Analysis of scaling policies for NFV providing 5G/6G reliability levels with fallible servers

Jorge Ortin, Pablo Serrano, *Senior Member, IEEE,* Jaime Garcia-Reinoso, and Albert Banchs, *Senior Member, IEEE*

***Abstract*—The softwarization of mobile networks enables an efficient use of resources, by dynamically scaling and re-assigning them following variations in demand. Given that the activation of additional servers is not immediate, scaling up resources should anticipate traffic demands to prevent service disruption. At the same time, the activation of more servers than strictly necessary results in a waste of resources, and thus should be avoided. Given the stringent reliability requirements of 5G applications (up to 6 nines) and the fallible nature of servers, finding the right trade-off between efficiency and service disruption is particularly critical. In this paper, we analyze a generic auto-scaling mechanism for communication services, used to de(activate) servers in a cluster, based on occupation thresholds. We model the impact of the activation delay and the finite lifetime of the servers on performance, in terms of power consumption and failure probability. Based on this model, we derive an algorithm to optimally configure the thresholds. Simulation results confirm the accuracy of the model both under synthetic and realistic traffic patterns as well as the effectiveness of the configuration algorithm. We also provide some insights on the best strategy to support an energy-efficient highly-reliable service: deploying a few powerful and reliable machines versus deploying many machines, but less powerful and reliable.**

***Index Terms*—Service virtualization, Resource on Demand, Reliability, Energy Consumption**

I. Introduction

The next generation of mobile networks will need to accommodate a large variety of services with very heterogeneous requirements in terms of *performability*, involving performance metrics such as delay or throughput, and dependability attributes such as availability, reliability, or integrity [1], among other Key Performance Indicators (KPIs) [2]. To efficiently support such diverse requirements, mobile networks rely on the *network slicing* paradigm [3], where different independent and isolated virtual instances of a network run over the same physical infrastructure. Each of these network instances is called network slice and consists of a set of interconnected software Virtual Network Functions (VNFs) which typically run over general-purpose servers [4].

To make an efficient use of the resources in a network slicing setting, these need to be re-assigned dynamically, scaling out additional physical resources assigned to the VNFs of a slice when the load increases, and scaling them in when the load decreases [5]. As shown in [6], such a dynamic allocation of resources can enable very high sharing gains, improving significantly the sustainability of a deployment. When activating resources for new VNFs, however, these need to be turned on in advance by anticipating their demands, as the activation of additional resources, possibly involving the deployment of new network instances, is not immediate. In fact, relevant standard specifications by ETSI [7] and 3GPP [8] recommend such practices.

When designing resource allocation algorithms for 5G/6G services, dependability requirements can be very stringent, with e.g. availability levels of 5 nines (99.999%) or more [9], [10]. In fact, position papers for 6G are already pushing for 9 nines [11]. One example of a service with extreme requirements is that of *remote driving*. In this case, the application controlling the vehicles is typically deployed at the edge for latency reasons. This kind of services can be deployed using the Multi-access Edge Computing (MEC) specifications. To ensure the proper driving of the cars, we need to ensure that the MEC server hosting this application has enough resources at all times. Another example is *connected industry*. To provide this service, we need to deploy a virtual network involving (among others) encryption/decryption functions to provide the desired security features. The downtime of such functions has to be reduced below a given threshold to meet the requirements of industrial applications.

Out of the different attributes of dependability, in this paper we focus the **reliability**, defined as the probability of correct service continuity [1], which is a critical metric for dependability. To provide the reliability guarantees required by 5G/6G, we need to make sure that, when a new VNF has to be deployed, there are resources available to host the VNF and that the VNF remains up and running throughout the entire lifetime of the service. While the scaling of resources is a well-known problem in cloud computing, traditional schemes have not been devised to meet such extreme levels of reliability. Indeed, existing scaling algorithms in the literature typically focus on coarse-grained requirements such as, e.g., response times, throughput, or cost [12], which are not appropriate to capture reliability requirements. Furthermore, they typically ignore the fact that servers are *fallible*, i.e., they can eventually crash or go down (even if infrequently). they cannot be ignored when pursuing such 5G/6G reliability levels.

The impact of server failures can be ignored when the reli-

J. Ortin is with Centro Universitario de la Defensa, Academia General Militar, 50090 Zaragoza, Spain.
E-mail: jortin@unizar.es

P. Serrano and A. Banchs are with the Department of Telematic Engineering, Univ. Carlos III de Madrid, 28911 Leganés, Spain.
E-mail: {pablo, banchs}@it.uc3m.es,

J. Garcia-Reinoso is with the Department of Automatic, Univ. of Alcalá, 28801 Alcalá de Henares, Spain.
E-mail: jaime.garciareinoso@uah.es

A. Banchs is also with Institute IMDEA Networks, 28912 Leganés, Spain.

ability requirements are loose,[1] since they yield much smaller failure times. However, they cannot be ignored when pursuing tighter reliability levels. This is particularly critical with the recent trend of deploying data centers using small or even nano servers (e.g., the Raspberry Pi [13]), which originally were not intended for carrier-grade operation. Although some recent works propose the use of redundancy to improve reliability (we discuss them in Section VI), they focus on coarse metrics (e.g., downtime in the order of seconds) and do not allow to configure the reliability level required by the service.

To provide the desired reliability levels, the scaling mechanism needs to keep a sufficient number of active resources, both to anticipate traffic demands (some type of fault prevention) and to mitigate the impact of server failures (fault tolerance). Still, it is important to make sure that no more resources than needed are turned on. Indeed, if too many resources are activated unnecessarily, this will result in a waste of resources. For instance, data centers running VNFs consume a very substantial amount of energy [14] and significantly contribute to the electricity bill of network operation, which is one of the chief concerns of network operators nowadays [15]. In fact, virtualized servers consume even more energy than physical servers [16]. The challenge when scaling resources, thus, is to achieve a good trade-off between performance (activating sufficient resources) and energy consumption (not activating more resources than needed), with the performance being characterized by a reliability guarantee in our case.

In this paper, we address the above by analyzing and optimizing the resource management and activation policies for a farm composed of fallible servers with non-zero start-up times, supporting a communication service with a target reliability level. While our work is motivated by the stringent reliability requirements in 5G/6G, the analysis and proposed scheme could also be applied to other scenarios and other metrics. More specifically, our key contributions are:

- We present an analytical model to characterize the performance of a horizontal auto-scaling server farm dedicated to communication services, where servers are (de)activated following a generic threshold-based policy. Our analysis captures system performance by modeling the reliability and energy consumption as a function of the (de)activation thresholds. In contrast to other most existing models for auto-scaling, we are the first to consider the fallibility of the servers.
- We design an algorithm to optimally configure the thresholds, which guarantees that the reliability is above a given level while minimizing the energy consumption. This contrasts with other proposals to support reliability, which are based on heuristics or machine learning and do not provide any configuration mechanism to meet a target failure probability.
- We perform an extensive performance evaluation that confirms the accuracy of the model and the effectiveness of the configuration algorithm for a wide range of reliability values, in contrast to previous works that only consider coarse metrics. By considering two different deployment strategies for the server farm, we shed some light on whether it is better to provide a service with multiple small machines, or with a few but powerful machines.

The rest of the paper is organized as follows. Section II formally introduces the system model. Section III derives the analytical modeling of the system. Section IV presents the algorithm for its optimal configuration. Section V assesses the accuracy of the analytical model and the performance of the configuration algorithm. Section VI reviews the related work. Finally, we conclude the paper in Section VII.

II. System Model

We consider a cloud system (e.g., MEC, edge cloud or central cloud) supporting a communication service with stringent reliability requirements. The system receives requests to execute one or several tasks with such requirements, e.g., deploy a connected industry service or to instantiate a remote driving slice, involving the execution of their components (i.e., tasks) for a given service duration. For instance, such tasks could be executed as one or more VNFs. The objective is to ensure that these tasks can be immediately started upon arrival, and that continuously run during their entire service lifetime.

To run the VNFs, the system has a number of physical machines (PMs)[2]. To ensure that VNFs are guaranteed the required PM resources (CPU, memory), we limit the number of tasks that can run in a PM. Given the need for a energy-efficient operation, another objective is to keep the number of active PMs to the minimum that guarantees the required reliability. Under these circumstances, service reliability is challenged by two real-life factors: the non-zero boot up times of PMs, and their fallibility, discussed next.

In case a task arrives and there are enough resources available to host it (i.e., at least one PM that is not completely occupied), we assume that it is immediately served, given that deploying VNFs over a running PM is very quick (e.g., in the order of 1 sec [17], [18]). In case there are no sufficient active PMs, the task is put in a queue to ensure that we do not disrupt the other tasks, and a new PM is activated to serve this task. This task will then stay in the queue until either another task finishes (thus freeing some resources) or the new PM becomes active (after the corresponding boot up time). We refer to this as a *failure upon arrival*. As typical boot up times of servers, including the so-called power-on self-test [19] can be in the order of minutes (according to our own measurements detailed in Section V-A, up to 3 min), such failures will create very significant disruption and hence we need to ensure that their probability is very small to provide the desired reliability level.

Since the PMs are fallible, an active PM can crash or shut down at any point in time. In case it was serving one or more VNFs, we assume that, as long as there are enough additional active resources available, these VNFs can be seamlessly migrated to a different PM without disruption. Note that existing technologies already support this operation, e.g., [20], [21] provide the means for restoring the state of a

[1] For instance, for an uptime requirement of 99.9%, we can allow up to 10 min long failure every week.

[2] In this paper, we use interchangeably the terms “server” and “physical machine.”

VNF in a different PM in a very short time; furthermore, for the case of the stateless operation envisioned in 5G [22], [23] this migration would be even simpler. In case there are not enough active PMs, the remaining tasks are placed on hold until sufficient PMs are activated. This is referred to as *failure during service* and again we need to ensure that the probability that this type of failure occurs is very small.

As the metric for the reliability, we will focus on the **failure probability** ($\boldsymbol{P_f}$) of a task, this being defined as the probability that a randomly chosen task is not successfully served. We consider that a task is successfully served only if it never suffers from any failure upon arrival nor any failure during service. To quantify resource consumption, we will compute the average energy consumed by the farm over time, which we refer hereafter as the **power consumption** ($\boldsymbol{\omega}$) of the servers. Note that following our analytical model described in the next section, it would be possible to derive other performance metrics of interest such as, e.g., availability, mean time to failure, or switch on/off rates.

Our analysis relies on the following assumptions:

- Tasks arrive following a Poisson process at a rate λ, and session length follows an exponential random variable with average equal to $1/\mu$.
- As long as the number of tasks running in a server does not exceed a given threshold N, the server can provide all tasks with the required computational resources.
- Boot up times follow an exponentially distributed random variable of average $1/\alpha$, while the time required to shut down a PM is zero.
- PMs are fallible, i.e., they can crash. We model this with a lifetime defined as the time since a PM is active until it crashes. In line with the usual assumptions in the literature [24]–[26], the lifetime follows an exponential random variable of average $1/\nu$.
- Once a server has crashed, it can be immediately activated, i.e., the repair time is zero. This corresponds to software failures that are repaired after a reboot.
- Following usual power consumption models [27], we assume that the power consumed by a server when deactivated is zero, and when active is given by two components: one fixed term that does not depend on the load (i.e., an idle component), and another term that is proportional to the number of served tasks in that server.
- Tasks can be seamlessly migrated across PMs even right before a server crash with no service disruption (thanks to the use of the techniques discussed above) nor extra energy consumption (due to their sporadic occurrence).

In the performance evaluation, we will also consider more general settings to show that our model and algorithm also work when these assumptions do not hold. In what follows, we present the components and operation of our system, and then we detail the threshold-based (de)activation policy.

