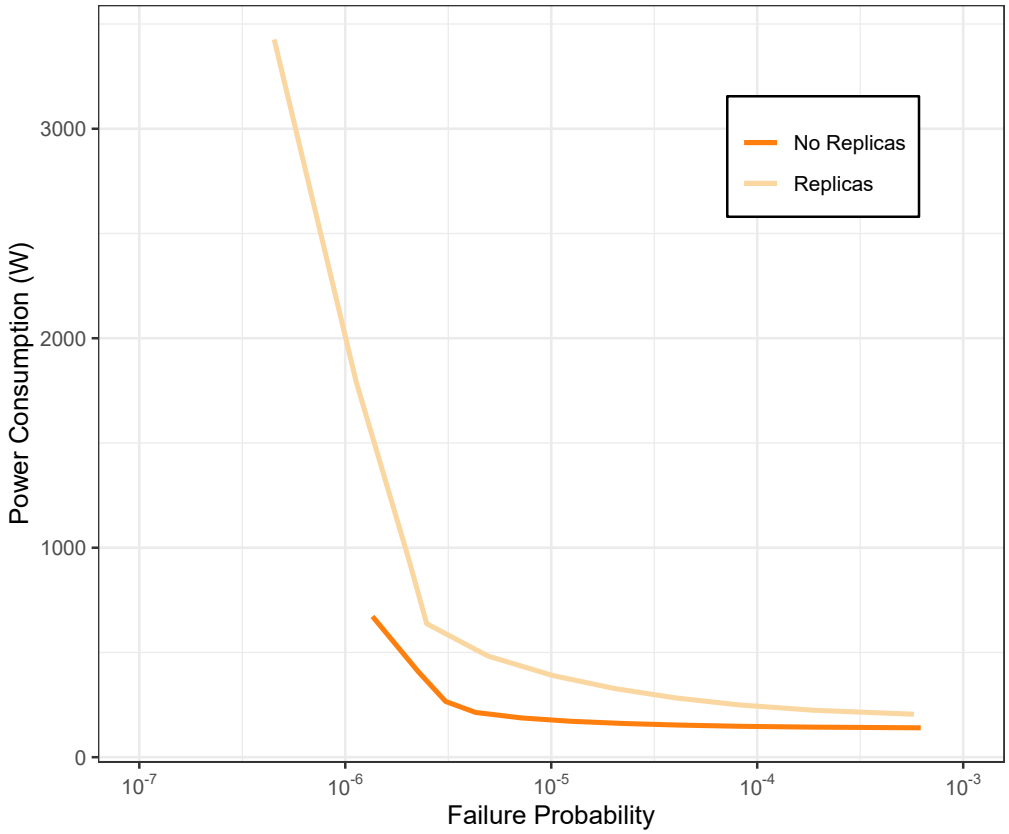

Figure 7: Power consumption vs. failure probability with and without replicas.

In contrast to the above, we focus on analyzing a server farm guaranteeing fine-grained reliability levels considering different performance figures. We take into account real-life features such as non-zero boot times and finite lifetime of the servers themselves, and rely on the use of real-life traces to identify the different trade-offs that appear between the performance figures.

6. Summary and future work

Motivated by the stringent reliability requirements envisioned in 5G and beyond and the need for an energy-efficient operation, we have analyzed different strategies to deploy a server farm to support a reliable service. Our conclusions from the scenarios considered are two-fold: first, a deployment composed of large and robust servers can be outperformed in terms of energy efficiency by a deployment composed of (many) smaller and less reliable servers, thanks to their increased agility and smaller power consumption; second, this smaller energy footprint comes with additional costs, such as the the higher (de)activation rates (which may fatigue the electronics) or migration probabilities across servers (which increases the management and complexity of the system).

As future work, we can identify the following. First, analyze the sensitivity of the results to certain key parameters, such as the fatigue of the electronics or the introduction of more complex resource on demand schemes (including e.g. hysteresis in the threshold). Second, take into account other scenarios in terms of traffic requirements and service guarantees. Third and finally, to formally introduce in our the analysis economic variables such as CAPEX and OPEX, to guide the design of servers depending on the deployment scenario.

