Power of random choices made efficient for fog computing

Roberto Beraldi, Gabriele Proietti Mattia
DIAG, La Sapienza University of Rome, Italy

Abstract—In this paper, we consider a load balancing protocol based on the power of random choices that is adapted to a fog deploy in which several independent fog nodes equipped with a set of servers or VM are serving the same geographical area. The protocol is based on a simple but effective mechanism based on a threshold T. When a fog node receives a unit of computation or a job, it immediately executes the job if the number of its occupied servers is *lower* than T, otherwise the node executes a randomized algorithm by probing F other fog nodes in the area, and delegates the execution of the job to the least loaded one, provided the workload is *lower* than the probing node.

Through a mathematical analysis we show that probing just one node ($F = 1$) when there are less than two free VMs provides the same performance of the well known power-of-two random choices centralized algorithm, but at a much lower delay and control overhead costs. Also, simulations are used to address the node heterogeneity and, with a real testbed, we offer results that prove the effective benefit of the proposed solution in practical applications.

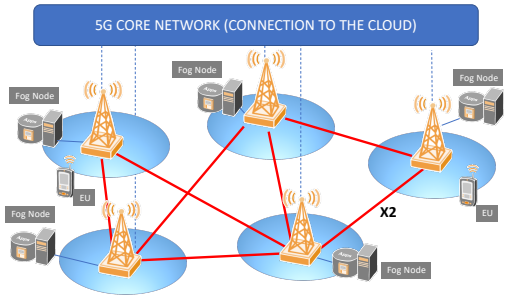


Figure 1: An illustration of the inter-node communication among 5G fog nodes, that can support the load balancing discussed in the paper.

1 Introduction

The ongoing advances in the ICT, from 5G to powerful open-source software libraries, e.g., open Computer Vision and TensorFlow, are creating the prerequisites for software applications positioned in the left upper corner of the latency bandwidth Cartesian product with use cases in different vertical domains, e.g., ranging from IoT, smart cities, AR/VR based applications with haptic interactions, as well as Tactile Internet, [1]. Fog computing is widely considered the key enabler for such applications as it makes backend capabilities physically close to the end-users and integrated with the usual cloud services. Fog computing is seen as an integrated intermediate layer along the thing-to-cloud path [2], [3], [4], [5]; readers may refer to [6] for an overview and tutorial on fog computing.

Fog enabled applications are a composition of units of computation, e.g., adhering the Function as a Service (FaaS) delivery model, with the time-critical parts deployed on fog nodes, [7]. OpenFog's Fog computing Reference Architecture [8], recently adopted as the IEEE-1934 standard, envisions fog nodes to form a mesh to provide load balancing and minimization of cloud communication. In particular, they may communicate laterally in a peer to peer fashion. For example, in the 5G architecture, a high number of devices are networked by a capillary network, and nearby Radio Access Network, equipped with Fog computing capabilities, (F-RAN) may communicate directly, through the so-called X2 or X2* interfaces, [9], [10], see Figure 1. This inter-node communication carries both control and user planes, and allows for increasing the fraction of the service requirements that can be responded locally, without interacting with the cloud computing center via the fronthaul links. For example, a computation request generated from an End User (EU) connected to an F-RAN can be served by another nearby F-RANs, even via a virtual multi-hop connection, where the number of hops is limited to some units.

Also, in the industrial standard ETSI's Multi-access Edge Computing (MEC) reference architecture, the envision orchestration leverages on code relocation and traffic steering to dynamically move application code or data, [11].

Motivated by the above scenarios, in this paper, we study the problem of how load balancing jobs among a set of fog computation resources that provide the same computation service, for example, the object detection service described in [12]. Load balancing in fog computing is, in fact, an open issue that may play an important role to enable fog based applications, [6], [13], [14], [15].

In more detail, we focus on a fog layer composed of N nearby fog nodes, with K identical servers. The fog nodes are willing to share their resources with the goal of reducing the fraction of jobs forwarded for execution to a distant cloud. Incentives to cooperate and the benefit of such sharing are discussed for example, in [16]. In addition, jobs

have no information related to their deadlines, computation requirements or priorities. It worth to note that N can potentially be high. For example, in the 5G architecture, the higher frequencies than 4G are not capable of traveling large distances. This requires placing 5G RAN every few hundred meters in order to utilize such higher frequency bands.

The proposed algorithm is an adaptation of the $LL(d)$ algorithm (Least Loaded among d random nodes) to the above fog model. A fog node normally executes jobs received from its direct users without any load balancing action. However, when its workload is higher than a threshold T, a node tries to delegate the execution of a job to another less loaded node among F randomly probed nodes [1]. We refer to this algorithm as $LL(F,T)$. This threshold regulation is a simple yet effective mechanism that reduces remote scheduling overhead remarkably, avoiding the inefficiency arising when autonomous schedulers compete on sharing a common set of resources, [17], [18], [19].

Contribution of the paper:

- Definition of $LL(F,T)$, Least Loaded among F nodes with threshold T, a new scheduling algorithm the fog computing model based on the power-of-random choices randomization principle.
- Mathematical analysis of the algorithm showing how $LL(1,K-2)$, i.e., probing just one node when one or two servers out of the K are idle, reduces up to one order of magnitude the control and delay overhead with respect to a vanilla randomized load balancing implementation of the power-of-two random choices algorithm.

The paper is organized as follows. In the next section, we report related works. Section III proposes $LL(F,T)$ and its performance model, while in Section IV we report a study on the effect of T on the behavior of randomization, and in Section V performance results. Conclusions are then given in Section VI.

2 Related Work

Our proposal has its root in the randomized algorithms based on the power-of-d choices result, [4]. Load balancing algorithms based on this result, adopt a unique scheduler that receives jobs to be dispatched to one of N equivalent workers. Scheduling decisions are performed by sampling the state of d workers and selecting the most convenient one according to their workloads. The main success of this strategy is that it avoids keeping the state of all the workers while being remarkable effective, which is particularly useful when N is high. A rich and sound literature has studied the property of these algorithms for $N \to \infty$. In this limit [20] shows the asymptotic independence among queues, which allows simplifying the analysis of these systems. Our analysis assumes this ansatz.

The studies that characterize this randomization algorithm in the context of load balancing can be divided into two bodies depending on the model of the workers, that either cannot or can lose jobs. The first set of results, e.g., [4], [21], [22], model workers as M/M/1 queues and the selection criteria is to pick the Shortest among the d sampled Queues, $SQ(d)$. The most relevant finding is that the average job execution delay decay doubly exponentially in the limit as the number of servers goes to infinity, which is a substantial improvement over a classical queue case, where the queue size decays exponentially. A finer workload measurement is used in [23]. This extraordinary improvement has motivated the research on a different worker model, that can somehow better fit applications to cloud computing, e.g., [24]. The second body of works, in fact, model the worker as a finite set of K servers, e.g., an M/M/K queue, and a job that cannot be scheduled it is blocked, [25], [24], [26], [22]. The selection criteria here is to pick the Least Loaded among d queues, $LL(d)$, where the server load is the number of busy servers. These works analyze the effect of randomization on the blocking probability, which is the counterpart of average delay. However, all models assume a single centralized dispatcher, and hence threshold-based control cannot be applied. The work in [27] is the most one related to our idea. In the studied protocol if a job execution request arrives at an overloaded fog node, the job is forwarded to a neighboring node with some probability. Hence, contrary to our protocol no power-of-choice is used. In [28] we have used power-of-random choices to enable cooperation among different fog providers. No thresholds are used and the protocol is limited to unitary fanout. In [29] random choices are used for a p2p load balancing protocols.

Load balancing for the fog model has been studied in several papers. In [30] an algorithm called Multi-tenant Load Distribution Algorithm for Fog Environments (MtLDF) has been proposed to optimize load balancing in Fogs environments considering specific multi-tenancy requirements. However, the proposed load balancing scheme adopts a centralized fog management layer that receives all the state information about the fog nodes.

In [31], the tasks that the nodes are called to complete, are characterized according to their computational nature and are subsequently allocated to the appropriate host. Edge networks communicate through a brokering system with IoT systems in an asynchronous way via the Pub/Sub messaging pattern. However, a centralized workload balancer is used in the solution.

In [32] an approach is presented to periodically distributing incoming tasks in the edge computing network so that the number of tasks, which can be processed in the edge computing network, is increased, and the quality-of-service (QoS) requirements of the tasks completed in the edge computing network are satisfied. The model, however, assumes that a set of tasks to be assigned is available, i.e., the tasks are not processed online.