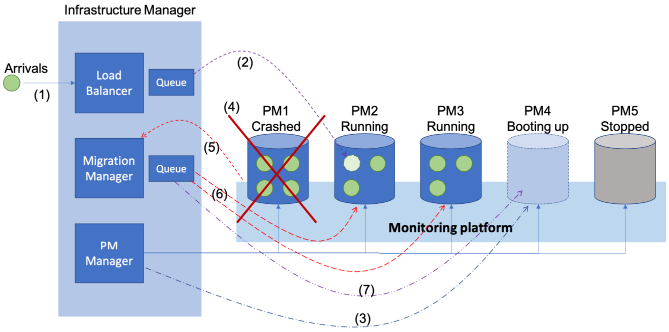


Fig. 1: Server farm with 5 physical machines (PMs), each one with a capacity of up to $N = 4$ tasks

A. System components and operation

The system we analyze in this paper is a server farm composed of an infrastructure manager (IM), and M PMs to handle the requests to deploy new VNFs. Following previous approaches (e.g., [28]–[31]), we assume that the IM is composed of three building blocks: (1) the load balancer, in charge of accommodating new tasks arriving at the system, or holding them until there are enough resources to serve it; (2) the migration manager, responsible for keeping a synchronized backup of the status of the tasks to support their migration across PMs; and (3) the PM manager, responsible for powering on and off the PMs following the resource on demand policy described next.[3] All these blocks are interconnected between them and with a monitoring platform, in charge of detecting the health of the overall infrastructure. The IM is always on, while the PMs can be in one of these states: (1) active, a state in which they can serve requests; (2) booting up, if the PM manager decided that more resources are needed; or (3) stopped (i.e., down), if they crashed or they are not required to handle the current traffic load.

Fig. 1 depicts a simple example composed of $M = 5$ PMs, where each PM can support up to $N = 4$ tasks, three PMs are running, one starting up, and one stopped, and there is a total of 9 tasks at the beginning of the example (represented as green circles). We next describe a sequence of events, numbered from (1) to (7) in Fig. 1, to illustrate the system operation. As we can see, when a new task arrives to the system (1), the load balancer selects the most appropriate PM to process it (PM2 in this case) and places it accordingly (2). Note that our model is independent of the specific algorithm used to assign tasks to PMs and holds both for an algorithm that performs load balancing across PMs as well as an algorithm that does not balance the load. Indeed, given our linear power consumption model (see Sec. III-C) and the assumption that tasks can be seamlessly migrated across PMs, the resulting power and failure performance does not change with the way tasks are assigned to PMs. We acknowledge, however, that performing load balancing might have some advantages in terms of fairness among tasks, but this metric falls outside our scope. Given the change in the load, the algorithm at the PM manager triggers the booting of PM4 (3). In the meantime, the monitoring platform alerts (4) about the incipient crash of

[3] Note that the first two blocks could be integrated into a single one, but we prefer to keep this separation like previous approaches, to properly distinguish between sources of unreliability, as discussed previously.

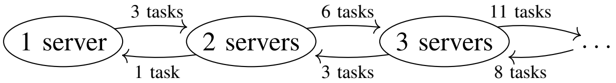


Fig. 2: Active servers and thresholds for the resource on demand policy specified by the equations (1) and (2).

PM1 (e.g., memory issues), and triggers the migration of the four tasks running at this PM (5).

Since PM4 is still booting up, only two out of the four orphan tasks can be migrated to the two available slots at PM2 and PM3, respectively. As discussed above, this migration is immediate, thanks to light-weight migration tools [20], [21] or the use of *stateless* VNFs [22], [23]. Following this, two tasks from PM1 are *seamlessly* migrated to PM2 and PM3 (6), while the third and fourth tasks are migrated from the IM queue to PM4 after it has completed its boot up (7).

B. Resource on demand policy

The policy followed by the PM manager for the management of the resources is as follows. At least one server should be active at all times (even when the system is empty) to avoid any delay for arriving tasks. For the rest of the servers, we employ a threshold-based policy that decides whether to switch on/off an additional server based on the number of tasks being served and the number of active servers.

More specifically, we denote by t_m^{on} the number of tasks in the system that trigger the activation of a m-th PM, and by t_m^{off} the number of tasks that triggers the deactivation of one PM when there are m active PMs. For illustrative purposes, Fig. 2 shows a policy based on the following activation and deactivation thresholds:

$$t_2^{\text{on}} = 3, t_3^{\text{on}} = 6, t_4^{\text{on}} = 11, \tag{1}$$

$$t_2^{\text{off}} = 1, t_3^{\text{off}} = 3, t_4^{\text{off}} = 8. \tag{2}$$

In the above example, with one active server the system activates a second server when there are 3 tasks in the system ($t_2^{\text{on}} = 3$); with 2 servers, a third server is activated when there are 6 tasks ($t_3^{\text{on}} = 6$), but if there is only one task in the system, one of the servers is deactivated ($t_2^{\text{off}} = 1$); and so on. Prior to the actual server deactivation, the PM manager coordinates with the migration manager to seamlessly migrate the tasks from the server to be deactivated (if any) to a different server.

In general, the activation thresholds should satisfy $t_{m+1}^{\text{on}} \geq t_m^{\text{on}}$, and the deactivation threshold for m servers should be smaller than its activation threshold, i.e., $t_m^{\text{off}} < t_m^{\text{on}}$. While meeting these constraints, the use of thresholds enables a large variety of policies, such as, e.g.,

- An *always on* policy, with $t_m^{\text{on}} = 0$ and $t_m^{\text{off}} = -1$.
- A *green* policy, where a new server is activated only when the currently active servers are completely loaded, i.e., $t_m^{\text{on}} = (m-1)N$, and a server is deactivated as soon as possible, i.e., $t_m^{\text{off}} = (m-1)N - 1$. For instance, with $N = 4$ tasks/server, a third server would be activated only when there are 8 tasks in the system, and deactivated when the number of tasks falls to 7.
- A policy that activates and deactivates servers based on relative loads, e.g., an 80-70 policy would activate an additional server when the 80% capacity is reached (i.e., $t_m^{\text{on}} = 0.8\ (m-1)N$) and deactivate a server when the occupation is below 70% (i.e., $t_m^{\text{off}} = 0.7\ mN$).

Variable	Description
M	Number of physical machines (PMs)
N	Capacity of one machine
C	Capacity of the server farm
λ	Task arrival rate
$1/\mu$	Session length
ρ	System load
$1/\alpha$	PM boot up time
$1/\nu$	PM lifetime
t_m^{on}	# tasks to activate an mth server
t_m^{off}	# tasks to deactivate one of the m servers
ω	Power consumed by the server farm
P_{idle}	Power consumption of one PM when idle
P_{load}	Load-proportional power consumption term
P_f	Failure probability
T_f	Target failure probability
(i,j)	State with i tasks and j servers

TABLE I: Main variables used throughout the paper.

In the following, we first characterize the performance of a system operated with a policy based on activation and deactivation thresholds, and then derive the optimal configuration that ensures that the failure probability is below a given threshold, while minimizing the power consumption.

III. System Analysis

The system described above can be modeled with a continuous-time Markov chain with states (i,j), where i represents the total number of tasks in the system, and j the number of active servers available to serve tasks. The total number of rows in the Markov chain is given by $M+1$, where the first row represents the cases with no available servers, while the rest of the rows correspond to the number of active servers. The columns account for the number of tasks, which is unbounded.

Fig. 3 illustrates the case of a system with $M = 3$ servers, where each server can serve up to $N = 2$ tasks, with activation thresholds $t_2^{\text{on}} = 2$ and $t_3^{\text{on}} = 4$, and deactivation thresholds $t_2^{\text{off}} = 0$ and $t_3^{\text{off}} = 2$. One example of a realization of the Markov chain, highlighted in light gray in the figure, would be the following: the initial state with an empty system corresponds to the (0,1) state, and then one arrival (at a rate λ) results in state (1,1), while another arrival leads to (2,1). Since now the number of tasks is equal to t_2^{on}, a novel server is activated (note the α rate), that would lead to state (2,2). From this state, a departure of any of the two tasks (hence the 2μ rate) leads to state (1,2), and the finalization of the last task leads (0,1), since the deactivation time of a server is zero.

In general, in a given state (i,j) there can be up to four possible transitions:

- A task arrives, at a rate λ.
- A task departs, at a rate given by μ times the number of tasks being served. This number is given by the minimum of the current capacity jN and the number of tasks in the system i.

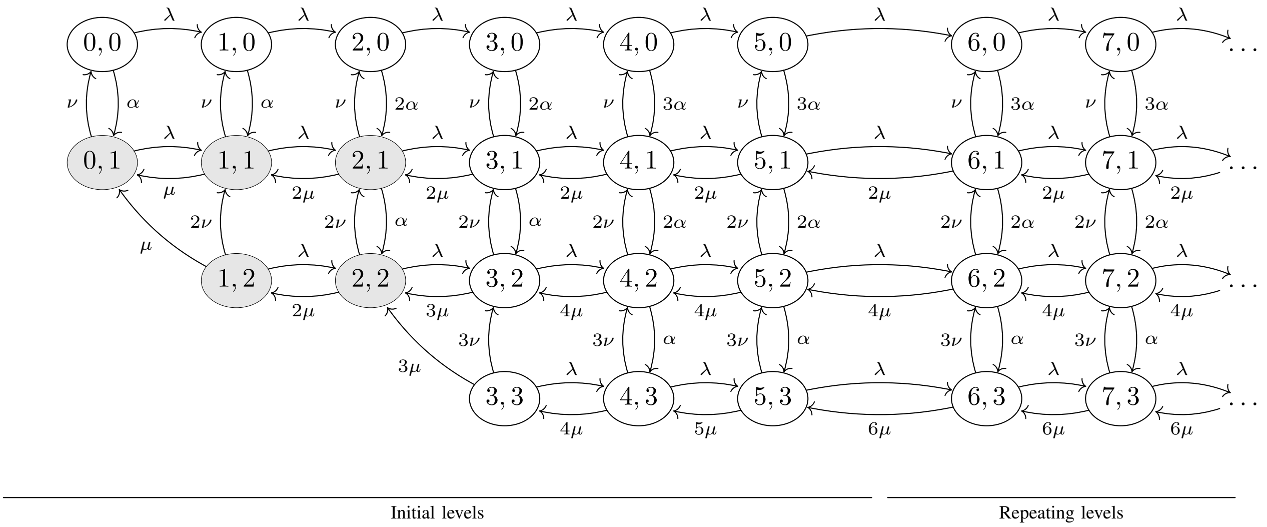


Fig. 3: Markov chain for a system with $M = 3$, $N = 2$, and thresholds $t_2^{\text{on}} = 2$, $t_3^{\text{on}} = 4$, $t_2^{\text{off}} = 0$, and $t_3^{\text{off}} = 2$. States corresponding to the example described in Section III are highlighted in light gray.

- A server crashes, at a rate given by ν times the number of active servers j.
- A server finishes its booting up, at a rate α times the number of servers that are in the setup phase, which depends on the activation thresholds $\{t_m^{\text{on}}\}$.

The above transition rates are defined by the generation (or transition) matrix $\mathbf{Q}$. In the next section, we specify how to compute this matrix.

A. Computation of the generation matrix

The structure of the system corresponds to that of a *quasi-birth-death process* (QBD) [32]. QBDs are a generalization of the birth-death process, where transitions are allowed only between neighboring states, and the transition matrix has a tridiagonal block structure. Given a state (i, j), i is the level of the state, which in our case corresponds to the number of tasks in the system, and j is the inter-level state or phase, which in our case corresponds to the number of active servers.

In general, in a QBD there is a finite number of initial levels I for which the block transition matrices are not necessarily identical, while for the rest of the (possibly infinite) levels, the block matrices are the same (these are called the repeating levels). However, unlike typical QBDs, in our case the number of phases in each initial level is not constant, since the maximum number of servers that could be active, denoted as $A(i)$, depends on the number of tasks i and the deactivation thresholds $\{t_m^{\text{off}}\}$ as follows

$$A(i) = \begin{cases} 1 & \text{if } i \le t_2^{\text{off}}, \\ m & \text{if } t_m^{\text{off}} < i \le t_{m+1}^{\text{off}}, \\ M & \text{if } i > t_M^{\text{off}}. \end{cases} \tag{3}$$

With this, the state space S of our system is given by

$$S = \{(i, j) \mid 0 \le i, 0 \le j \le A(i)\}. \tag{4}$$

Eq. (5) illustrates the transition matrix $\mathbf{Q}$, which contains the matrix $\mathbf{B}$ for the initial levels. The repeating levels correspond to the cases where all servers are active or booting up and the number of tasks is greater than or equal to the capacity of the system (see Fig. 3); while the number of initial levels correspond to the case when some servers are neither active nor booting up or when the number of tasks is lower than the capacity of the system; thus, $I = \max(N \cdot M, t_M^{\text{on}})$.