Acknowledgment

This work has been partly funded by the European Commission through the H2020 project Hexa-X (Grant Agreement no. 101015956) and by NEC Laboratories Europe Student Research Fellowship program of 2021. This work is also partially supported 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 Jaime Garcia-Reinoso has been partly funded by the Spanish Ministry for Science and Innovation through the ADMINISTER (TED2021-131301B-I00) project. The work of J. Ortin was funded in part by the Spanish Ministry of Science under Grant RTI2018-099063-B-I00, 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-2021_11.

References

[1] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, H. Bakker, Network slicing to enable scalability and flexibility in 5g mobile networks, IEEE Communications Magazine 55 (5) (2017) 72–79. doi:10.1109/MCOM.2017.1600920.

[2] P. Singh, P. Gupta, K. Jyoti, A. Nayyar, Research on auto-scaling of web applications in cloud: Survey, trends and future directions, Scalable Computing: Practice and Experience 20 (2019) 399–432. doi:10.12694/scpe.v20i2.1537.

[3] G. Garcia-Aviles, C. Donato, M. Gramaglia, P. Serrano, A. Banchs, Acho: A framework for flexible re-orchestration of virtual network functions, Computer Networks 180 (2020) 107382.

[4] M. Gramaglia, P. Serrano, A. Banchs, G. Garcia-Aviles, A. Garcia-Saavedra, R. Perez, The case for serverless mobile networking, in: 2020 IFIP Networking Conference (Networking), 2020, pp. 779–784.

[5] Chapter 5 - windows, linux, and macintosh boot processes, in: D. Kleiman, K. Cardwell, T. Clinton, M. Cross, M. Gregg, J. Varsalone, C. Wright (Eds.), The Official CHFI Study Guide (Exam 312-49), Syngress, Rockland, 2007, pp. 265–285.

[6] Y. Jin, Y. Wen, Q. Chen, Energy efficiency and server virtualization in data centers: An empirical investigation, in: 2012 Proceedings IEEE INFOCOM Workshops, IEEE, 2012, pp. 133–138.

[7] P. Popovski, C. Stefanovic, J. J. Nielsen, E. de Carvalho, M. Angjelichinoski, K. F. Trillingsgaard, A.-S. Bana, Wireless access in ultra-reliable low-latency communication (urllc), IEEE Transactions on Communications 67 (8) (2019) 5783–5801. doi:10.1109/TCOMM.2019.2914652.

[8] F. P. Tso, D. R. White, S. Jouet, J. Singer, D. P. Pezaros, The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures, in: 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, IEEE, 2013, pp. 108–112.

[9] ETSI GS NFV 002, Network functions virtualization (NFV); architectural framework v1.2.1, Tech. rep., ETSI (December 2014).
URL https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf

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

[11] S. Nadgowda, S. Suneja, N. Bila, C. Isci, Voyager: Complete Container State Migration, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 2137–2142. doi:10.1109/ICDCS.2017.91.

[12] P. Serrano, M. Gramaglia, D. Bega, D. Gutierrez-Estevez, G. Garcia-Aviles, A. Banchs, The path toward a cloud-aware mobile network protocol stack, Transactions on Emerging Telecommunications Technologies 29 (5) (2018) e3312, e3312 ett.3312. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.3312, doi:https://doi.org/10.1002/ett.3312.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3312

[13] L. Versluis, R. Mathá, S. Talluri, T. Hegeman, R. Prodan, E. Deelman, A. Iosup, The workflow trace archive: Open-access data from public and private computing infrastructures, Vol. 31, 2020, pp. 2170–2184. doi:10.1109/TPDS.2020.2984821.
URL https://doi.org/10.1109/TPDS.2020.2984821

[14] Google, Workflow trace archive google trace (Jun. 2019). doi:10.5281/zenodo.3254540.
URL https://doi.org/10.5281/zenodo.3254540

[15] K.-I. Goh, A.-L. Barabási, Burstiness and memory in complex systems, Europhysics Letters 81 (4) (2008) 48002.