In [33] a mechanism for load balancing policy is presented with a dynamic threshold, which is computed each time the scheduling is applied.

In [34] and [35] the load balancing algorithm is based on a BFS search, by also addressing the problem of the secure authentication between nodes. Other works are focused on the trade-off between energy consumption and latencies [36].

1. We use a different symbol, F to indicate the number of probed nodes to make the difference with the original protocol clearer.

3 Power of random choices load balancing for Fog Computing

In this section, we describe our solution to the load balancing problem for fog computing. In general terms, the load balancing problem consists in determining how to allocate jobs generated by end-users to fog nodes in an efficient way. More formally, we are given a set of N homogeneous fog nodes with limited resource capacity, where each one is receiving a continuous flow of computation requests, or jobs. A job is blocked when it cannot be executed due to the lack of available resources. We call p_B the probability that this event occurs. A load balancing algorithm is a rule that assigns jobs to servers in a way that p_B is minimized. The efficiency of the rule is measured by the amount of control overhead generated.

A power-of-d random choices load balancing algorithm is a rule executed by a dispatcher to decide where the jobs received have to be forwarded and executed, among a set of N equivalent workers. On receiving a job, the dispatcher probes a small subset d of workers and then sends the job to the Least Loaded among them with ties broken at random. The key characteristic of the algorithm is $d \ll N$ and even $d = 2$ has shown to be very effective compared to probing all nodes. The algorithm, referred as $LL(d)$ requires an overhead as small as d control messages per job and scalable as this overhead doesn't depend on N.

A general fog computing deploy model is a hierarchical multi-tier architecture where communication can also occur among nodes of the same tier, [8]. We focus on the first tier that provides access to end-users via N nodes. The application of $LL(d)$ as-is implies the existence of N dispatchers and N workers, and this has two main drawbacks. The first one is that in general concurrent dispatchers tend to be inefficient as their decisions are biased towards workers that appear lightly loaded, [17], [19]. In some extreme cases, this implementation may even deteriorate the initial performance, e.g., see [18]. The second implication is that the dispatching operations introduce a delay overhead for any job, even when nodes do not need to distribute the load, for example, because most of its current servers are idle, as discussed later in the text. For these reasons, we propose the following algorithm that keeps the load balancing approach unchanged but allows reducing the aforementioned drawbacks. The design of the algorithm is based on the following assumptions:

- A1: The communication latency among nodes, although not negligible, is small compared with the job execution time. This assumption follows from the observation that for example, the time required for an image recognition is in the order of tens of ms, while an X2 or X2* interface that connects Fog nodes - either directly or through a few relaying nodes, is likely to be not higher than a few ms, see Figure 1.
- A2: Probing occurs only among N nodes in the same tier composed of nearby fog nodes, where N can be high. This assumption follows from the observation that the density of fog nodes in a geographical can be high. For example, F-RAN can cover just hundreds of meters. If a job cannot be executed in the tier, the job is forwarded to the cloud. The protocol aims at reducing the probability that these events occur. Nodes do not have a waiting queue for incoming jobs, i.e., the workload is measurement is the number of running jobs.
- A3: nodes are homogeneous, they have the same number of servers K and provide the same service at the same speed. Generalization to no homogenous case is easy and left as future work. We will however report simulation results that cover this case.

3.1 $LL(F,T)$: Least Loaded with fan-out F and Threshold T

In this section we describe the Least Loaded among F random nodes with threshold T load balancing protocol, referred to as $LL(F,T)$. The value F is called the protocol *fan-out*. The algorithm works as follows, see Figure 2.

When a node receives a job, if $k < T$, then the node executes the job immediately, otherwise it probes the state of F different random nodes and computes the minimum among the returned values, say m. When $k < K$, if $m \geq k$, then the node executes the job locally; otherwise, it forwards the job to the node reporting m, with ties broken at random. If the receiving node, as well as the probed nodes, have no free servers, i.e., $k, m = K$, the job cannot be served by this fog layer and we count the job as being blocked. A blocked node can trigger different actions, like sending the job to an upper fog layer (e.g., the cloud). We limit to consider this metric as an important performance measure of the algorithm with higher blocking probability being associated eventually to a worse end-user performance.

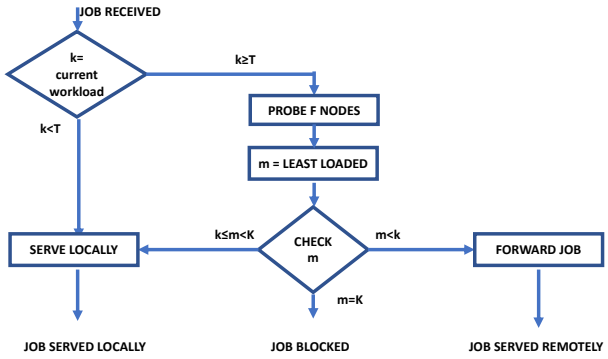


Figure 2: Least Loaded among F probed nodes with Threshold T control flow. K is the total number of servers available, k the number of busy servers at the receiving node. A job executed remotely means that the serving fog node is not the node that received the job.

3.2 Protocol analysis

We study the performance of the $LL(F,T)$ protocol under a Poisson traffic with rate λ jobs per unit of time per node and exponential service time with mean one. Table 1 summarizes the main symbols used in this study.

Number of nodes	N
Number of servers	K
Arrival rate per node	λ
Number of nodes running k jobs	n_k
Number of nodes running at least k jobs	$\widetilde{n}_k = \sum_{i=1}^{k} n_i$
Stationary state probability for the infinity model	π_k
Tail of the stationary state probability	$\widetilde{\pi}_k = \sum_{i=k}^{K} \pi_i$

Table 1: Main symbols used in the analysis.

3.2.1 Protocol analysis for N finite

This model considers a system of N homogeneous fog nodes with K servers. The model is based on a standard Continuous Time Markov Chain (CTMC). The state of the system is expressed through the vector:

$$\mathbf{n} = (n_0, n_1, \ldots, n_K) \quad 0 \leq n_i \leq N, \sum_i n_i = N$$

where n_i is the number of nodes with i running jobs. This direct method can only be applied to small values N, K, as the state space of the model explodes (see the appendix for a quantitative evaluation). However, it is anticipated that the characteristics of the load balancing protocol appear even with small values for N, K. The chain is solved numerically from its canonical formulation:

$$\pi Q = 0, \;\; \mathbf{1}\pi = 1$$

To deal with the memory issues, we've avoided direct matrix inversion and used a power method instead, as it allows for storing the spare matrix efficiently.

Let $\mathbf{e_k}$ be a $K+1$ sized vector of all 0s except 1 in position k. The entry $q(\mathbf{n}, \mathbf{n}')$ of the infinitesimal transition matrix Q is expressed as:

$$q(\mathbf{n}, \mathbf{n}') = \begin{cases} \lambda_i(\mathbf{n}, F) & \mathbf{n}' = \mathbf{n} - \mathbf{e_k} + \mathbf{e_{k+1}} \\ \mu_i(\mathbf{n}) & \mathbf{n}' = \mathbf{n} + \mathbf{e_{k-1}} - \mathbf{e_k}, \\ -\sum_i [\lambda_i(\mathbf{n}, F) + \mu_{i+1}(\mathbf{n})] & \mathbf{n} = \mathbf{n}' \\ 0 & otherwise \end{cases}$$

where $\mu_i(\mathbf{n}) = i \times \mu n_i$ (n_i is the i-th component of $\mathbf{n}$). Let define the function:

$$\delta_{Nn}^{F} = \frac{n!}{(n-F)!} \frac{(N-F)!}{N!}$$

with the convention that $\delta_{Nn}^{F} = 0$ if $F < 0$. This number represents the probability to extract form an urn of N balls containing n balls of the same type, F balls of such a type without replacement.