In what follows, we address the computation of the different blocks of the matrix $\mathbf{Q}$. We first describe how to compute the $B_{i,j}$ blocks of the $\mathbf{B}$ matrix, and then the A_i blocks for the repeating levels.

1) Computation of the $B_{i,i}$ matrices: These square matrices correspond to the transitions within the same level, when there is no change in the number of tasks in the system but only in the number of servers that are available, which depends on the booting up and crashing rates (α and ν, respectively). We have the three following cases.

Cases $i \le t_2^{\mathit{off}}$: These are transitions across levels with up to one active server. The transition matrix is the 2×2 matrix

$$B_{i,i} = \begin{bmatrix} - & \alpha \\ \nu & - \end{bmatrix}. \tag{6}$$

Note that the above matrix does not show the diagonal elements, as they will be computed along with the rest of the diagonal elements of $\mathbf{Q}$ in Section III-A5.

Cases $t_m^{\mathit{off}} < i \le t_{m+1}^{\mathit{off}}$: The transition matrix is a tridiagonal matrix of size $(m+1) \times (m+1)$ given by

$$B_{i,i} = \begin{bmatrix} - & a_{i,0} & & & 0 \\ \nu & - & a_{i,1} & & \\ & 2\nu & \ddots & \ddots & \\ & & \ddots & \ddots & a_{i,m-1} \\ 0 & & & m\nu & - \end{bmatrix}. \tag{7}$$

$$\mathbf{Q} = \left[\begin{array}{cccccc|cccc} B_{00} & B_{01} & 0 & 0 & \dots & 0 & & & & \\ B_{10} & B_{11} & B_{12} & 0 & \dots & 0 & & & & \\ 0 & B_{21} & B_{22} & B_{23} & \dots & 0 & & & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \dots & \big\}\,\mathbf{B}\text{: matrix for the initial levels} & & & \\ 0 & \dots & \dots & B_{I-2,I-3} & B_{I-2,I-2} & B_{I-2,I-1} & 0 & & & \\ 0 & \dots & \dots & 0 & B_{I-1,I-2} & B_{I-1,I-1} & A_0 & 0 & & \\ \hline 0 & \dots & \dots & 0 & 0 & A_2 & A_1 & A_0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \dots & & & & \end{array}\right]. \tag{5}$$

where $a_{i,j}$ represents the activation rate when there are i tasks and j active servers. To compute $a_{i,j}$, we proceed as follows. Let $S(i)$ represent the minimum number of servers that should be active when there are i tasks in the system. This is given by

$$S(i) = \begin{cases} 1 & \text{if } i < t_2^{\text{on}}, \\ m & \text{if } t_m^{\text{on}} \leq i < t_{m+1}^{\text{on}}, \\ M & \text{if } i \geq t_M^{\text{on}}. \end{cases} \tag{8}$$

Then, $a_{i,j}$ can be computed by subtracting j from $S(i)$, i.e. the number of servers being boot up. Thus,

$$a_{i,j} = \alpha \cdot \max\{(S(i) - j), 0\}. \tag{9}$$

Cases $i > t_M^{off}$: This is special case of the previous one with $m = M$, so we can use the same expression used there to compute $B_{i,i}$.

2) Computation of the $B_{i,i+1}$ matrices: In these transitions the number of tasks in the system increases. The matrices can be either square or rectangular, depending on the level: in some cases the number of phases does not vary (e.g., 2nd and 3rd level in Fig. 3) while in others it increases by one (first level in the figure).

Cases $i < t_2^{off}$: The transition matrix $B_{i,i+1}$ is a diagonal matrix with all diagonal elements equal to λ, i.e. $B_{i,i+1} = \lambda \times \mathbf{I}_2$, where $\mathbf{I}_n$ is the identity matrix of size n.

Case $i = t_2^{off}$: This case corresponds to the transitions from a level with 2 phases to a level with 3 phases. The matrix $B_{i,i+1}$ is a 2×3 matrix formed by the column concatenation of the matrix $\lambda \times \mathbf{I}_2$ and a vector of size 2 with all zeros.

Cases $t_m^{off} < i < t_{m+1}^{off}$: These cases correspond to transitions between levels with the same number of phases $m+1$, leading to the the transition matrix $B_{i,i+1} = \lambda \times \mathbf{I}_{m+1}$.

Cases $i = t_m^{off}$: Here the transitions are from a level with m states to a level with $m+1$ states. This leads to a rectangular matrix $B_{i,i+1}$ of size $m \times (m+1)$ formed by the column concatenation of the matrix $\lambda \times \mathbf{I}_m$ and a vector of size m with all zeros.

Cases $i > t_M^{off}$: When the number of servers that can be active is M, the number of phases is $M+1$ and the transition matrix is $B_{i,i+1} = \lambda \times \mathbf{I}_{M+1}$.

3) Computation of the $B_{i,i-1}$ matrices: In these cases the number of tasks in the system decreases, i.e., a task was served. The transition rates depend on the number of active servers and the number of tasks that are being served. Let $d_{i,j}$ denote the departure rate with i tasks and j active servers, which can be expressed as

$$d_{i,j} = \mu \cdot \min\{i, jN\} \tag{10}$$

where the $\min$ takes into account the cases when the number of tasks in the system i exceeds the capacity with the current numbers of active servers j, which is jN. Like before, we divide the computation of $B_{i+1,i}$ in a number of cases.

Cases $i < t_2^{off}+1$: These are transitions across levels with one active server. The transition matrix is a 2×2 diagonal matrix with diagonal elements given by the vector $[d_{i,0}, d_{i,1}]$.

Case $i = t_2^{off}+1$: These transitions correspond to the case when the second server is deactivated. The transition matrix, of size 3×2, is formed by the row concatenation of the transition matrix given in the previous case and a row vector of size 2 with all its elements zero except the last one, which is $d_{i,2}$.

Cases $t_m^{off}+1 < i < t_{m+1}^{off}+1$: In this case, $B_{i+1,i}$ is a diagonal matrix of size $(m+1) \times (m+1)$ with diagonal elements given by the vector $[d_{i,0}, d_{i,1}, \dots d_{i,m}]$.

Cases $i = t_{m+1}^{off}+1$: Similarly to the case $i = t_2^{\text{off}}+1$ discussed above, here the matrix is a rectangular diagonal matrix of $(m+2) \times (m+1)$ elements, formed by the row concatenation of the transition matrix of the previous case and a row vector of size $m+2$ with all its elements zero except the last one, which is $d_{i,m+1}$.

Cases $i > t_M^{off}+1$: In this case the transition matrix is a diagonal matrix of size $(M+1) \times (M+1)$ with diagonal elements given by the vector $[d_{i,0}, d_{i,1}, \dots d_{i,M}]$.

4) Computation of the A matrices:

Matrix A_1: This corresponds to the transition rates across phases of the same repeating level. The transition matrix is a tridiagonal matrix of size $(M+1) \times (M+1)$ with the upper diagonal being $[M\alpha, (M-1)\alpha, \dots, \alpha]$, the lower diagonal $[\nu, 2\nu, \dots, M\nu]$, and the main diagonal computed in Section III-A5.

Matrix A_0: This corresponds to the transitions from a given repeating level to the next one. It is a square matrix of size $(M+1) \times (M+1)$ given by $\lambda \times I_{M+1}$.

Matrix A_2: This matrix of size $(M+1) \times (M+1)$ captures the transitions from a repeating level to the previous one. Given that all servers are completely occupied (each one serving N tasks), A_2 is a diagonal matrix with diagonal elements given by the vector $[0, N\mu, 2N\mu, \ldots, MN\mu]$.

5) Computation of the diagonal elements: Once we have computed all the off-diagonal elements of $\mathbf{Q}$, we can compute the diagonal elements by applying the property that all the elements of each row of $\mathbf{Q}$ have to sum zero.

Cases $i \leq t_2^{\mathit{off}}$:

$$b_{i,j} = \begin{cases} -\alpha - \lambda & \text{if } j = 0, \\ -\nu - \lambda - d_{i,1} & \text{if } j = 1. \end{cases} \tag{11}$$

Cases $t_m^{\mathit{off}} < i \leq t_{m+1}^{\mathit{off}}$:

$$b_{i,j} = \begin{cases} -a_{i,j} - j\nu - \lambda - d_{i,j} & \text{if } 0 \leq j < m, \\ -m\nu - \lambda - d_{i,m}, & \text{if } j = m. \end{cases} \tag{12}$$

Cases $i > t_M^{\mathit{off}}$:

$$b_{i,j} = \begin{cases} -a_{i,j} - j\nu - \lambda - d_{i,j} & \text{if } 0 \leq j < M, \\ -M\nu - \lambda - d_{i,M} & \text{if } j = M. \end{cases} \tag{13}$$

As for the diagonal elements b_j of the A_1 matrix, we have

$$b_j = -(M-j)\alpha - j\nu - \lambda - j\mu N. \tag{14}$$

B. Computation of the steady-state distribution

Given the transition matrix $\mathbf{B}$ for the initial levels and the transition matrices A_0, A_1 and A_2 for the repeating levels, the computation of the steady-state probabilities $\pi_{i,j}$ for each state (i,j) is a well-known procedure (see e.g. [32]) that can be applied as long as the system is stable.

As demonstrated in the Appendix, the stability condition depends on the system load ρ, defined as the ratio between the input rate and the maximum service rate, i.e.,

$$\rho \equiv \frac{\lambda}{NM\mu} < \frac{\alpha}{\nu + \alpha}. \tag{15}$$

C. Computation of the performance metrics

We next address how to compute the performance metrics we consider in this paper, namely, the power consumption and the failure probability.

1) Power consumption: In line with traditional power consumption models in the literature [14], [33], we characterize the power consumption behavior of a single PM with the following two terms: the idle power consumption P_{idle} and a term that is proportional to the utilization P_{load} (we do not consider the power consumption of the IM as this is the same throughout all comparisons). Based on these, the average energy consumed by the PMs can be computed by weighting the stationary probability of each state by its associated power consumed, i.e.,

$$\omega = \sum_{(i,j)\in S} \pi_{i,j} j \left(P_{idle} + P_{load} \frac{\min(i, jN)}{jN} \right), \tag{16}$$

which can be rewritten as

$$\omega = \sum_{(i,j)\in S} \pi_{i,j} \left(j P_{idle} + \min(i, jN) \frac{P_{load}}{N} \right). \tag{17}$$

where the term P_{load}/N can be understood as the energy cost associated with a single request.

2) Failure probability: To obtain the failure probability P_f, we first compute the following two probabilities:

- P_{wait}, defined as the probability that a task has to wait upon arrival.
- P_{int}, defined as the probability that a task is interrupted during service because of a server crash.

Given that we target small values of P_f, we assume that the above events are mutually exclusive (i.e., we neglect the probability that a task has to wait and is interrupted). With this approximation, P_f can be computed as:

$$P_f = P_{\text{wait}} + P_{\text{int}}. \tag{18}$$

Furthermore, note that we also make the worst-case approximation that any task that goes into the queue always yields a failure, even if there is a small chance that this task remains in the queue for a short time and hence does not cause a significant disruption. Our performance evaluation results confirm the accuracy of this approximation.

Computation of P_{wait}: A task has to wait when it arrives to the system when there are no free resources. Because of the PASTA (Poisson Arrivals See Time Averages) property, this is given by

$$P_{\text{wait}} = \sum_{(i,j)\in S} \mathbf{1}(i \geq jN) \pi_{i,j}, \tag{19}$$

where $\mathbf{1}(i \geq jN)$ is an indicator function which is equal to 1 when $i \geq jN$, i.e., when the number of tasks i is equal or higher than the available capacity, and zero otherwise.