[16] W. Nakimuli, J. Garcia-Reinoso, J. E. Sierra-Garcia, P. Serrano, I. Q. Fernández, Deployment and evaluation of an industry 4.0 use case over 5g, IEEE Communications Magazine 59 (7) (2021) 14–20. doi:10.1109/MCOM.001.2001104.

[17] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, F. Zhao, Energy-aware server provisioning and load dispatching for connection-intensive internet services, in: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI'08, USENIX Association, USA, 2008, p. 337–350.

[18] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, R. Subbiah, Worth their watts? - an empirical study of datacenter servers, in: HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture, 2010, pp. 1–10.

[19] J. A. Aroca, A. Chatzipapas, A. F. Anta, V. Mancuso, A measurement-based characterization of the energy consumption in data center servers, IEEE Journal on Selected Areas in Communications 33 (12) (2015) 2863–2877. doi:10.1109/JSAC.2015.2481198.

[20] The Power Consumption Database, http://www.tpcdb.com, [Online; accessed 17-Feb-2022] (2022).

[21] G. L. Santos, P. T. Endo, G. Gonçalves, D. Rosendo, D. Gomes, J. Kelner, D. Sadok, M. Mahloo, Analyzing the it subsystem failure impact on availability of cloud services, in: 2017 IEEE Symposium on Computers and Communications (ISCC), 2017, pp. 717–723. doi:10.1109/ISCC.2017.8024612.

[22] K. S. Trivedi, A. Bobbio, Reliability and Availability Engineering: Modeling, Analysis, and Applications, Cambridge University Press, 2017. doi:10.1017/9781316163047.

[23] M. Harchol-Balter, Performance Modeling and Design of Computer Systems, Cambridge University Press, 2013.

[24] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, Concurrency and Computation: Practice and Experience 24 (13) (2012) 1397–1420.

[25] C. Dupont, F. Hermenier, T. Schulze, R. Basmadjian, A. Somov, G. Giuliani, Plug4green: A flexible energy-aware vm manager to fit data centre particularities, Ad Hoc Networks 25 (2015) 505–519, new Research Challenges in Mobile, Opportunistic and Delay-Tolerant Networks Energy-Aware Data Centers: Architecture, Infrastructure, and Communication. doi:https://doi.org/10.1016/j.adhoc.2014.11.003.
URL https://www.sciencedirect.com/science/article/pii/S1570870514002376

[26] H. Wang, H. Tianfield, Energy-aware dynamic virtual machine consolidation for cloud datacenters, IEEE Access 6 (2018) 15259–15273.

[27] S. Telenyk, E. Zharikov, 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, 2018, pp. 289–296.

[28] S. Horovitz, 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.

[29] A. N. Al-Quzweeni, A. Q. Lawey, T. E. H. Elgorashi, J. M. H. Elmirghani, Optimized energy aware 5g network function virtualization, IEEE Access 7 (2019) 44939–44958.

[30] L. Qu, C. Assi, K. Shaban, 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 14 (3) (2017) 554–568. doi:10.1109/tnsm.2017.2723090.
URL http://dx.doi.org/10.1109/TNSM.2017.2723090

[31] L. Dong, W. Wu, Q. Guo, M. N. Satpute, T. Znati, D. Z. Du, Reliability-aware offloading and allocation in multilevel edge computing system, IEEE Transactions on Reliability 70 (1) (2021) 200–211. doi:10.1109/TR.2019.2909279.

[32] W. E. Smith, K. S. Trivedi, L. A. Tomek, J. Ackaret, Availability analysis of blade server systems, IBM Systems Journal 47 (4) (2008) 621–640. doi:10.1147/SJ.2008.5386524.

[33] D. S. Kim, F. Machida, 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. doi:10.1109/PRDC.2009.64.

[34] M. Sedaghat, E. Wadbro, J. Wilkes, S. De Luna, O. Seleznjev, E. Elmroth, Diehard: reliable scheduling to survive correlated failures in cloud data centers, in: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2016, pp. 52–59.

[35] M. Korupolu, R. Rajaraman, Robust and probabilistic failure-aware placement, ACM Transactions on Parallel Computing (TOPC) 5 (1) (2018) 1–30.