The birth rate is conveniently divided into the sum of two flows. The first flow:

$$\lambda_{1k}(\mathbf{n}, F) = \lambda n_k \begin{cases} 1 & k < T \\ \delta_{N-1\widetilde{n}_k - 1}^{F} & k \geq T \end{cases}$$

represents jobs that arrive directly to fog nodes. For $k \geq T$ a job is served only if the state of all the F probed nodes is at least k. Let $\widetilde{n}_k$ the number of jobs with at least k jobs running. As the number of these nodes excluding the node itself is $\widetilde{n}_k - 1$, this event occurs with probability given by $\delta(\cdot)$ in the above expression. The second contribution is given by:

$$\lambda_{2k}(\mathbf{n}, F) = \lambda \begin{cases} \widetilde{n}_T(\delta_{N-1\widetilde{n}_k - 1}^{F} - \delta_{N-1\widetilde{n}_{k+1} - 1}^{F}) & k < T \\ \widetilde{n}_{k+1}(\delta_{N-1\widetilde{n}_k - 1}^{F} - \delta_{N-1\widetilde{n}_{k+1} - 1}^{F}) & k \geq T \end{cases}$$

where the term in the parenthesis is the probability that the minimum state of at least one probed node is k.

3.2.2 Performance metrics N finite

The following performance metrics are defined in terms of the stationary probabilities π_k of the CTMC process. As far as the delay is concerned, we approximate its evaluation by assuming that each round trip time is a uniform Random Variable $U = (0, 1]$ and correlations exit among probings.

- Blocking probability, p_b. This value corresponds to the probability of the event that a job received from a node cannot be executed, which occurs when the state of the receiving node, as well as all the F probed nodes, is K. As a job is blocked when the state of all the F different probed nodes is K, we have

$$p_b = \sum_{\pi_{\mathbf{n}}: n_K \geq F+1} \frac{n_K}{N} \cdots \frac{n_K - F}{N - F} \pi_{\mathbf{n}}$$

- Average delay per job, D.
  This delay is the sum of the delay due to probing, job forwarding and result reply and it is given by:

$$D = \Big[\frac{F}{F+1} \frac{1}{N} \sum_{\mathbf{n}: n_k \geq T} n_k \pi_{\mathbf{n}}\Big] + \frac{1}{2} fwd$$

  where fwd is the fraction of received jobs that are forwarded to another node. The first term is the mean of F uniform RV and $\frac{n_k}{N}$ is the probability that a job arrives to a node whose state is k. The fraction of job forwarded is given by:

$$fwd = \sum_{\pi_{\mathbf{n}}: \widetilde{n}_k > 0} \frac{n_k}{N} (1 - \prod_{i=1}^{min\{\widetilde{n}_k, F\}} \frac{\widetilde{n}_k - i}{N - i}) \pi_{\mathbf{n}}$$

  which represents the probability that a job arrival to any of the N nodes, sees the fog node with k servers busy, and the state of all the probed nodes is at least k.
- Overhead per job, ovh.
  As probing is triggered when a job arrives to any of the n_k out of N nodes whose state is $k \geq T$ and the probability such a job arrival event occurs is $\frac{n_k}{N}$, we have:

$$ovh = \frac{F}{N} \sum_{\mathbf{n}: n_k \geq T} n_k \pi_{\mathbf{n}}$$

3.2.3 Protocol analysis N infinite

We now characterize the $LL(F, T)$ protocol in the limit of $N \to \infty$ fog nodes. We make the *conjecture* that when N grows, the dynamics of a set of finite nodes tend to become independent one from each other. While a formal proof of such asymptotic independence is difficult, we remark that this approach has been taken in other works, e.g., [24], [20], [26] and it is the basis of the widely used mean-field theory. We prove that the model arising from this assumption, considered per se has a single solution. If the conjecture is

true, the model then provides correct results. The presented numerical results have similar shapes compared to the model with finite N, thus making this conjecture reasonable.

Suppose then that queues describing our nodes are independent and let's focus on a single tagged node. The state of this tagged node boils down to the number of its busy servers and changes according to a Birth-Death (BD) process with state-dependent birth transition rates, as defined next.

Let π_k the probability of the state being k, and $\widetilde{\pi}_k = \sum_{j=k}^{K} \pi_j$ the tail distribution of the state variable. The BD process describing the state of a node should satisfy the following set of K equations:

$$\lambda_k \pi_k = (k+1)\pi_{k+1} \quad k = 0 \dots K-1 \tag{1}$$

where $\sum_k \pi_k = 1$ and $\lambda_k = \lambda_{1k} + \lambda_{2k}$.

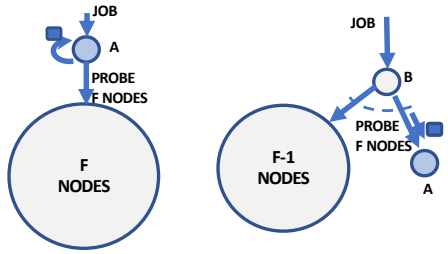


Figure 3: Traffic flows seen by a node A. A job is served by the node when (i) it is received directly from A (left); (ii) it is probed by some other node B, that then forwarded to it.

Our tagged node executes jobs generated directly from its users (left side in Figure 3), when $k < T$, or $T \leq k < K$ and the state of all the F probed nodes is at least k. The birth rate associated to these events is

$$\lambda_{1k} = \lambda \times \begin{cases} 1 & k < T \\ \widetilde{\pi}_k^F & k \geq T \end{cases} \tag{2}$$

Node A may also execute a job of behalf of another node, say B in the right side of Figure 3. This occurs when B selects A and before that B probed other $F-1$ nodes. If the state of A is k, this occurs when the state of B is higher than k, k is also the minimum state state of other i out of the $F-1$ other probed nodes, and B selected A among the $i+1$ least loaded nodes. Since B probes other nodes when the number of busy servers is at least T, these transitions occur with rate

$$\lambda_{2k} = \alpha(k,T) \sum_{i=0}^{F-1} \binom{F-1}{i} \pi_k^i \widetilde{\pi}_{k+1}^{F-i-1} \frac{1}{i+1} \tag{3}$$

where

$$\alpha(k,T) = \lambda \begin{cases} \widetilde{\pi}_T & k < T \\ \widetilde{\pi}_{k+1} & k \geq T, \end{cases}$$

3.2.4 Properties

Theorem 1 (Solution). *Equations Equ. (1) have a single solution.*

Lemma (Upper bound). *For given $F, T, \lambda < K$, let*

$$\hat{\widetilde{\pi}}_k = \frac{\lambda^{(F+1)^{\frac{k-T-1}{F}}}}{\Pi_{i=0}^{k-T}(k-i)^{(F+1)^i}}$$

then if $\lfloor \lambda \rfloor \leq T$ when λ in not an integer and $\lambda < T$ otherwise, the following upper bound holds

$$\widetilde{\pi}_k \leq \hat{\widetilde{\pi}}_k \quad k = T, \dots, K$$

Proof. see appendix. □

Lemma (Equivalence). *$LL(F,1)$ is equivalent to $LL(d)$, where $d = F+1$.*

Proof. See appendix. □

3.2.5 Performance metrics

The counterpart of the performance metrics for this model is defined in terms of the stationary probabilities π_k of the BD process describing the $LL(F,T)$ algorithm.

- Blocking probability, p_B.

  $$p_B = \pi_K^{F+1}$$

  In fact, this value corresponds to the probability of the event that the state of the node receiving a job as well as all the other F probed nodes is K. Note that $p_B \leq \hat{\pi}_K^{F+1}$
- Average delay overhead per job, D.

  $$D = \frac{F}{F+1}\widetilde{\pi}_T + \frac{1}{2} fwd$$

  where fwd is the fraction of received jobs that are forwarded to another node. As a node with state $k \geq T$ forwards a received job if the state of at least a probed node is lower that k, and this occurs $1 - \widetilde{\pi}_k^F$, it is:

  $$fwd = \sum_{T=k}^{K} \pi_k (1 - \widetilde{\pi}_k^F)$$

- Average number of generated probing messages per received job, h. As F messages are generated when the state of the job's receiving node is at least T, this value is hence given by:

  $$ovh = F\widetilde{\pi}_T$$

4 Power of Choice with a Threshold

It is known that the extraordinary efficacy of the power-of-d-choices algorithm is because the state dynamic of a node under $LL(d)$ is radically different from when the node works in isolation. Our $LL(F,T)$ algorithm works in between two extreme points. For $T = 1$ the algorithm is equivalent to $LL(d)$ with $d = F+1$, and hence it completely exploits the power of choices effect, whereas with no cooperation, $T = \infty$, each node works in isolation. It is then worth to better understand how T determines the raising of this change, e.g., how and if it is smooth or not. We provide here some numerical results concerning this aspect.

A formal way to measure how T affects the state distribution is to compute the difference among the schemes as a *distance*:

$$dist(F,T) = \left(\sum_k |\pi_k - \pi'_k|^2 \right)^{\frac{1}{2}}$$

where π_k (π'_k) is the state probability of $LL(F,T)$ and $LL(d)$ with $d = F+1$, respectively.