Computation of P_{int}: The probability that a task is interrupted during service can be computed as the ratio between the average number of tasks that are disrupted per time unit, which we denote as γ, over the average arrival rate λ:

$$P_{\text{int}} = \frac{\gamma}{\lambda}. \tag{20}$$

We can compute γ as

$$\gamma = \sum_{(i,j)\in S} \pi_{i,j} \; j\nu \; F_{i,j}^r, \tag{21}$$

where $j\nu$ is the server crash rate in the (i,j) state, and $F_{i,j}^r$ denotes the expected number of running tasks that are disrupted upon a server crash. $F_{i,j}^r$ can be computed as the difference between the number of running tasks and the capacity of the system right after the failure, i.e., with $j-1$ active servers,

$$F_{i,j}^r = \min(\max(i - (j-1)N, 0), N), \tag{22}$$

where the $\max$ operator prevents negative numbers in case no task is affected (because there is enough available capacity with $j-1$ servers), and the $\min$ operator limits the effect of a failure to the maximum capacity of a single server.

IV. Optimal configuration

In what follows, we address the optimal configuration of the activation and deactivation vectors $\{t_m^{\text{on}}\}$ and $\{t_m^{\text{off}}\}$, respectively. The objective is to obtain the configuration that minimizes the power consumed ω while guaranteeing that the failure probability P_f is smaller than a target failure probability T_f.

A. Problem formulation

The optimization problem can be formulated as

$$\min_{\{t_m^{\text{on}}\},\{t_m^{\text{off}}\}} \quad \omega, \tag{23a}$$

$$\text{subject to} \quad P_f \leq T_f. \tag{23b}$$

Finding the exact solution to the above optimization problem in reasonable time would be infeasible due to the large size of the configuration space and the combinatorial nature of the problem. Hence, in what follows we rely on a approximations to efficiently compute the threshold configuration.

Following the power consumption model, it is reasonable to assume that the most efficient configuration has the activation threshold for the m-th server right after the deactivation threshold for this additional server, i.e., $t_m^{\text{off}} = t_m^{\text{on}} - 1$. The rationale for this choice is that, if having m servers active is sufficient to keep the desired failure probability, we would not want to keep more servers active when the number of running tasks goes below i. For the sake of readability, in what follows we will simplify notation and refer to t_m^{on} as t_m, i.e., $t_m \equiv t_m^{\text{on}}$. Next, we first present two approximate analysis for the computation of ω and P_f, respectively, and then propose an algorithm to solve the optimization problem.

B. Approximation to compute the power consumption

We make the approximation that, as far as power consumption is concerned, the server crash probability can be neglected and the activation time is very small. With this approximation, the system behaves as an M/M/C queueing model with $C = MN$, and therefore it is straightforward to compute the steady-state distribution vector $\{p_i\}$, where p_i denotes the probability of having i tasks.

To compute the power consumption, we consider the following two terms, one associated with the idle power consumption P_{idle} and one corresponding to the load term P_{load}:

$$\omega = \omega^{idle} + \omega^{load}. \tag{24}$$

The m-th server will be on whenever the number of running tasks is equal to or above t_m, so its contribution to the idle power consumption can be expressed as

$$\omega_{m,t_m}^{idle} = \sum_{i=t_m}^{\infty} p_i P_{idle}, \tag{25}$$

and therefore

$$\omega^{idle} = \sum_{m=1}^{M} \omega_{m,t_m}^{idle}. \tag{26}$$

To compute ω^{load}, we denote with $P_{req} = P_{load}/N$ the power consumed by single task (see (17)), and consider the aggregate consumption resulting from all running tasks up to the total system capacity C, which leads to

$$\omega^{load} = \sum_{i=1}^{\infty} p_i \min(i, C) P_{req}. \tag{27}$$

With the above, the overall power consumed is given by

$$\omega = \sum_{m=1}^{M} \omega_{m,t_m}^{idle} + \sum_{i=1}^{\infty} p_i \min(i, C) P_{req}. \tag{28}$$

Note that the second summation in the above expression is independent of the setting of $\{t_m\}$ and plays the role of a constant; thus, when computing the optimal configuration we only need to take into account the first summation.

C. Approximation to compute the failure probability

The failure probability can be computed as follows

$$P_f = P_\alpha + P_\nu, \tag{29}$$

where P_α and P_ν are the probabilities of a task being affected by the boot up delay triggered by exceeding an activation threshold and a server crash, respectively.

1) Computation of P_α: The probability that a task is affected by a boot up delay can be expressed as the ratio between the rate of tasks affected (i.e., the average number of affected tasks per unit time), denoted as δ, over the arrival rate λ:

$$P_\alpha = \frac{\delta}{\lambda}. \tag{30}$$

To compute δ, we leverage the steady-state probability vector $\{p_i\}$ assuming an M/M/C system, and focus on the states right before the activation thresholds $\{t_m\}$, each one with probability p_{t_m-1}. Under such states, if there is a new arrival, the activation of a new server is triggered, leading to a rate for the boot up of the m-th server given by $p_{t_m-1}\lambda$. If we denote by F_m^δ the average number of tasks affected during the boot up of the m-th server, this leads to

$$\delta = \sum_{j=1}^{M} p_{t_m-1} \lambda F_m^\delta. \tag{31}$$

To compute F_m^δ, we focus on the behavior of the system during the boot up of the m-th server as a function of the threshold t_m, assuming that the rest of thresholds are separated by N. Note that this assumption is needed to decouple the effect of the thresholds other that t_m, since we assume that the impact of each threshold in the power consumption and the failure probability is independent of the rest of thresholds. With this, the system can be modeled with a continuous-time Markov chain similar to the one depicted in Fig. 3, but with several absorbing states corresponding to (1) the case when either the activation of the m-th server has finished (a downward transition at a rate α happened), and (2) the activation has been canceled because enough tasks have left the system (no downward transition is possible anymore). We illustrate this chain in Fig. 4 for the same system as in Fig. 3, assuming as initial state (2,1), i.e., for $t_2 = 2$. The absorbing states are those corresponding to either the activation of a

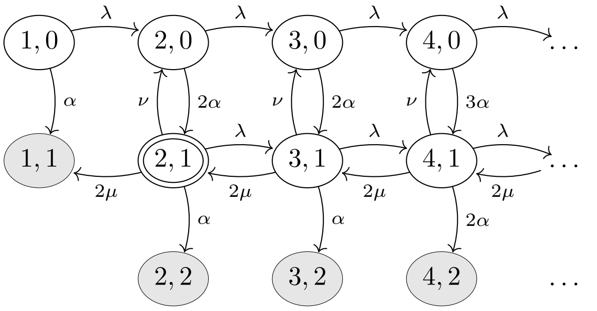


Fig. 4: Markov chain for the computation of F_m^δ for the same system as in Fig. 3. The initial state is (2,1) and the absorbing states are highlighted in light gray.

server, i.e., (2,2), (3,2), etc., or the cancel of the activation after a task departure, i.e., (1,1).

We denote the matrix corresponding to this chain as $\mathbf{Q}_m$, which is obtained from $\mathbf{Q}$ by removing all the rows/columns corresponding to transitions from/to states where

- the number of servers is higher or equal to m (i. e., the m-th server has booted up), or
- there are not enough tasks to activate the m-th server anymore

With the above, we can compute the expected time that the Markov chain spends in the non-absorbing states as

$$\boldsymbol{L}_m \boldsymbol{Q}_m = -\boldsymbol{\pi}_m(0), \tag{32}$$

where $\boldsymbol{L}_m$ is an infinite-length vector restricted to the non-absorbing states considered in the evaluation, where each position corresponds to the expected time the Markov chain spends in a non-absorbing state prior to absorption, and the vector $\boldsymbol{\pi}_m(0)$ corresponds to the initial state probabilities of the Markov chain (in our case, all the elements are zero except the one corresponding to the state $(t_m, m-1)$, which is 1). Equation (32) is a system of equations that can be solved numerically.

During the boot up of the m-th server, there are two sources of failure: newly arriving tasks that have to wait, and running tasks that must be moved to the queue because a server crash happened.

Thus, we can express F_m^δ as

$$F_m^\delta = F_m^{\delta,n} + F_m^{\delta,r}, \tag{33}$$

where $F_m^{\delta,n}$ is the average number of tasks that cannot be served immediately upon arrival, and $F_m^{\delta,r}$ is the average number of tasks affected by a server crash. $F_m^{\delta,n}$ can be computed as the arrival rate times the average time that the chain spends on states with all active servers full in the absorbing Markov chain, i.e.,

$$F_m^{\delta,n} = \lambda \sum_{(i,j)\in S} \mathbf{1}(i \geq jN) L_{i,j}, \tag{34}$$

where $L_{i,j}$ are the different elements of $\boldsymbol{L}_m$. $F_m^{\delta,r}$ can be computed as the average time spent in each state in the absorbing Markov chain, times the server crash rate in that state, times the average number of tasks affected by a server crash in that state, i.e.,

$$F_m^{\delta,r} = \sum_{(i,j)\in S} L_{i,j}\ j\nu\ F_{i,j}^r, \tag{35}$$

where the term $F_{i,j}^r$ is given by (22).

2) Computation of P_ν: The computation of the failures triggered by a server crash can be computed by

$$P_\nu = \sum_{i=0}^{t_2-1} p_i P_{i,1}^\nu + \sum_{i=t_2}^{t_3-1} p_i P_{i,2}^\nu + \ldots + \sum_{i=t_M}^{\infty} p_i P_{i,M}^\nu, \tag{36}$$

where $P_{i,m}^\nu$ is the probability that a task is affected by a server crash when there are i tasks in the system and m active servers.

Since our optimization algorithm needs to decouple the effect resulting from each threshold, we want that each of the terms in the above sum depends on one threshold only. To this end, we make the following approximation:

$$\begin{aligned} P_\nu \approx & \sum_{i=0}^{t_2-1} p_i P_{i,1}^\nu + \sum_{i=t_3-N}^{t_3-1} p_i P_{i,2}^\nu + \ldots \\ & + \sum_{t_M-N}^{t_M-1} p_i P_{i,M-1}^\nu + \sum_{i=t_M}^{\infty} p_i P_{i,M}^\nu. \end{aligned} \tag{37}$$

The probability $P_{i,m}^\nu$ can be computed as the ratio between the rate at which tasks are affected by a server crash over the arrival rate λ:

$$P_{i,m}^\nu = \frac{m\nu F_{i,m}^\nu}{\lambda}, \tag{38}$$

where $m\nu$ represents the rate at which a server crashes when there are m active servers, and $F_{i,m}^\nu$ represents the average number of tasks affected by a server crash when there were i tasks in the system and m active servers.

A server crash might trigger three types of failures: (1) running tasks that are moved to the queue; (2) newly arriving tasks that cannot be immediately served because no resources are yet available; and (3) running tasks that are moved to the queue because of additional server crashes during the booting up of the first server crash. By denoting the average number of these tasks as $F_{i,m}^{\nu,q}$, $F_{i,m}^{\nu,n}$, and $F_{i,m}^{\nu,r}$, respectively, we have

$$F_{i,m}^\nu = F_{i,m}^{\nu,q} + F_{i,m}^{\nu,n} + F_{i,m}^{\nu,r}. \tag{39}$$

$F_{i,m}^{\nu,q}$ can be computed as the difference between the number of running tasks and the capacity of the system right after the failure (i.e., with $m-1$ active servers), which is given by (22). To compute $F_{i,m}^{\nu,n}$ and $F_{i,m}^{\nu,r}$, we proceed as in the boot up delay case (Section IV-C1) but with all the elements of the initial probability vector equal to 0, except for the one corresponding to the $(i, m-1)$ state which is 1.