Table 2 reports the distance $dist(F,T)$ for different values F, T, and $K = 30, \lambda = 25$. The difference falls sharply as T becomes lower than λ. For example a distance of less than 0.005 is reached when $T = \lambda = 25$. Also, the fan-out F makes this change even stronger.

T	30	28	26	25	22
F=1	2.7e-01	1.7e-01	2.4e-02	5.0e-03	2.9e-05
F=2	3.9e-01	3.0e-01	2.1e-02	2.0e-03	1.0e-06
F=4	4.9e-01	3.8e-01	1.4e-02	4.3e-04	1.1e-08

Table 2: Distance $dist(F,T)$ for different fan-out F and thresholds T, $K = 30, \lambda = 25$ (top), and $\lambda = 29$ (bottom).

Though the previous distance provides an objective measurement, a better understanding can be gained by analyzing the whole state probability distribution function. As the highest deviation from an isolated worker is registered with $d = 2$, [37], we focus on this case. Figure 4 shows the π's pdf for $K = 30, \lambda = 25$ and different T. The $LL(1,\infty)$ plot is the pdf of the M/M/K/K queue (no load balancing).

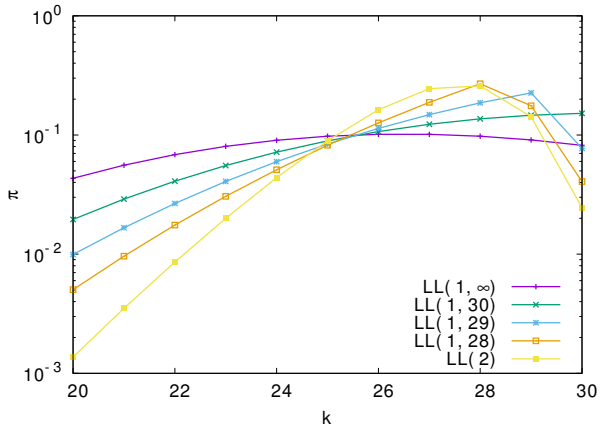


Figure 4: Effect of the threshold on $\pi_k, F = 1$.

From Figure 4, we observe how for $T < K$, the state probability after some k starts to decrease fairly sharply, whereas this is not true for node working in isolation or $T = K$. Figure 5 shows the exact and upper bound of $\widetilde{\pi}_k$, and confirms how the state probability falls sharply with K.

5 Performance Results

This section reports some representative performance results of the $LL(F,T)$ protocol, using the infinite and finite models.

5.1 Infinity model

Figure 6 shows the blocking probability as a function of the load for $K = 50, F = 1$. The way the blocking probability

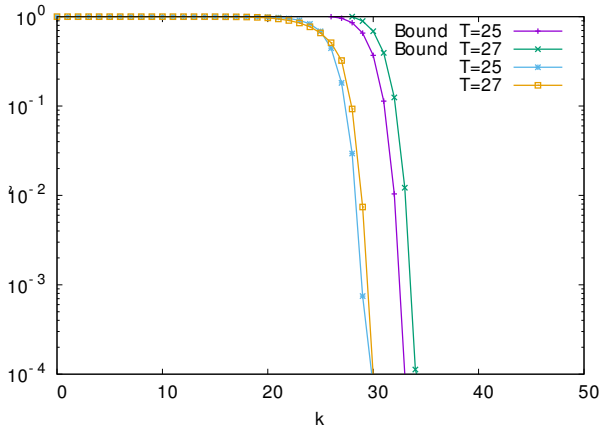


Figure 5: Upper bound and exact $\widetilde{\pi}_k$ for different thresholds $T, K = 50, \lambda = 25$

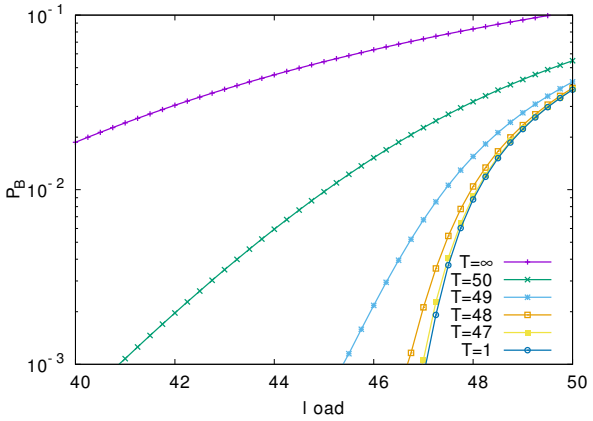


Figure 6: Blocking probability vs load, $F = 1$

changes with the load varies remarkably when load balancing is introduced. The performance can be seen under different angles. For example, suppose that job execution latency is such that they cannot be executed in the cloud[2] and that jobs' blocking probability, called target probability hereafter, should be $p_B \leq 10^{-3}$. Without load balancing, the node can serve up to a traffic load of $\lambda' \approx 33$ requests per unit of time (this value is not shown in the figure), whereas with load balancing with $T = 1$, which is equivalent to a classic Least Loaded policy, $LL(2)$, this traffic raises to $\lambda'' \approx 47$, corresponding to a traffic intensity of $\rho = \frac{\lambda}{K} = 0.94$. To ensure the same p_B the node should increase the number of servers to $K = 67$. We can envision a scenario where the traffic temporarily increases beyond λ' and the node should elastically accommodate this peak. Even assuming a virtualized environment that allows resource scaling, the node can benefit from load balancing during the start-up time of the new 17 resources, which may be not negligible (clearly the traffic of the other nodes should be less than λ'). The plot shows how the same blocking probability can be achieved with $T = 48$.

Another interpretation of this result is the following.

2. Typical values of the round trip time to hit a cloud service can be as high as 100 ms, whereas fog-to-fog latency is likely to be as small as a few ms.

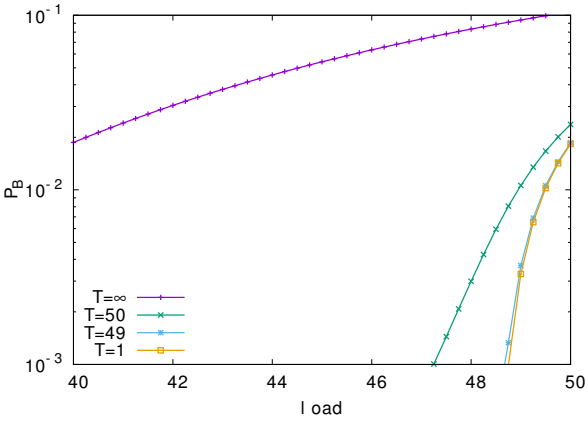


Figure 7: Blocking probability vs load, $F = 3$

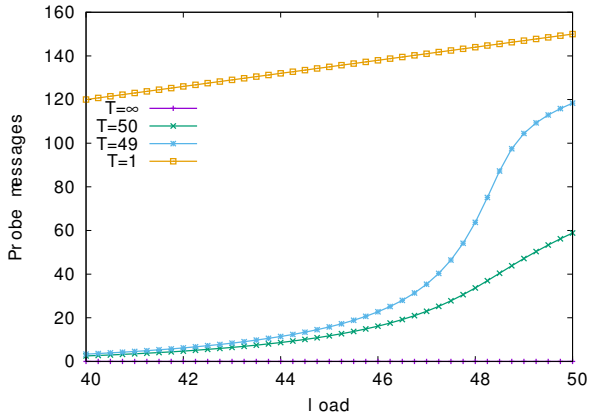


Figure 9: Probe message frequency vs traffic, $F = 3$.

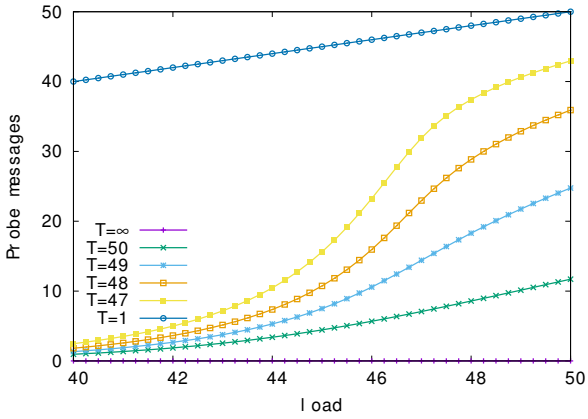


Figure 8: Probe message frequency vs traffic, $F = 1$.

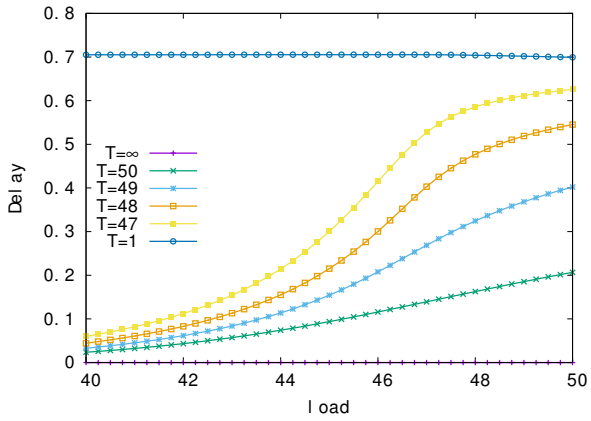


Figure 10: Average control delay vs traffic, $F = 1$

Suppose that jobs are never lost as a node can delegate job execution to the cloud if congested. A node working in isolation can serve traffic at a rate of 47 jobs per unit of time, but a fraction of them, corresponding to the blocking probability, roughly 10 % according to Figure 6, is executed in the cloud, thus expediency a longer latency. If load balancing is used, this fraction is reduced to just 0.1 %. Clearly, additional latency is to be considered that is due to load balancing penalty. By using a threshold, this latency can however reduced, as discussed next.

Figure 7 shows the blocking probability when the fan-out is increased to $F = 3$. Now it's enough to set $T = 49$ to get the same fraction of blocked jobs.

Figure 8 shows the frequency of probing messages generated as a function of the traffic and different T. The advantage of using a threshold is measured by the reduction of probe messages. For $T = 48$ and $\lambda = 47$, the messages rate decreases of a half with respect to an uncontrolled load balancing activity, and much more for lower traffic. For example, for $\lambda = 40$, the target p_B is reached with $T = 50$ at the cost of just a few probe messages. At this traffic, the node should still increase its servers (to $K = 59$) if working in isolation. Besides the obvious benefit of decreasing the control overhead, this reduction also reduces the risk of race conditions that may weaken the effectiveness of load balancing, as remarked in [17], [18].

Figure 9 shows the frequency of probing messages for $F = 3$. Under the overhead point of view, the best fan-out is a matter of the working conditions. For example, if the load is less than $\lambda = 46$ and the target loss fraction is 10^{-3}, then $F = 1$ or $F = 3$ generate almost the same amount of control messages of less than 20 messages per unit of time, i.e. less than a half if threshold is not used. If the load is increased, $F = 3$ may be required to meet the target blocking probability.

Figure 10 shows the average control delay penalty introduced by the protocol as a function of the traffic. This delay also reduces remarkably using a threshold. For example, from 0.7 to 0.4 for $\lambda = 47$ and to 0.05 for $\lambda = 40$, i.e. of one order of magnitude. Note that for $T = 1$ the average delay is not one, because a job is not always forwarded (the delay is the sum of the average probing message, i.e., 0.5, plus the delay of job forwarding and reply, that occurs only when a job is forwarded.

The delay for $F = 3$ is shown in Figure 11. Even under high load $\lambda = 50$ corresponding to $\rho = 1$, using a threshold reduces the control delay. This result corroborates the claim that to achieve the benefit of randomization it not required to always probe other nodes.

5.2 Finite model

This section reports the main three metrics, obtained with $N = 8, K = 14$. Figure 12 shows the blocking probability

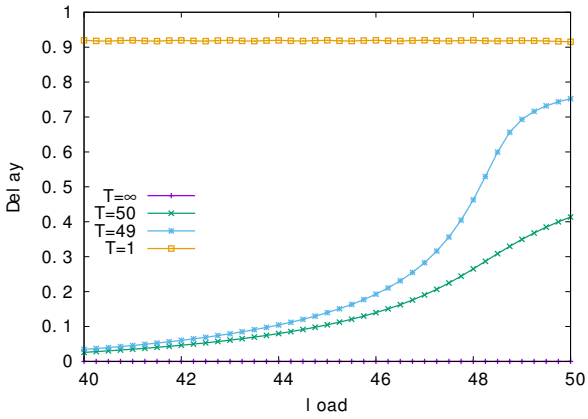


Figure 11: Average control delay vs traffic, $F = 3$

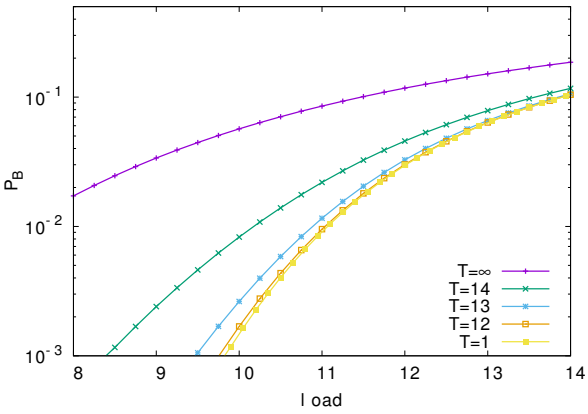


Figure 12: Blocking probability vs traffic, $F = 1$.

as a function of the traffic for $F = 1$. Load balancing allows reducing the blocking probability considerably allowing to serve a traffic of up to approximately 10 requests per unit of time for our hypothetical target p_B, corresponding to a traffic intensity of 0.7. The same value is reached with $T = 12$.

The effect of increasing the fanout is visible in Figure 13. Clearly, the margin for improvement is now limited by the finite number of nodes. We can also see see that by setting $T = K - 2$, the protocol reaches the same performance of $LL(d)$.

Figure 14 and Figure 16.(a) show the benefit of threshold on the control overhead and delay. As for the infinity model, this advantage is even higher for lower traffic.

Figure 15 and Figure 16.(b) shows the same performance index when $F = 2$. Similarly to the results obtained from the infinity model, increasing F allows to further reduce the blocking probability, though the improvement is limited by the finite number of nodes.

6 Simulation

In this section, we report the results of a simulated model that considers additional details missed in the mathematical study, concerning the effect of delay and heterogeneity among fog nodes.

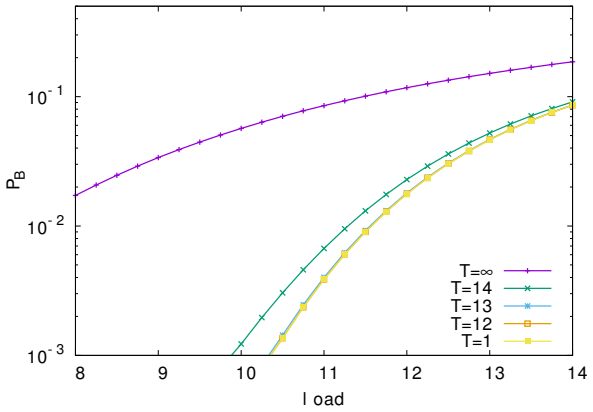


Figure 13: Blocking probability vs traffic, $F = 2$.

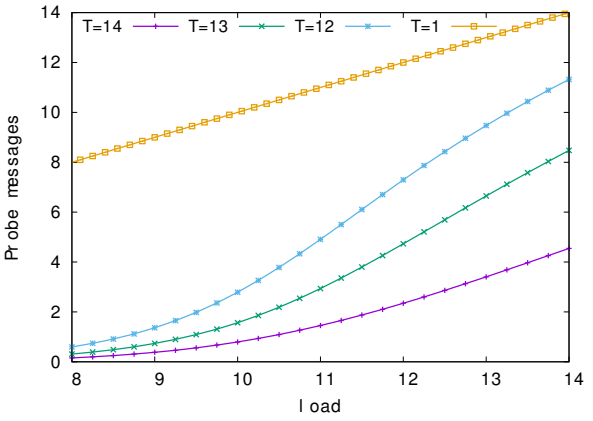


Figure 14: Probe message frequency vs traffic, $F = 1$.