3) Failure probability in terms of $\{t_m\}$: By substituting the expressions for P_α and P_ν and into (29), we obtain

$$\begin{aligned} P_f = & \sum_{m=1}^{M} p_{t_m-1} F_m^\delta + \sum_{i=0}^{t_2-1} p_i P_{i,1}^\nu + \sum_{i=t_3-N}^{t_3-1} p_i P_{i,2}^\nu + \ldots \\ & + \sum_{i=t_M-N}^{t_M-1} p_i P_{i,M-1}^\nu + \sum_{i=t_M}^{\infty} p_i P_{i,M}^\nu, \end{aligned} \tag{40}$$

which can be re-arranged as follows

$$P_f = \sum_{m=2}^{M} P^f_{m,t_m}, \tag{41}$$

where the term P^f_{m,t_m} is given by

$$P^f_{m,t_m} = p_{t_m-1} F^\delta_m + P^{f,\nu}_{m,t_m}, \tag{42}$$

with

$$P^{f,\nu}_{m,t_m} = \begin{cases} \sum\limits_{i=0}^{t_m-1} p_i P^\nu_{i,1} \quad \text{for } m = 2, \\ \sum\limits_{i=t_m-N}^{t_m-1} p_i P^\nu_{i,m-1} \quad \text{for } m = 3, \ldots, M-1, \\ \sum\limits_{t_M-N}^{t_M-1} p_i P^\nu_{i,M-1} + \sum\limits_{i=t_M}^{\infty} p_i P^\nu_{i,M} \quad \text{for } m = M. \end{cases} \tag{43}$$

With the above, we have a number of components such that each one is associated with one activation threshold, and independent from the other thresholds. This is the basis of the algorithm below to select the optimal configuration of each threshold.

D. Algorithm to compute the optimal thresholds

With the above approximations, we can re-formulate our optimization problem as follows. Let $x_{m,k} = 1$ be a binary variable indicating if the threshold on the m-th server is k (for instance, $x_{2,5} = 1$ indicates that $t_2 = 5$). Let $\mathcal{M}_m$ be the set of feasible values of t_m, which goes from 0 to $N(m-1)$. Then, the optimization problem can be formulated as the following multiple-choice knapsack problem (MCKP):

$$\underset{\{x_{m,k}\}}{\text{minimize}} \quad \sum_{m=2}^{M} \sum_{k \in \mathcal{M}_m} x_{m,k} \omega^{idle}_{m,k}, \tag{44a}$$

$$\text{subject to} \quad \sum_{m=2}^{M} \sum_{k \in \mathcal{M}_m} x_{m,k} P^f_{m,k} \leq T_f, \tag{44b}$$

$$\sum_{k \in \mathcal{M}_m} x_{m,k} = 1, \quad \forall m = 2, \ldots, M, \tag{44c}$$

$$x_{m,k} \in \{0,1\} \quad \forall m = 2, \ldots, M, k \in \mathcal{M}_m. \tag{44d}$$

In the above problem, the objective is to minimize the power consumption (44a) (more specifically, the first term of the overall power consumption given by (28)), while guaranteeing that the failure probability is below a given threshold (44b). The restriction (44c) ensures that only one value is chosen for each threshold t_j. The MCKP is a well-known problem for which exists fully polynomial-time approximation scheme (FPTAS) algorithms. For instance, the algorithm in [34] obtains a 4/5-bounded solution in $O(n \log m)$ time, with n being the total number of items (in our case, $n < N(M-1)$) and m being the number of multiple-choice classes (in our case, $m = M - 1$).

As the optimization is based on a number of simplifying assumptions, it could possibly happen that the solution results in a failure probability P_f slightly different to the target T_f: if it is higher than the target, we should reduce it by decreasing some thresholds; if it is lower than the target, there might be room for energy savings by increasing some thresholds. Additionally, it could also happen that some thresholds do not satisfy the requirement that $t_m \geq t_{m'}$ when $m > m'$. To refine the solution of the optimization problem, we follow the post-optimization procedure described in Algorithm 1. This procedure works as follows. First, if for any m, m' such that $m' > m$ we have that $t'_m < t_m$, then we swap the values of these thresholds. Next, we sort the thresholds t_m in decreasing order of their corresponding values of P^f_{m,t_m}. Then, we loop through them (lines 3–19): if the failure probability P_f obtained with the model is higher than its target value T_f (line 4), we decrease the corresponding threshold t_m as long as $P_f > T_f$ and $t_m > 0$; otherwise, if the P_f is smaller than T_f (the `else` in line 9), there is room for energy saving by increasing the threshold: this is done until the maximum value is reached ($t_m < mN$, line 10) as long as the condition $P_f > T_f$ is not met (line 13). In this case, we decrease the threshold to fix the last increment and move to the next threshold.

Algorithm 1: Post-optimization tuning

```
input : Initial solution $\{t_m\}$ to the optimization (44)
output: Fine-tuned solution $\{t_m\}^*$
1  Swap($t_m, t'_m$) for any $m, m'$ such that $m > m'$ and
     $t_m < t_{m'}$;
2  Compute $P_f$ with $\{t_m\}$ using the model (18);
3  Sort $\{t_m\}$ in decreasing order of $P^f_{m,t_m}$;
4  for $t_m$ in sorted $\{t_m\}$ do
5      if $P_f > T_f$ then
6          while $t_m > 0$ and $P_f > T_f$ do
7              $t_m \leftarrow t_m - 1$;
8              Re-compute $P_f$;
9          end
10     else
11         while $t_m < mN$ do
12             $t_m \leftarrow t_m + 1$;
13             Re-compute $P_f$;
14             if $P_f > T_f$ then
15                 $t_m \leftarrow t_m - 1$;
16                 Re-compute $P_f$;
17             end
18         end
19     end
20 end
```

V. Performance evaluation

Throughout our performance evaluation we will compare two different deployments, namely, one consisting of many affordable nano computers, and one consisting of a few more powerful servers. This will serve to validate our performance evaluation and optimal configuration algorithm in a variety of scenarios, and to shed some light on whether deployments consisting of general-purpose hardware can provide similar performance to those consisting of high-performing hardware.

Throughout most of this section, the performance evaluation will consist of a comparison between results obtained from the analytical model and those extracted from simulations. Results from the analytical model are obtained using `Matlab` Release 2020b, while simulation results are obtained from a discrete event simulator written in C++. To assess the accuracy of the model under realistic traffic conditions, we also evaluate its performance using real traces in Sec. V-G. All these computations were executed on a server with an Intel Core i7 980x Extreme edition CPU with 6 cores, 12 threads, using its base frequency of 3.33GHz and 24 GB of RAM.

A. *System parameters*

In the following, we discuss the quantitative figures considered during the performance evaluation, but we also have confirmed the accuracy of the analytical model and optimization algorithm for other numerical figures. Following the usual approaches in the literature (e.g., [35], [36]), we collect several parameters from previous studies, and make reasonable assumptions for those that are not available (as reported in [25], data of this type is typically vendor-confidential and not available for publication).

We assume that tasks arrive following a Poisson process at a rate λ, which we will vary in our experiments, and that service times follow an exponential random variable with an average equal to $1/\mu = 1$ hour. The two deployments we consider are described in Table II, each one supporting up to $M \times N = 256$ simultaneous tasks (during our performance evaluation we also analyze smaller and larger deployments).

Parameter	Rack servers	Nano servers
M	8 servers	64 servers
N	32 tasks/server	4 tasks/server
$1/\mu$	1 hour	1 hour
$1/\alpha$	3 min	20 sec
$1/\nu$	32 days	8 days
P_{idle}	150 W	4.6 W
P_{load}	120 W	3.0 W

TABLE II: Deployments considered throughout our performance evaluation

Deployment 1: Rack servers. This deployment is composed of $N = 8$ servers, each one modeled from a Dell Power Edge server equipped with 32 GB of memory and supporting $M = 32$ tasks per server. Following *the Power Consumption Database* (TPCDB),[4] a server consumes $P_{max} = 270$ W at its peak load and P_{idle}= 150 W while in idle (note that these numbers are in line with those of the least consuming servers analyzed in [14]). Assuming that a server reaches its peak power consumption when fully loaded, we can compute the proportional term as $P_{load} = P_{max} - P_{idle} = 120$ W. According to our own measurements, a server boots in 3 min, while following [37], we assume that the mean time to failure (MTTF) of a server is 768 h (i.e., 32 days).

Deployment 2: Nano servers. This deployment is composed of $N = 64$ nano servers, which we modeled from a Raspberry Pi (RPi) 4b with 4 GB of memory, each one supporting up to $M = 4$ simultaneous tasks. Following TPCDB,[5] each nano server consumes 4.6 W while in idle and 7.6 W at the highest load level. According to our measurements, an RPi boots in approximately 20 sec. Nano servers are typically much cheaper and less reliable than carrier-grade servers, which are typically designed with redundant components. Based on this, we assume that their reliability is one-fourth of that of a rack server, i.e., 8 days. While this assumption is somehow arbitrarily as there is no data available on the reliability of nano servers, we believe that it serves to capture the trend of using this type of server. Note also that our contribution is not affected by the specific parametrization that we use and one could use our model to evaluate the performance of any other parametrization.

B. *Model validation*

We start our performance evaluation by comparing the results from our analytical model to those obtained via simulations. To this aim, we analyze the performance of the two deployments for three different configurations of the thresholds:

- Green configuration: a novel server is powered on only if all the active servers are completely occupied, and deactivated the moment the capacity for $N+1$ additional tasks is available (i.e., the capacity of one server plus one task). This would correspond to the least consuming scaling policy, and the one that should provide the highest failure probability.
- Red configuration: a novel server is powered on when 95% of the current capacity is reached, and deactivated when, without one server, the occupation would be 85% or less. This configuration should provide a smaller failure probability than the green one, but a higher power consumption.
- Yellow configuration: a novel server is powered on when the occupation is equal to or above 95%, and powered off when it would fall below 95% without a server. The performance of this configuration should fall between the two cases considered above.

We plot the failure probability versus the arrival rate for these three configurations and the two considered deployments in Fig. 5, where we use points for the simulation values and lines for the analysis, and the color of each line and each point corresponds to the name of the configuration. Each point represents the average of 20 simulations, each simulation consisting of 100 million tasks. We also show in the figure the 95% confidence intervals (they are so small that they can only be seen for very low probability values). The results corresponding to the rack servers are represented with solid lines, while the ones corresponding to the nano servers are represented with dashed lines. The figure confirms the accuracy of the model, as the results

[4] http://www.tpcdb.com/product.php?id=2325

[5] http://www.tpcdb.com/product.php?id=4417

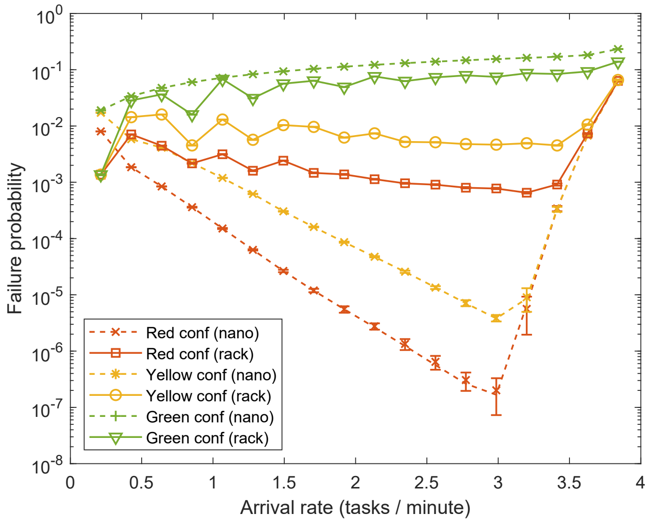


Fig. 5: Failure probability versus arrival rate: rack servers (solid lines) and nano servers (dashed lines).