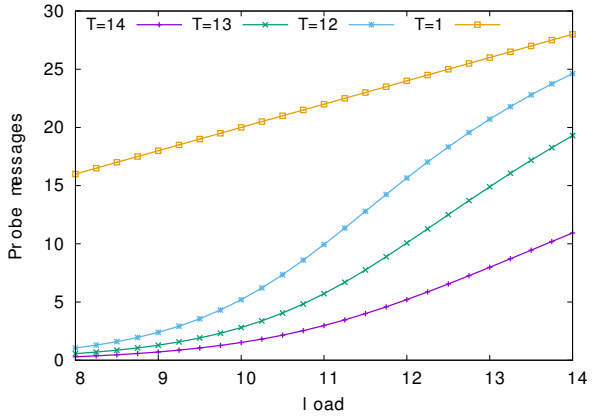
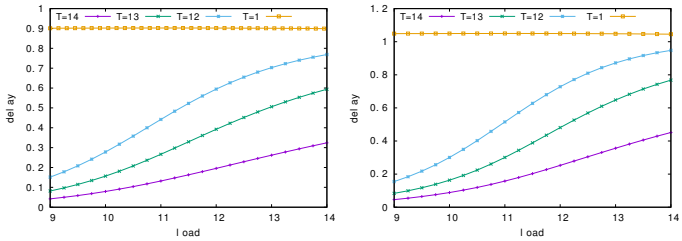


Figure 15: Probe message frequency vs traffic, $F = 2$.



(a) $F = 1$ (b) $F = 2$

Figure 16: Average control delay vs traffic, finite model

We've used a custom discrete-event simulator, written in Python. The system under observation is composed of N_1 fast fog nodes and N_2 slower nodes that provide the same service to a set of end users in a given restricted area. Globally, the traffic generated by users covered by the same fog node is a Poisson process, whereas the average execution of the provided service requires s_1 ms by fast nodes and s_2 by slow nodes[3]. The Poisson assumption can capture a realistic scenario of moving end users entering and leaving areas covered by fog nodes and requesting an application service, i.e., object recognition for VR/AR applications, as described for example in [12]. The parameters used for simulations are reported in the following Table:

Total Number of Nodes	32
K	14
Traffic rate λ [jobs/s]	35,105,210
Job duration, $\frac{1}{\mu}$ [ms]	300,100,50
Traffic intensity per node $\rho = \frac{\lambda}{K\mu}$	0.75
Task duration on a fast server [ms]	90%
Job Length [MB]	1
Device-to-Fog Delay [ms]	Uniform [5,5.5]
Fog-to-Fog Delay [ms]	Uniform [5,10]
Device-to-Fog Bandwidth [Mbps]	100
Fog-to-Fog Bandwidth [Mbps]	54

Each simulation lasts until at least 3000 loss events are detected. The plots report the average among 5 independent repetitions of a same simulation.

We have compared our protocol with a centralized round-robin load balancing algorithm, where all fog nodes first send their jobs to the centralized scheduler (assumed with infinity capacity), then that applies the Round-Robin rule (RR).

Figure 17 shows the blocking probability as a function of the threshold (left) and the delay for the same traffic intensity $\rho = 0.75$ and different execution times of a task. When the service time of a task is much higher than the latency of control messages, 300 ms corresponding to control delay of about 3% of the service time, the blocking probability follows what has been predicted by the model: as the threshold decreases, it falls sharply to its minimum value and remain unchanged. However, as the execution time becomes comparable with the control delay, after the minimum, the blocking probability increases again. This is especially evident for service time 50 ms that corresponds to control delay 15%. The reason is that exactly because of the control message transmission delay, the state of a remote node at probing time can differ from its state when a job is actually received. It may then happen than the workload of a received fog node may be higher than the workload of the sending node, hence weaken the effectiveness of the load balancing mechanism. Also, we have noticed from inspecting simulation traces that in some cases when the state of a node is close to K a job is dropped by the probed node because differently from what it has reported the node has no longer idle servers. The left side of Figure 17 shows that the blocking probability of the RR balancer is higher than our protocol. With the optimal threshold our protocol drops about 0.25% of message, whereas under RR about 2.5%, i.e. one order of magnitude higher. The blocking probability of RR did not changed with the service time (recall that the traffic intensity is kept fixed in the experiment to 0.75).

Figure 17 also shows the delay (on the right), measured as the time from when a job is generate by an IoT device until the device gets the reply. This value was found slightly higher for low thresholds, which is due to the addition job transfer time and probing overheads. The delay of the RR protocol is always higher than LL, hence we can obtain a net advantage since more jobs are served without any delay penalty.

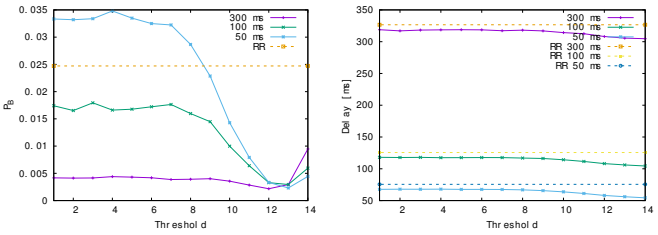


Figure 17: Impact of the execution time on the blocking probability and delay.

The second set of experiments measured the effect of server heterogeneity. In this experiment half of the nodes execute a job in 300ms (slow node) and the other half in 250 ms (fast node), i.e. half of the nodes are approximately 20% faster. Users are connected to a slow or fast node. The load of a node is $\lambda = 35$ req/s. Figure 18 reports the blocking probability as a function of the threshold seen by users that send their job requests to a fast or slow node. The figure also shows the total job's response time, i.e. the time elapsed from when a device sends a job execution request until it gets the reply. We can see that the effect of the threshold is still effective in case of server heterogeneity, namely a threshold of $T = 12$ provides similar results of the homogeneous case. Since the load balancing allocates jobs to random servers, the response time for jobs coming from users connected to a fast node is higher than 250 ms, i.e., the execution time when executed on fast nodes, time because it may occur that a job is executed on a slower server. The advantage lays in the higher number of served jobs. For the same reason, the response time seen by users connected to a slow node becomes lower than when jobs are not forwarded to any faster node.

3. Due to the insensitiveness to the service distribution of loss models, only the average matters.

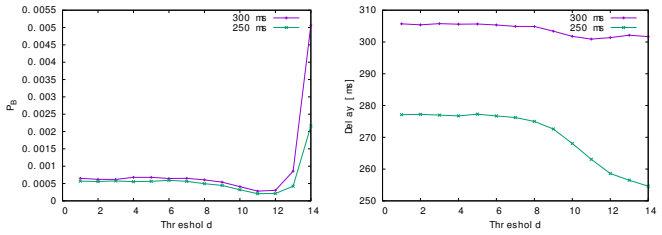


Figure 18: Performance for the heterogenous case.

7 Implementation

An experimental test of the proposed scheduling algorithm has been conducted on a framework, dubbed *P2PFaaS*, that we conceived and designed according to our findings described in the previous sections. Each fog node runs an instance of such a framework, see Figure 20. The framework relies on the Function-as-a-Service model (FaaS), [38] according to which the service provided by a fog node is exposed as a stateless function. The exported functionality is an image detection service that is provided by all the fog nodes, [4]. Function invocation takes an image as input returns the coordinate of a rectangle containing a face in the passed image. The invocation of such a function bounded to a specific image and corresponds to the unit of execution, called job or task hereafter. Job transfer corresponds to transfer the image to recognize to another fog node.

The framework, that is completely written in *Go*, is composed by a discovery service, which allows nodes to know each other, and a scheduler service in which the core of the scheduling policy resides. In particular, the scheduler service can forward image detection requests to other nodes, do probing or schedule the function execution to the current node. The framework implements only the scheduling logic since the actual function execution is delegated to OpenFaaS [40]. Both OpenFaaS and our framework rely on Docker and Docker Swarm, for which every node represents a swarm/cluster with only one node. This one to one mapping is done in order to avoid to use the Docker Swarm pre-built scheduler, which always assigns a new job to the least loaded node among the cluster it manages. To avoid conflicting decisions, OpenFaaS auto-scaling is disabled and the maximum number of concurrent functions is set to $K = 10$.

A client that needs to perform a face detection task, sends an HTTP request to a fog node. When the node receives the request, the scheduling policy is applied: the current number of running functions is checked and if it is below the threshold T the request is immediately delegated to OpenFaaS, which executes the face detection function. In the other case, when the current load is equal to or above the threshold T, the scheduler service picks F node IDs at random and probes their load via parallel HTTP requests. When all the replies have been collected, the scheduler decides where to schedule the request, and if it is forwarded to another node, it performs another HTTP request. When this node completes the execution the result is returned via HTTP response to the origin node which returns it to the client. Again the client waits for the job completion and it executes only one HTTP request, all what happens behind it's totally transparent to the client.

4. The function that implements this face detection is the Pigo Face Detector [39] function and implements the Pixel Intensity Comparison-based Object detection that is a modification of the standard Viola-Jones method

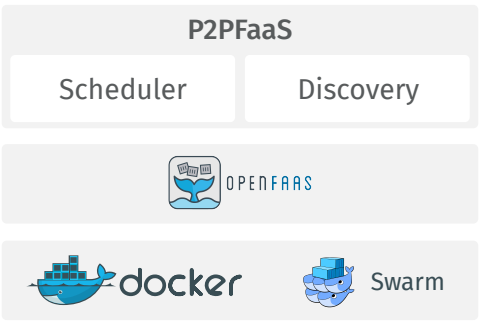


Figure 19: P2PFaaS concept diagram, illustrating the complete stack of services

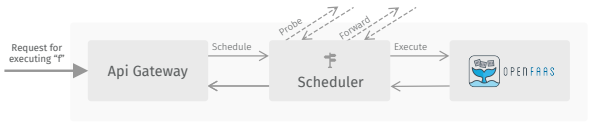


Figure 20: Scheme of execution of a function in the P2PFaaS framework

An important optimization that has been introduced since the initial concept version of the framework, regards the probing. Indeed, after some tests, it resulted that serializing a JSON for replying to a probe, with node load information, is too demanding to be performed since it requires a considerable amount of CPU time. For circumventing this problem, load information is now passed via HTTP headers and this allowed to drop the average probing time from 40ms to 10ms.

7.1 Results

We have conducted several tests by exploiting a cluster of two servers equipped with Intel Xeon @ 2.80 GHz, which are used to instantiate 8 VMs with assigned 1 core, 3GB of RAM and with Debian installed. Every machine has been equipped with Docker, OpenFaaS and our framework with $K = 10, F = 1$, and they have been set up as master nodes of single-node Docker Swarms. Then a ninth VM, the "benchmarker", has been instantiated in order to generate the traffic of the requests and collect all the data. All VMs are connected via fast ethernet within the same local network. A series of Python scripts generates parallel traffic of image recognition requests to every machine, then they collect the number of dropped requests, the average execution time of a function and a series of other parameters that regard probing times, forwarding times, number of http errors and many others. The average execution time of a single image recognition is 300ms. Each experiment consisted of sending 20.000 detection requests at rate $\lambda = 3.00$, thus having $\rho = 0.9$.

The experiment has been repeated 7 times, due to its duration ($\approx$ 24hrs), and in the following figures, results are shown by using a confidence interval with $\alpha = 0.1$ and with sample mean error of $\pm\, t_{\frac{\alpha}{2},n-1}\sqrt{\frac{S^2}{n}}$ (where S^2 is the sample variance).

Figure 21 shows the estimated blocking probability (ratio of dropped requests to the total number of requests generated). This experiment shows a minimum for $T = 8$. As predicted by the theoretical model (see Figure 12) the blocking probability drops sharply as T decreases; however, rather than remaining almost constant at that value it starts to increase when T is further reduced. The reason is that when T is lowered, the workload due to job scheduling at each fog node increases since the number of probe per job increases. In the limit of $T = 1$, unless the fog node is idle *every* job arrival triggers a probe-reply cycle. While the length of such control messages is overall negligible, their processing is not and it has the net effect of reducing the CPU cycles allocated to the image detection or, equivalently, to increase the duration job execution. This aspect is not captured by the model. The reduced CPU execution speed clearly increases the average execution time of served job, as reported in Figure 22.

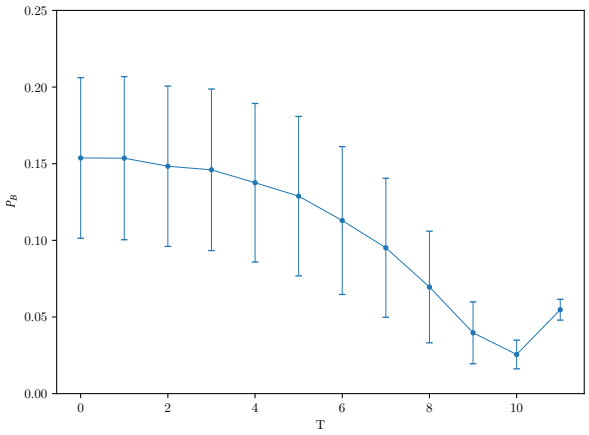


Figure 21: Blocking probability as a function of the threshold.

Finally Figure 23 reports the output of the RRDTool performance profiler used during the trials, showing the CPU breakdown, total memory usage and control traffic of a fog node. Each pause in the trace corresponds to decreasing the threshold of one unit, starting from $T = 10$. We can see how the CPU usage slightly increases as T decreases and is approximately 0.9, which is consistent with the nominal generated traffic intensity, $\frac{\lambda}{\mu}$. Such an increase is due to an increase in the control message processing, as outlined above. Neither the memory nor the network is saturated, although they both increase as the protocol becomes more proactive.

8 CONCLUSION

This paper studied the performance of the Least Loaded among d nodes, $LL(d)$ when adapted to a fog computing deploy. This adaptation consists of triggering the randomized search only when the workload of the current fog node

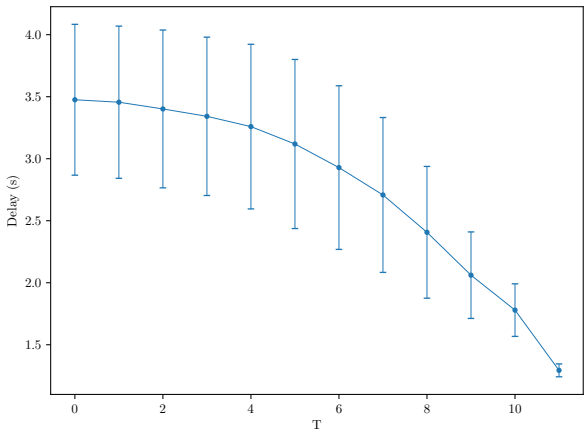


Figure 22: Average end to end delay as a function of the threshold.

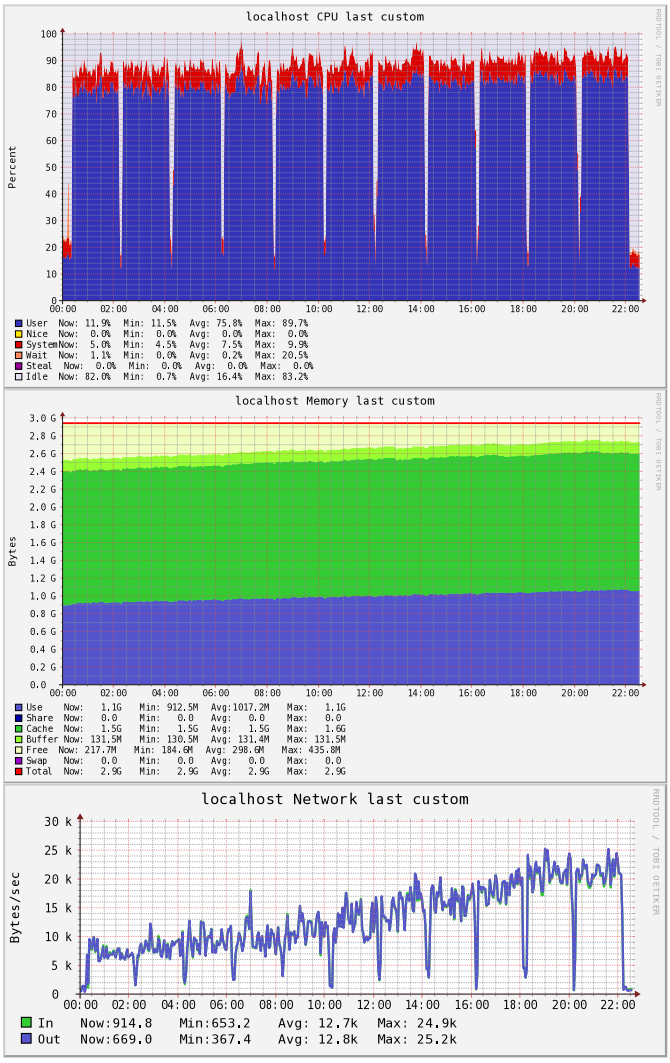


Figure 23: Performance profile during a test. The threshold is varied approximately every 2 hours

that receives a new job to execute is above a threshold value, T. The threshold is shown to be a simple way to reduce control delays without affecting the very nature of

the power-of-random choices principle.