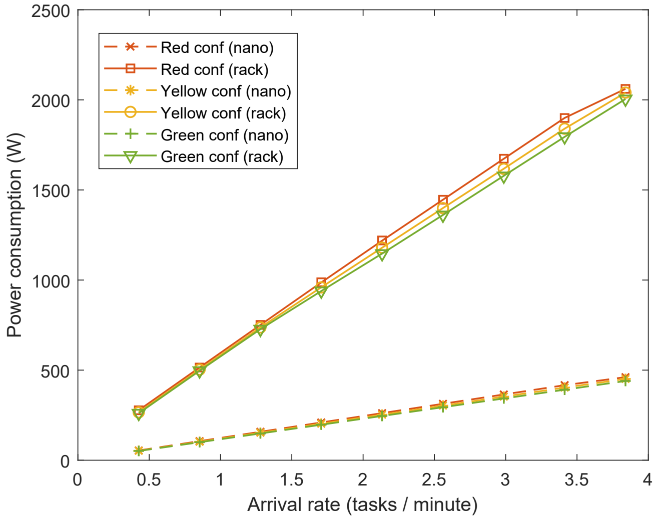


Fig. 6: Power consumption versus arrival rate: rack servers (solid) and nano servers (dashed).

from the simulation coincide with those from the analysis in practically all the cases. Furthermore, we note that this accuracy is achieved despite the various orders of magnitude of the failure probability, i.e., between approximately 10^{-3} and 10^{-1} for the rack servers, and between 10^{-7} and 10^{-1} for the nano servers. Secondly, we observe that the behavior of the failure probability is not monotonous with the arrival rate for the yellow and red configurations: it decreases with λ up to a point (approximately 3 tasks/min), and then it increases again. The decrease is caused by the anticipated activation of additional resources, while the increase starts when there are no more resources to be activated despite the load variations. By comparing the performance of the two deployments, the figure shows that, depending on the configuration considered, it is possible to provide smaller failure probabilities with enough nano servers despite their shorter lifetime. Finally, we validate our assumption that a task going to the queue always yields a failure. An analysis of the waiting time shows that 95% of the tasks going into the queue have to wait for more than 1 sec, hence suffering substantial disruption. This confirms that when a task goes into the queue, this results in a failure with a very high probability.

We next validate the power consumption model following a similar approach, with the results being depicted in Fig. 6. , where the power consumption is computed as the total energy consumed during the simulation divided by the duration of the simulation. Again, lines and points practically coincide, which confirms the accuracy of the model. In the remaining figures of the paper we do not depict confidence intervals to avoid harming their readability; in any case, confidence intervals always fall below 1% of the average. In contrast to the previous case, we have a monotonous behavior of the power consumption (ω) the arrival rate. The results also show the usual trade-off between performance and resource consumption, as the configuration policies that lead to smaller failure probabilities correspond to higher ω values. Finally, the figure also suggests that the nano server deployment could guarantee smaller failure probabilities at a smaller power consumption, although for a fair comparison, we should optimize the configuration of both deployments, which is addressed next.

We would like to note that the above results are obtained with the specific numerical values provided in Table II and therefore cannot be generalized. However, we do believe that they provide some insights on the performance that one may achieve when using smaller and less reliable servers. In any case, our model allows the evaluation of any parametrization and hence can be used to assess the performance of other types of servers.

C. *Optimization algorithm*

We next assess the performance of the optimization algorithm presented in Section IV. To this aim, we consider the same two deployments as before, three values of the system load $\rho = \lambda/MN\mu$, namely, 10%, 30%, and 50%, and a number of target failure probability levels (T_f) between 10^{-3} and 10^{-6}. For each case considered, we compare the power consumption with the (de)activation thresholds computed by our algorithm against the power consumption obtained from an exhaustive search in the whole space of thresholds. To implement this search, we implement a modified version of the branch and bound algorithm based on judiciously discarding parts of the state space leveraging on the system behavior (e.g., if a target probability failure T_f is not met with a given threshold t_m^{on}, any positive increase of t_m^{on} will neither met T_f). While the configuration algorithm takes on average between 20 sec and 37 sec to compute the optimal configuration, the exhaustive search takes up to 3 hours for the case of the rack servers and more than 10 days for the case of nano servers. The results are shown in Fig. 7 where, for easy viewing, we use linepoints for our algorithm (solid lines for rack servers, dashed lines for nano servers), and circles for the exhaustive search.

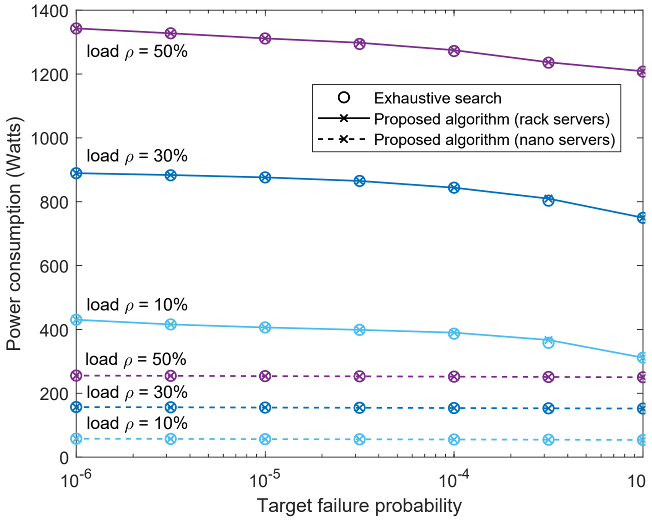
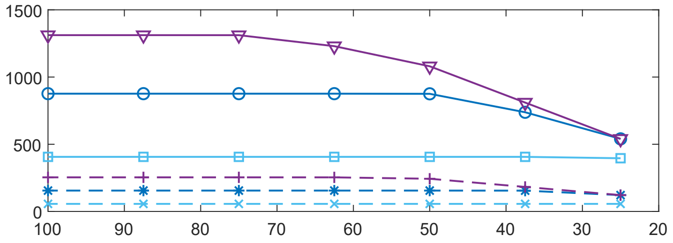


Fig. 7: Optimal power consumption for different requireme and system loads.



(b) Power consumption (W) versus percentage of available servers

Fig. 8: Performance when M is reduced.

According to the results, the performance obtained with our configuration algorithm is practically identical to the one obtained after an exhaustive search in the whole configuration space: the maximum difference of our algorithm from the exhaustive search is 2.5%, whereas the average difference is 0.12%. Like before, the power consumption decreases when the reliability guarantees are smaller in both types of deployments, although in the nano servers case it is less evident due to the logarithmic scale of the x axis. Regarding the impact of the load, we observe the same qualitative behavior presented in Fig. 6, with the power consumption being proportional to the load. Finally, it is worth remarking that, for the configurations considered, the power consumption of the nano servers deployment is always smaller than the one of the rack servers.

D. Variable number of servers

We next analyze how performance varies when the number of servers M changes. Firstly, we consider the impact of a reduction in M on performance. More specifically, we assume a target failure probability $T_f = 10^{-5}$ and both server farms optimally configured, and analyze the resulting value of P_f and ω when the percentage of available servers is reduced from 100% to 20%. The results are depicted in Fig. 8, for the same system loads as above. Regarding P_f (Fig. 8a), the results show that the smaller the load ρ, the smaller the number of servers strictly required to ensure the target T_f: for instance, a 10% load can be supported even with less than half of the servers in the original deployment, while for a 50% load at least 85% of the servers are required. Regarding ω (Fig. 8b), when there are enough resources the consumption only depends on the value of T_f and the load, while when M is reduced to a certain level the value of ω corresponds to the consumption of the remaining servers operating at their full capacity.[6]

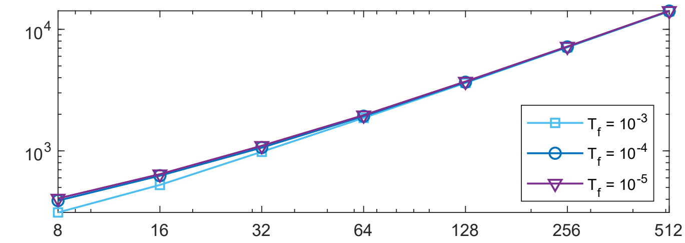
We next analyze performance for larger deployments. To this aim, we assume the rack servers configuration and increase the number of servers M, assuming a relative load of 10% and three target failure probabilities $T_f = \{10^{-3}, 10^{-4}, 10^{-5}\}$. For each value of M the computation of the parameters of (44) can take hours (for the largest configurations), while the optimization itself including the Algorithm 1 is solved within minutes. We compute the resulting failure probability P_f and the power consumption ω, with the results being depicted in Fig. 9. The figure confirms that our algorithm guarantees that the resulting P_f (Fig. 9a) is below T_f for all considered values of M. Regarding ω (Fig. 9b), for small values of M (up to 64 servers), it can be seen that the power consumption increases with T_f as before; for larger values of M, the impact of T_f is harder to notice due to the log scales.

E. Variable load

In this section we apply our algorithm to a scenario with variable load. Since the optimal thresholds are challenging to compute in real time, these are precomputed for different target probabilities and traffic loads. For this specific example, we precompute the optimal thresholds for a target failure probability $T_f = 10^{-4}$ and arrival rates corresponding to traffic loads in steps of 10%, i.e. $\rho \in \{10\%, 20\% \ldots\}$. The algorithm then monitors the task interarrival time to estimate the arrival rate λ using a simple exponentially weighted moving average (EXMA) with $\alpha = 0.99$, and uses the precomputed thresholds corresponding to the load closest to the estimated one.

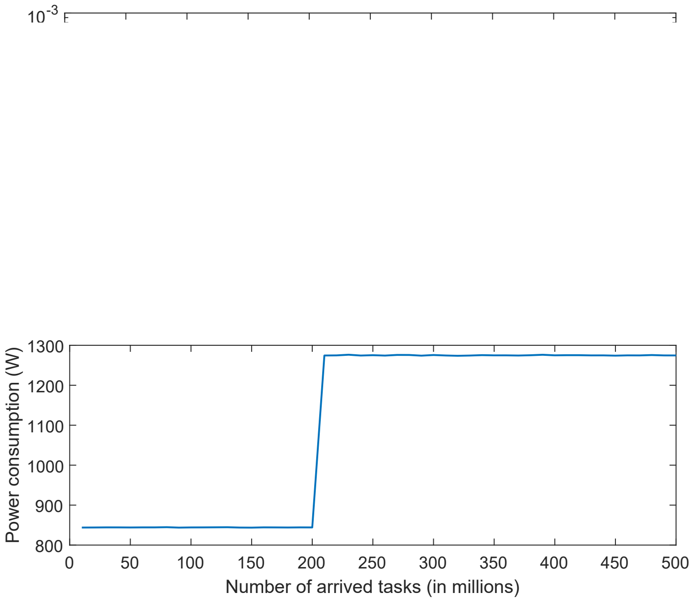
We simulate the arrival of 500 million tasks, the first 200 million with an interarrival time corresponding to a load of $\rho = 30\%$, and the rest with an interarrival time corresponding

[6] We note that although these results confirm that smaller servers might be more energy efficient for a given target reliability, our results are limited to the parameters considered in Table II and do not consider key factors such as e.g. physical constrains or maintenance.



(b) Power consumption (W) versus number of servers

Fig. 9: Performance when M increases.



(b) Power consumption (W) with variable load

Fig. 10: Performance when the load varies.

to a load of $\rho = 50\%$. Fig. 10a confirms the effectiveness of the approach, as the resulting P_f, averaged over windows of 10 million tasks, shows minor variations around the target value T_f, and the average P_f over the whole simulation is $0.96 \cdot 10^{-4}$. Fig. 10b illustrates the increment in ω as the load increases, in line with the results in Fig. 7. While this simple approach serves to confirm the practicality of our approach for scenarios with variable load, we leave as future work the use of more sophisticated schemes, based on e.g., server load estimation techniques such as the ones discussed in [38].

Approach	Config.	Description
Heuristic [39]	Conf. 1	MM: low=50%, high=90%
	Conf. 2	MM: low=40%, high=80%
Q-learning [40]	Conf. 1	$\beta = 0.8$
	Conf. 2	$\beta = 1.0$

TABLE III: Approaches considered.

F. Comparison with other approaches

To put the performance of the optimization algorithm in context, we next compare it versus other alternatives. Since no previous work provides a configuration mechanism to guarantee a given reliability level, we consider the two schemes summarized in Table III, each with the configurations used by the respective papers where they were proposed.[7] These two schemes are summarized next:

Heuristic: This algorithm performs an energy-aware allocation of virtual machines (equivalent to the tasks in our proposal) to devices in a data center [39]. The algorithm uses two occupation thresholds to activate or deactivate physical resources as the load in the data center varies. Since authors do not propose any scheme to compute these thresholds, we select the best performing policy and configurations, namely, the so-called the minimization of migrations (MM) policy, and the following pair of occupation thresholds: $\{50\%, 90\%\}$ and $\{40\%, 80\%\}$.