Through a mathematical analysis we show that under Poisson arrivals and an exponential distributed service time, setting $T = K - 2$, where K is the number of servers of a node, achieves practically the same performance of the power-of-choice's classical implementation requiring a single global scheduler, but at much lower delay penalty and control overhead of up to one order of magnitude less. Simulation experiments and a real implementation corroborated our finding.

REFERENCES

[1] A. Aijaz, M. Dohler, A. H. Aghvami, V. Friderikos, and M. Frodigh, "Realizing the tactile internet: Haptic communications over next generation 5g cellular networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 82–89, April 2017.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, 2012, pp. 13–16.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[4] A. W. Richa, M. Mitzenmacher, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," *Combinatorial Optimization*, vol. 9, pp. 255–304, 2001.

[5] H. El-Sayed, S. Sankar, M. Prasad, D. Puthal, A. Gupta, M. Mohanty, and C.-T. Lin, "Edge of things: The big picture on the integration of edge, iot and the cloud in a distributed computing environment," *IEEE Access*, vol. 6, pp. 1706–1717, 2017.

[6] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE access*, vol. 6, pp. 47 980–48 009, 2018.

[7] S. N. Buyya, Rajkumar et al. B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. S. Netto, A. N. Toosi, M. A. Rodriguez, I. M. Llorente, S. D. C. D. Vimercati, P. Samarati, D. Milojicic, C. Varela, R. Bahsoon, M. D. D. Assuncao, O. Rana, W. Zhou, H. Jin, W. Gentzsch, A. Y. Zomaya, and H. Shen, "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 105:1–105:38, Nov. 2018. [Online]. Available: http://doi.acm.org/10.1145/3241737

[8] OpenFog Consortium Architecture Working Group, "Openfog reference architecture for fog computing," https://iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf, Tech. Rep. OPFRA001.020817, Feb 2017.

[9] Z. Li, M. L. Sichitiu, and X. Qiu, "Fog radio access network: A new wireless backhaul architecture for small cell networks," *IEEE Access*, vol. 7, pp. 14 150–14 161, 2019.

[10] P. Marsch, I. Da Silva, O. Bulakci, M. Tesanovic, S. E. El Ayoubi, T. Rosowski, A. Kaloxylos, and M. Boldi, "5g radio access network architecture: Design guidelines and key considerations," *IEEE Communications Magazine*, vol. 54, no. 11, pp. 24–32, November 2016.

[11] ETSI Industry Specification Group (ISG), "Mobile edge computing (mec); framework and reference architecture," https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf, ETSI, Standard ETSI GS MEC 003 V1.1.1, mar 2016.

[12] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1270–1278.

[13] Y. Xiao and M. Krunz, "Distributed optimization for energy-efficient fog computing in the tactile internet," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 11, pp. 2390–2400, Nov 2018.

[14] S. M. A. Oteafy and H. S. Hassanein, "Leveraging tactile internet cognizance and operation via iot and edge technologies," *Proceedings of the IEEE*, pp. 1–12, 2018.

[15] K. Velasquez, D. P. Abreu, M. R. Assis, C. Senna, D. F. Aranha, L. F. Bittencourt, N. Laranjeiro, M. Curado, M. Vieira, E. Monteiro *et al.*, "Fog orchestration for the internet of everything: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 14, 2018.

[16] A. N. M. Roberto Beraldi, Abderrahmen Mtibaa, "Cico: A credit-based incentive mechanism for cooperative fog computing paradigms," in *Globecom 2018*. IEEE, 2018.

[17] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. ACM, 2013, pp. 69–84.

[18] R. Beraldi and H. Alnuweiri, "Sequential randomization load balancing for fog computing," in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Sept 2018.

[19] M. Mitzenmacher, "How useful is old information?" *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 1, pp. 6–20, Jan 2000.

[20] M. Bramson, Y. Lu, and B. Prabhakar, "Asymptotic independence of queues under randomized load balancing," *Queueing Syst. Theory Appl.*, vol. 71, no. 3, pp. 247–292, Jul. 2012.

[21] F. Garcia-Carballeira and A. Calderón, "Reducing randomization in the power of two choices load balancing algorithm," in *2017 International Conference on High Performance Computing and Simulation (HPCS)*, 07 2017, pp. 365–372.

[22] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Mean-field analysis of loss models with mixed-erlang distributions under power-of-d routing," in *29th International Teletraffic Congress (ITC 29)*, Sept 2017.

[23] T. Hellemans and B. Van Houdt, "On the power-of-d-choices with least loaded server selection," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 2, Jun. 2018. [Online]. Available: https://doi.org/10.1145/3224422

[24] Y. L. Qiaomin Xie, Xiaobo Dong and R. Srikant, "Power of d choices for large-scale bin packing: A loss model," in *Proceedings of ACM SIGMETRICS 2015*. IEEE, 2015.

[25] S. R. E. Turner, "Resource pooling in stochastic networks," 1997. [Online]. Available: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.627069

[26] A. Mukhopadhyay, R. R. Mazumdar, and F. Guillemin, "The power of randomized routing in heterogeneous loss systems," in *2015 27th International Teletraffic Congress*, 2015, pp. 125–133.

[27] C. Fricker, F. Guillemin, P. Robert, and G. Thompson, "Analysis of an offloading scheme for data centers in the framework of fog computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 1, no. 4, pp. 16:1–16:18, Sep. 2016.

[28] R. Beraldi, H. Alnuweiri, and A. Mtibaa, "A power-of-two choices based algorithm for fog computing," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2018.

[29] S. Fu, C.-Z. Xu, and H. Shen, "Random choices for churn resilient load balancing in peer-to-peer networks," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, April 2008, pp. 1–12.

[30] E. C. P. N. G. C. F. Aires, "An algorithm to optimise the load distribution of fog environments," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2017.

[31] A. Kapsalis, P. Kasnesis, I. S. Venieris, D. I. Kaklamani, and C. Z. Patrikakis, "A cooperative fog approach for effective workload balancing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 36–45, March 2017.

[32] Y. Song, S. S. Yau, R. Yu, X. Zhang, and G. Xue, "An approach to qos-based task distribution in edge computing networks for iot applications," in *2017 IEEE international conference on edge computing (EDGE)*. IEEE, 2017, pp. 32–39.

[33] S. K. Mishra, M. A. Khan, B. Sahoo, D. Puthal, M. S. Obaidat, and K.-F. Hsiao, "Time efficient dynamic threshold-based load balancing technique for cloud computing," in *2017 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 2017, pp. 161–165.

[34] D. Puthal, M. S. Obaidat, P. Nanda, M. Prasad, S. P. Mohanty, and A. Y. Zomaya, "Secure and sustainable load balancing of edge data centers in fog computing," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 60–65, 2018.

[35] D. Puthal, R. Ranjan, A. Nanda, P. Nanda, P. P. Jayaraman, and A. Y. Zomaya, "Secure authentication and load balancing of

distributed edge datacenters," *Journal of Parallel and Distributed Computing*, vol. 124, pp. 60–69, 2019.
[36] S. K. Mishra, D. Puthal, B. Sahoo, S. Sharma, Z. Xue, and A. Y. Zomaya, "Energy-efficient deployment of edge dataenters for mobile clouds in sustainable iot," *IEEE Access*, vol. 6, pp. 56 587–56 597, 2018.
[37] T. Pering, Y. Agarwal, R. Gupta, and R. Want, "Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces," in *Proceedings of the 4th international conference on Mobile systems, applications and services*. ACM, 2006, pp. 220–232.
[38] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "Serverless programming (function as a service)," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2658–2659.
[39] E. Simon. (2020, Jan) esimov/pigo. [Online]. Available: https://github.com/esimov/pigo/
[40] OpenFaaS. (2020, Jan) Openfaas - serverless functions made simple. [Online]. Available: https://www.openfaas.com/

Roberto Beraldi He is an associate professor at DIAG, "La Sapienza" University of Rome, Italy. His current research interests include mobile networking, fog computing, and distributed systems. Regularly serves as TPC member of international conferences and journals in these fields.

Gabriele Proietti Mattia received the BS and MS degrees in Engineering in Computer Science from Sapienza University of Rome, Italy, in 2017 and 2019, respectively. He is currently a PhD student and his research interests include fog computing and scheduling algorithms.