Q-learning: we adapt the approach proposed in [40], which uses Q-learning (a technique receiving a lot of attention for networking problems, see e.g., [41]) to minimize the consumption of resources while aiming for a given Service Level Agreement (SLA). The state is defined by the number of tasks and active servers, and at each update interval (which we set to half the inter-arrival time), an action is taken, namely, to switch on a server, switch it off, or keep the current number of active servers. The penalty is given by the a weighted sum of the number of failures during the interval (n_t) and the normalized power consumption over the same period (ω_t) as follows

$$p_t = \beta n_t + (1-\beta)\omega_t, \tag{45}$$

where the β parameter serves to tune the trade-off between reliability and power consumption. As in the case of [39], since performing an exhaustive search on the whole configuration space would result impractical, for this and the rest of the parameters we use the same values as defined in [40], i.e., $\beta = \{0.8, 1\}$.

For each of these algorithms and configurations, and considering different traffic loads and the rack servers configuration, we compute their resulting failure probability and power consumption, as well as those obtained with our algorithm assuming $T_f = 10^{-5}$. The results are summarized in Fig. 11, where the target T_f is highlighted with a dashed line, each approach is represented with a different symbol, and each traffic load is represented with a different color. The results confirm that using our approach the system achieves the

[7] Since these methods adopt fixed configurations, there is no computational time required to obtain them.

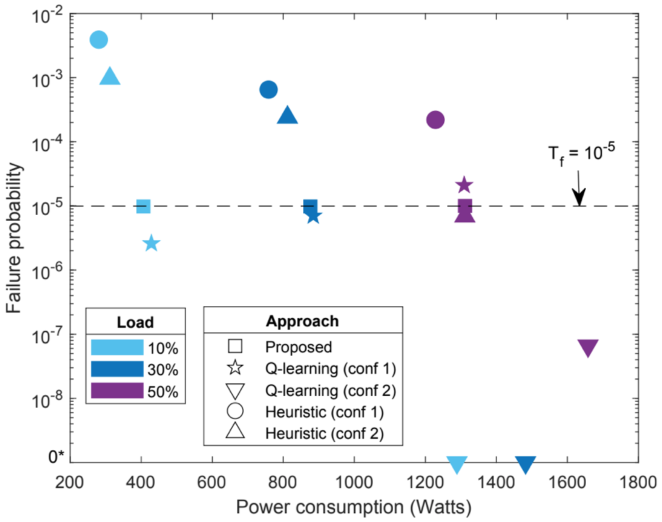


Fig. 11: Comparison versus other approaches.

desired performance, as the square symbols are aligned with the target T_f. In contrast, the other schemes do not provide the means to drive the failure probability to the desired target. Furthermore, as they rely on parameters which need to be configured heuristically, the resulting performance is uncertain: in some cases the achieved probabilities are way larger than those considered in 5G networks, while in other cases they are overly small thus leading to a wastage of energy.

G. *Trace-based evaluation*

We finalize our performance evaluation by assessing the accuracy of the model under real traffic. To this end, we perform a similar experiment to that presented in Fig. 5, taking real-life workload as input traffic and evaluating the failure probability against the model. In particular, the behavior of jobs arriving at the system is derived from a trace of the Trinity supercomputer located at the Los Alamos National Laboratory [42], both for the time between arrivals and the time a request spends in the system. This trace comprises information of jobs issued during three months to a large-scale supercomputer system [43]. To analyze the system in the periods under the largest activity, we have removed the 25% of the longest inter-arrival times from the trace.

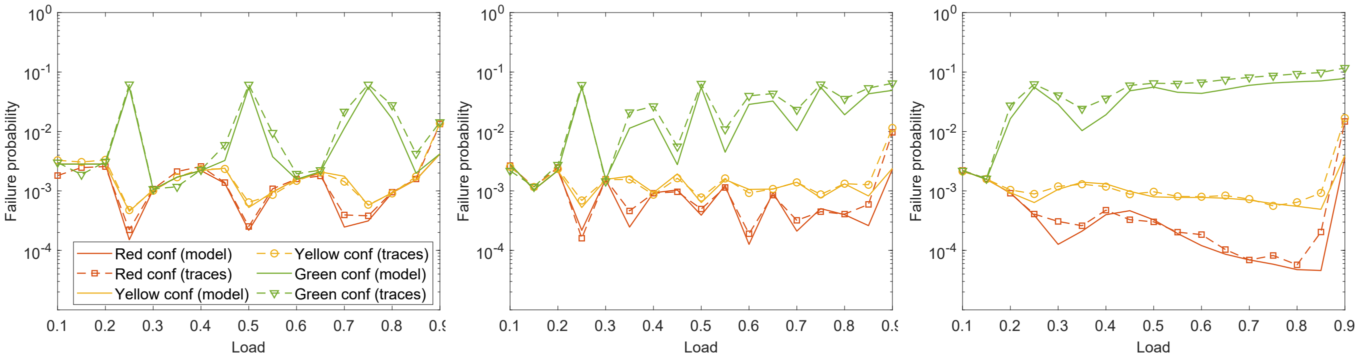
To evaluate the performance of the model under different loads, we change the configuration of M and N as follows: we set the number of servers to $M = \{4, 8, 16\}$ and perform a sweep on the capacity per server N such as the load $\rho = \lambda/(MN\mu)$ is adequately sampled. The results are shown in Fig. 12 for the same green, yellow and red configurations as in Fig. 5. For ease of visualization, we use solid lines to represent the results from the analytical model, and dashed linepoints to represent the trace-based simulation results.

The results confirm the accuracy of the model under real-world traces, as the results from the analysis closely follow those from simulations, with the median error across all configurations being 13%. We can observe that the three policies provide different levels of reliability, the red one providing the best reliability and the green one the worst. This configuration also leads to the largest differences between the simulation and the analysis, since key assumptions of our model (e.g., independence of failure types) become less accurate as the failure probability increases. We further observe that the behavior of the reliability with the load is non-monotonic, specially for the $M = 4$ case. Here, whenever the load ρ approaches a multiple of $1/M$, i.e., for $\rho = \{0.25, 0.5, 0.75\}$, the performance of the green configuration severely degrades, as only when all servers are 100% occupied a new resource is activated. In contrast, for the yellow and red configurations this activation is anticipated, thus leading to a relative performance improvement. For the other M cases the effect is less evident given the considered sampling of the load.

In summary, these results show that, even though our model has been derived under some assumptions that may not always hold, the model remains fairly accurate under realistic traffic patterns.

VI. Related work

Algorithms for scaling server farms. The dynamic management of cloud resources to reduce consumption has received notable attention from the research community. For instance, in [44], authors assess the impact of different static algorithms to (de)activate resources and reallocate tasks in a data center, in terms of energy consumption and so-called service violations, defined as when a task is not provided with enough resources. In a series of follow-up papers, [33], [39], [45], they propose heuristics to adapt the thresholds to the estimated conditions (e.g., occupation), to reduce these service violations. Although these works are based on a similar threshold-based operation as ours, they do not provide any algorithm to compute them, thus leading to sub-optimal performance and/or the need to perform numerical searches. The Plug4Green algorithm [46] takes as input the SLAs to identify constraints, with the goal of reducing power consumption. However, the SLAs do not include reliability (they include e.g. hardware variables such as the minimum number of CPUs, or network guarantees such as minimum bandwidth). Along the same lines, [29] presents a framework to accommodate tasks in servers, also with goal of guaranteeing SLAs while reducing the energy consumption, but again the reliability is not considered. As mentioned before, one key difference between our work and the cloud techniques from the literature including the ones above is that they cannot provide high levels of reliability such as the ones required in 5G (see e.g. [12] for a recent survey). Some approaches use of machine learning to tune the auto-scaling of server farms. The work in [40] 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 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 [47] for a similar purpose, although based on a more complex penalty function, which suffers from similar issues to the ones of [40].



(a) M=4 servers

(b) M=8 servers

(c) M=16 servers

Fig. 12: Failure probability versus load from real traces

Queueing models. In [48] 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, but in the context of 5G networks, is presented in [49], 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 take into account the finite lifetime of servers nor the failure probability of the service, as we do in this paper. Other approaches, such as parallel queues with vacations [50], [51], do not fit with the considered problem as they do not include a policy to (de)activate servers depending on the number of tasks in the system.

Reliability of server farms. The seminal work of [25] 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 [35], where authors analyze the reliability of a virtualized and non-virtualized system composed of two hosts. A related system is later considered in [36], where a *sensitivity analysis* is performed to identify the parameters that more critically affect reliability. None of these micro-scale models consider a server farm with an auto-scaling policy, nor its optimal configuration. In contrast, we analyze a generic server farm composed of a number of servers where each one is characterized by a lifetime, and derive the configuration of its auto-scaling policy that minimizes energy consumption while ensuring a given minimum reliability. Other proposals such as [52] or [53] aim at improving reliability by replicating tasks and selecting the most appropriate server (based on a failure model for the power outages and network components), but they do not consider energy consumption.

NFV and reliability. The softwarization of networks introduces a great deal of flexibility to decide 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 [54], with the aim of maximizing resource efficiency while providing *mild* service guarantees (e.g., CPU availability, inter-VNF delays). For instance, [55] presents a mixed integer linear programming to minimize the power consumption while guaranteeing traffic constrains which optimizes the VNF locations. In contrast to these works, we present an analytical framework and optimization algorithm for an auto-scaling scheme to ensure a strict and accurate guarantee (i.e., keep the failure probability below a given and very small threshold). For the particular challenge of providing a reliable service, we can highlight: [56] presents some heuristics for a routing optimization problem, where redundant VNFs are used to ensure an average reliability level; [57] relies on machine learning to predict the next failure of a server, based on the knowledge of past failures, to trigger a proactive launch of a novel virtual machine; and [58], analyzes the decomposition mobile applications in several components and their best placement in the edge, and also proposes the use of inactive copies to support reliability. In contrast to these works, we focus on the analysis of a fine-grained metric and provide the means to optimally configure the deployment to provide very stringent levels. The work of [59] also addresses the reliability of a softwarized deployment, but the focus is on the understanding of the interconnection of virtual and physical resources. More specifically, the contribution is a framework to diagnose and mitigate the so-called *cascading effect*, i.e., when a single node failure results in large-scale network collapse.

VII. Conclusions

The softwarization of mobile networks introduces the ability to scale resources as demand varies. However, to support the stringent reliability guarantees required by some 5G applications, enough resources should be activated in advance, both to mitigate the impact of start-up delays on performance, and to introduce redundancies to lessen the impact of server failures. In this paper, we have modeled the behavior of a server farm with an auto-scaling mechanism based on (de)activation thresholds. We have characterized its failure probability and energy consumption for a wide range of scenarios, including different deployment configurations, and derived an optimal configuration algorithm that fulfills a given service level agreement while optimizing performance. The accuracy of the model and the optimality of the algorithm have been extensively validated via simulations. Furthermore, by comparing the performance of two different deployment strategies optimally configured, we have shed some light on whether it is better to support a given service with a few carrier-grade machines or many low-power servers. While our results suggest that deploying enough less powerful machines could be a valid and more efficient strategy, our analysis has not considered key aspects such as the management costs and therefore the final decision depends on the criteria to optimize.

Acknowledgements

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. The work of P. Serrano was partly funded by the European Commission (EC) through the H2020 project Hexa-X (Grant Agreement no. 101015956). The work of J. Garcia-Reinoso was partially supported by the EC in the framework of H2020-EU.2.1.1. 5G EVE project (Grant agreement no. 815074). The work of A. Banchs was partially supported by the EC in the framework of H2020-EU.2.1.1. 5G-TOURS project (Grant agreement no. 856950). The work of P. Serrano and A. Banchs was also partially supported by the Spanish State Research Agency (TRUE5G project, PID2019-108713RB-C52PID2019-108713RB-C52/AEI/10.13039/501100011033).

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[2] T. Norp, “5g requirements and key performance indicators,” *Journal of ICT Standardization*, vol. 6, no. 1, pp. 15–30, 2018.

[3] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, “Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions,” *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[4] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker, “Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks,” *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017.

[5] M. Gramaglia, P. Serrano, A. Banchs, G. García, A. Garcia-Saavedra, and R. Perez, “The case for serverless mobile networking,” in *IFIP Networking 2020 - Network Slicing workshop*, June 2020.

[6] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, “How Should I Slice My Network?: A Multi-Service Empirical Evaluation of Resource Sharing Efficiency,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking - MobiCom ’18*. New Delhi, India: ACM Press, 2018, pp. 191–206.

[7] 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.

[8] 3rd Generation Partnership Project (3GPP), “Technical Specification Group Services and System Aspects; Management and orchestration; Provisioning,” TS 28.531, v. 15.0.0, 2019.

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

[10] 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.

[11] G. Berardinelli, N. H. Mahmood, I. Rodriguez, and P. Mogensen, “Beyond 5g wireless irt for industry 4.0: Design principles and spectrum aspects,” in *2018 IEEE Globecom Workshops (GC Wkshps)*, 2018, pp. 1–6.

[12] 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, 05 2019.

[13] F. P. Tso, D. R. White, S. Jouet, J. Singer, and 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.

[14] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, and 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.

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

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

[17] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, “A comparative study of containers and virtual machines in big data environment,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 178–185.

[18] T. L. Nguyen, “Fast delivery of virtual machines and containers : understanding and optimizing the boot operation,” Theses, Ecole nationale supérieure Mines-Télécom Atlantique, Sep. 2019. [Online]. Available: https://tel.archives-ouvertes.fr/tel-02418752

[19] “Chapter 5 - windows, linux, and macintosh boot processes,” in *The Official CHFI Study Guide (Exam 312-49)*, D. Kleiman, K. Cardwell, T. Clinton, M. Cross, M. Gregg, J. Varsalone, and C. Wright, Eds. Rockland: Syngress, 2007, pp. 265–285. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781597491976500067

[20] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, “Voyager: Complete Container State Migration,” in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2137–2142.

[21] G. Garcia-Aviles, C. Donato, M. Gramaglia, P. Serrano, and A. Banchs, “Acho: A framework for flexible re-orchestration of virtual network functions,” *Computer Networks*, vol. 180, p. 107382, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128619317165

[22] P. Serrano, M. Gramaglia, D. Bega, D. Gutierrez-Estevez, G. Garcia-Aviles, and A. Banchs, “The path toward a cloud-aware mobile network protocol stack,” *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 5, p. e3312, 2018, e3312 ett.3312. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3312

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

[24] K. S. Trivedi and A. Bobbio, *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge University Press, 2017.

[25] 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.

[26] R. Garg, M. Mittal, and L. H. Son, “Reliability and energy efficient workflow scheduling in cloud environment,” *Cluster Computing*, vol. 22, no. 4, pp. 1283–1297, Feb. 2019. [Online]. Available: https://doi.org/10.1007/s10586-019-02911-7

[27] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and 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*, ser. NSDI’08. USA: USENIX Association, 2008, p. 337–350.

[28] H. A. A-Shehri and K. Hamdi, “Multi-objective vm placement algorithms for green cloud data centers: An overview,” in *2018 21st Saudi Computer Society National Computer Conference (NCC)*, 2018, pp. 1–8.

[29] H. Wang and H. Tianfield, “Energy-aware dynamic virtual machine consolidation for cloud datacenters,” *IEEE Access*, vol. 6, pp. 15 259–15 273, 2018.

[30] M. Guazzone, C. Anglano, and M. Canonico, “Exploiting vm migration for the automated power and performance management of green cloud computing systems,” in *International Workshop on Energy Efficient Data Centers*. Springer, 2012, pp. 81–92.

[31] A. Kumar, A. Sharma, and R. Kumar, “Servmegh: framework for green cloud,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 4, p. e3903, 2017.

[32] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains*. Wiley-Interscience, 2006.

[33] 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. [Online]. Available: https://doi.org/10.1145/1890799.1890803

[34] G. Gens and E. Levner, "An approximate binary search algorithm for the multiple-choice knapsack problem," *Information Processing Letters*, vol. 67, no. 5, pp. 261 – 265, 1998.

[35] 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.

[36] 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.

[37] G. L. Santos, P. T. Endo, G. Gonçalves, D. Rosendo, D. Gomes, J. Kelner, D. Sadok, and 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.

[38] M. Masdari and A. Khoshnevis, "A survey and classification of the workload forecasting methods in cloud computing," *Cluster Computing*, vol. 23, no. 4, pp. 2399–2424, 2020. [Online]. Available: https://doi.org/10.1007/s10586-019-03010-3

[39] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[40] 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.

[41] 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.

[42] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben, "Workflow trace archive lanl_trinity trace," Jun 2019.

[43] ——, "On the diversity of cluster workloads and its impact on research results," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 533–546.

[44] 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.

[45] A. Beloglazov and 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*, vol. 24, no. 13, pp. 1397–1420, 2012.

[46] C. Dupont, F. Hermenier, T. Schulze, R. Basmadjian, A. Somov, and G. Giuliani, "Plug4green: A flexible energy-aware vm manager to fit data centre particularities," *Ad Hoc Networks*, vol. 25, pp. 505–519, 2015, new Research Challenges in Mobile, Opportunistic and Delay-Tolerant Networks Energy-Aware Data Centers: Architecture, Infrastructure, and Communication. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870514002376

[47] 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.

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

[49] Y. Ren, T. Phung-Duc, J. Chen, and Z. Yu, "Dynamic auto scaling algorithm (dasa) for 5g mobile networks," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

[50] M. F. Neuts and D. M. Lucantoni, "A markovian queue with n servers subject to breakdowns and repairs," *Management Science*, vol. 25, no. 9, pp. 849–861, 1979.

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

[52] M. Sedaghat, E. Wadbro, J. Wilkes, S. De Luna, O. Seleznjev, and 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.

[53] M. Korupolu and R. Rajaraman, "Robust and probabilistic failure-aware placement," *ACM Transactions on Parallel Computing (TOPC)*, vol. 5, no. 1, pp. 1–30, 2018.

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

[55] 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, 2019.

[56] 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. [Online]. Available: http://dx.doi.org/10.1109/TNSM.2017.2723090

[57] H. Huang and S. Guo, "Proactive failure recovery for NFV in distributed edge computing," *IEEE Commun. Mag.*, vol. 57, no. 5, pp. 131–137, 2019. [Online]. Available: https://doi.org/10.1109/MCOM.2019.1701366

[58] L. Dong, W. Wu, Q. Guo, M. N. Satpute, T. Znati, and D. Z. Du, "Reliability-aware offloading and allocation in multilevel edge computing system," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 200–211, 2021.

[59] 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. [Online]. Available: http://dx.doi.org/10.1109/MWC.2016.7498079

[60] M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. Dover Publications, 1995.

APPENDIX: CONVERGENCE OF THE MARKOV CHAIN

Theorem 1. *The QBD process is stable as long as*

$$\frac{\lambda}{NM\mu} < \frac{\alpha}{\nu + \alpha}. \tag{46}$$

Proof. The convergence of the QBD process will be guaranteed as long as the drift of the system to higher levels is smaller than the drift to lower levels. Mathematically, this condition translates to [60]

$$\boldsymbol{\pi}_A \mathbf{A}_0 \mathbf{1} < \boldsymbol{\pi}_A \mathbf{A}_2 \mathbf{1}, \tag{47}$$

where $\boldsymbol{\pi}_A$ is the steady-state probability vector of the generator matrix $\mathbf{A} = \mathbf{A}_0 + \mathbf{A}_1 + \mathbf{A}_2$, leading to the tridiagonal matrix

$$A = \begin{bmatrix} - & M\alpha & 0 & & \\ \nu & - & (M-1)\alpha & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & - & \alpha \\ & & & M\nu & - \end{bmatrix} \tag{48}$$

where the computation of the diagonal elements is straightforward via the normalization condition. The resulting matrix corresponds to a BD process of $M + 1$ states, with $\boldsymbol{\pi}_A$ given by

$$\boldsymbol{\pi}_A = [p_0, p_1, \ldots, p_M] \tag{49}$$

and illustrated in Fig. 13. The balance equations result in the following relation among consecutive states

$$p_i = \frac{(M - i + 1)\alpha}{i\nu} p_{i-1}, \tag{50}$$

which results in

$$p_i = \binom{M}{i} \left(\frac{\alpha}{\nu}\right)^i p_0. \tag{51}$$

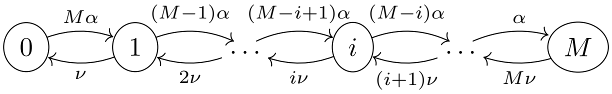


Fig. 13: BD process corresponding to the matrix A (48).

From the normalization condition $\sum p_i = 1$ we have

$$p_0 \sum_{i=0}^{M} \binom{M}{i} \left(\frac{\alpha}{\nu}\right)^i = 1. \tag{52}$$

The summation in the LHS of the previous equation corresponds to the binomial expansion of the polynomial $(1+\alpha/x)^M$. Therefore

$$p_0 = \left(1 + \frac{\alpha}{\nu}\right)^{-M}. \tag{53}$$

With the above, the computation of $\boldsymbol{\pi}_A$ is completed, so we can use its value on the condition given by (47). The LHS of the equation results in

$$\begin{aligned}\boldsymbol{\pi}_A \mathbf{A}_0 \mathbf{1} &= [p_0 \dots p_M] \begin{bmatrix} \lambda & 0 & \\ & \ddots & \\ & 0 & \lambda \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \qquad (54) \\ &= \lambda \sum_i p_i = \lambda,\end{aligned}$$

whereas the computation of the RHS of (47) leads to

$$\begin{aligned}\boldsymbol{\pi}_A \mathbf{A}_2 \mathbf{1} &= [p_0 \dots p_M] \begin{bmatrix} 0 & & & \\ & N\mu & & \\ & & \ddots & \\ & & & MN\mu \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \qquad (55) \\ &= N\mu \sum_i i p_i = N\mu p_0 \sum_{i=1}^{M} i \binom{M}{i} \left(\frac{\alpha}{\nu}\right)^i. \qquad (56)\end{aligned}$$

Using again the binomial expansion and substituting the value of p_0, the above can be simplified as

$$\boldsymbol{\pi}_A \mathbf{A}_2 \mathbf{1} = N\mu M \frac{\alpha}{\nu + \alpha}. \tag{57}$$

By combining (54) and (57), (47) can be expressed as

$$\lambda < N\mu M \frac{\alpha}{\nu + \alpha}, \tag{58}$$

which completes the demonstration of the Theorem. □

Jorge Ortin is currently 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 and analysis of communications systems.

Pablo Serrano (M'09, SM'16) received his degree in telecommunication engineering and his Ph.D. from Universidad Carlos III de Madrid (UC3M) in 2002 and 2006, respectively. He has been with the Telematics Department of UC3M since 2002, where he currently holds the position of associate professor. He has over 100 scientific papers in peer-reviewed international journal and conferences. He regularly serves TPC member of a number of conferences and workshops.

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 Univ. of Alcala since 2021 and he has published over 60 papers in the field of broadband computer networks in top magazines and conferences. He has been involved in several international and national projects.

Albert Banchs (M04-SM12) received his M.Sc. and Ph.D. degrees from the Polytechnic University of Catalonia in 1997 and 2002, respectively. He worked at ICSI in 1997, Telefonica I+D in 1998 and NEC Laboratories from 1998 to 2003. Since 2003 he is with the University Carlos III of Madrid, Spain, where he is currently Full Professor and has a double affiliation as Deputy Director of the IMDEA Networks institute. Prof. Banchs has served in many conference TPCs (including IEEE INFOCOM) and journal editorial boards (including IEEE/ACM Transactions on Networking and IEEE Transactions on Wireless Communications).