A random walk based load balancing algorithm for Fog Computing

Roberto Beraldi*, Claudia Canali†, Riccardo Lancellotti†, Gabriele Proietti Mattia*
*Department of Computer, Control and Management Engineering "Antonio Ruberti",
University of Rome "La Sapienza",
Email: beraldi@diag.uniroma1.it, proiettimattia@diag.uniroma1.it
†Department of Engineering "Enzo Ferrari",
University of Modena and Reggio Emilia,
Email: claudia.canali@unimore.it, riccardo.lancellotti@unimore.it

***Abstract*—The growth of large scale sensing applications (as in the case of smart cities applications) is a main driver of the fog computing paradigm. However, as the load for such fog infrastructures increases, there is a growing need for coordination mechanisms that can provide load balancing. The problem is exacerbated by local overload that may occur due to an uneven distribution of processing tasks (jobs) over the infrastructure, which is typical real application such as smart cities, where the sensor deployment is irregular and the workload intensity can fluctuate due to rush hours and users behavior. In this paper we introduce two load sharing mechanisms that aim to offload jobs towards the neighboring nodes. We evaluate the performance of such algorithms in a realistic environment that is based on a real application for monitoring in a smart city. Our experiments demonstrate that even a simple load balancing scheme is effective in addressing local hot spots that would arise in a non-collaborative fog infrastructure.**

I. Introduction

Fog computing promises to change the architecture of software applications from the current cloud-only support to a multi-layer system, where computational facilities are available at each level along the path from data sources to a centralised cloud data centre, [1]. This deployment model, which materializes with an intermediate computational layer called Fog, will address demanding constraints on response time (order of 10 ms), throughput (order of 10 Gbps), as well as high security and privacy [2].

In this paper, see Fig. 1, we consider a model where fog nodes are densely distributed in a given geographic area and provide a service to end devices, e.g., IoT. For instance, these nodes are integrated into the radio access networks of the 5G (F-RAN) architecture, [3] and support the implementation of a Smart City, as detailed in the IETF Internet-Draft [4] or an object detection vision service as described in [5]. The cloud layer, instead, is used as a support to the fog one, indeed, in our perspective when requests cannot be executed in the fog they are supposed to be forwarded to the cloud.

From the research point of view, fog computing introduces several challenges (see [6] for a general discussion). A research topic currently under investigation is the design of an algorithm for resource sharing among uncoordinated and heterogeneous fog nodes. Although resource sharing is a classical and well-studied topic in the computer science community, this

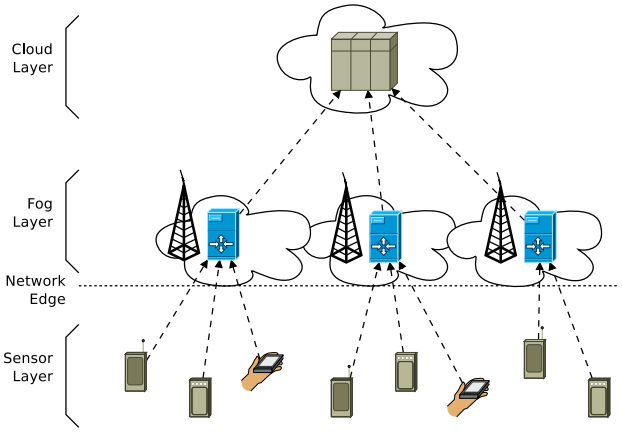

Fig. 1. A typical fog computing deploy model.

model of fog computing does not fit all the assumptions of the studies available in the literature. In particular, the following elements are new to the fog deployment: (i) the execution time of a job is comparable to the time required to transfer the job from the node of origin to another node; (ii) the absence of a centralized entity that acts as a load balancer; (iii) the heterogeneity among resource availability and local scheduling policies.

This work proposes two load balancing algorithms, dubbed *sequential forwarding* and *adaptive forwarding* designed to take these peculiarities into account. The algorithms allocate jobs that are continuously generated from end devices (on-line load balancing). The basic idea of the proposed algorithms is the following. We assume that the fog computing layer provides an elementary service to end-users, e.g., consisting in object detection inside a video frame [5]. When a new job arrives at a fog node, the node estimates the current waiting time of the job, i.e., the time the job is supposed to wait before its execution ends. If this time is considered too long (based on a threshold value that may be self-tuning), the fog node forwards *blindly* the job to another nearby fog node. This node, in turn, follows a similar decision process. The algorithm is repeated up to a certain number of trials or steps M, after which the job is no longer moved, i.e., the last node has to

serve the job or drop it if congested. A different way to look at the cooperation mechanism supported by these algorithms, is as a random walk routing process with maximum lifetime M steps, searching for a suitable node that can serve the job. The algorithm fits the three challenges of fog computing because it places a significant effort in limiting the number of (potentially expensive) hops between nodes, it is completely distributed and it does not rely on the assumption of a homogeneous scenario, taking explicitly into account the absence of uneven load distributions. Since the waiting time at a node depends on its processing rate or other local policies, for example based on a priority, the algorithm, thanks to its simple formulation, can be easily adapted to deal with heterogeneous resources and local scheduling policies. In addition, we believe that these features and the simplicity of the forwarding decisions, e.g., the lack of probing or resource reservation mechanisms [7], make the algorithm suitable for the fog model and allows to address the underlying multi-objective optimization problem of maximizing the number of served jobs *and* minimizing their waiting time in a simple and feasible way. The contribution of the paper can be summarized as follows:

- definition of a lightweight randomized on-line distributed load balancing algorithm characterized by *independent* providers and *heterogeneous* load conditions, along with a variant based on a *self-tuning* mechanism
- mathematical analysis and experimental evaluation of the algorithms on a realistic scenario, showing evidence of significant improvements compared to unbalanced nodes, e.g., loss rate reduced from 25% to 5% and up 0% for the adaptive case, shorter response time of about 40%.

The rest of this paper is organized as follows. Section II describes the proposed algorithms while Section III describes a mathematical model of the protocol. In Section IV, we provide both numerical results from the model and a simulation study where the proposed algorithms are applied to a realistic scenario of a smart city application deployment. Section V discusses related works while conclusions and future research directions are provided in Section VI.

II. Sequential forwarding algorithms

In this section, we describe the proposed distributed load balancing algorithms. The intuition behind our proposal is to obtain resource sharing by fairly forwarding jobs to randomly sampled nodes until the node receiving a job guesses that a less loaded node exists in the network.

We recall that our reference architecture is described in Fig. 1, with a set of sensors sending jobs to a set of nodes. Each sensor is assigned to one fog node, selected, for example, based on geographic distance as in [8]. The workload of fog nodes can be highly heterogeneous, due to fluctuations in the workload patterns or simply because the sensors are not evenly distributed among the fog nodes [9]. Furthermore, we assume that the fog nodes are able to interact among themselves by forwarding jobs from one node to another[1]. In this section, we present two algorithms aiming to define *when* a Job should be forwarded to a neighbor, and *to which* neighbor the job should be forwarded.

In particular, we start by describing the *Sequential Forwarding* algorithm; next, we describe an evolution of this algorithm *Adaptive Forwarding* algorithm. It is worth noting that these algorithms operate blindly with respect to the state of the other nodes, so the choice of the neighbor that will receive the forwarded job is random. Finally, we discuss a baseline algorithm, namely *No LB*, that is the case where no load balancing occurs among the fog nodes.

A. Sequential Forwarding algorithm

The *Sequential Forwarding* algorithm uses a threshold Θ to determine whether an incoming job is to be forwarded to a random neighbor or not. The threshold is applied to the system load, which is the number of jobs queued in the fog node (or being executed). This metric is used as an estimation of the waiting time for the incoming job. Furthermore, the algorithm defines a maximum number of steps M to guarantee a bound on the delay experienced by each job during the load balancing phase.

Algorithm 1 Sequential Forwarding Algorithm

```
Require: M, Θ, Job
  if Job.Steps ≥ M then
    ProcessLocally()
  else
    if System.Load() ≤ Θ then
      ProcessLocally()
    else
      Neigh ← Random(System.Neighbors())
      Job.Steps++
      Forward(Job, Neigh)
    end if
  end if
```

Algorithm 1 provides an overview of the proposed load balancing approach. When a job arrives, if the job has already been forwarded M times, we schedule the job for local processing. The local processing is carried out with a server with a finite queue, where we denote the maximum queue length as Q. The queue in the processing node is responsible for the drop of the job is the queue is full. As our focus is on the cooperation algorithm, we consider the queue length as a constant value of limited interest for our analysis. If we haven't reached the M-th step we consider the number of jobs already scheduled for processing in the fog node (that is the system load). If the value does not exceed the threshold Θ, the job is accepted and scheduled for local processing. Otherwise, we blindly select a random neighbor among the fog nodes and we forward the job to that neighbor. We stress the *blind* and *memoryless* nature of the algorithm so that we do not need to probe the status of the neighbor nor there is the need to adapt any kind of resource reservation (to make sure that the job can find a resource available after probing [7]). This makes

[1]For example, the X2 interface allows direct communications among 5G eNB nodes.

the algorithm extremely simple to implement and facilitates its adoption among different providers.

B. Adaptive Sequential Forwarding algorithm

The Sequential Forwarding proposed in Sec. II-A has two separate parameters, Θ and M, that show an inherent interdependence. Indeed, if Θ is low we are likely to experience a high number of forwarding, thus making M crucial for the algorithm performance. This complex parameters tuning, that can be even worsened in the case of heterogeneous environments, leads us to conceive an adaptable version of the algorithm.

The Adaptive Sequential Forwarding algorithm (in the following simply called *Adaptive Forwarding*), is an evolution of the Sequential Forwarding proposed in Sec. II-A aiming to provide some self-tuning ability.

Algorithm 2 Adaptive Forwarding Algorithm

Require: M, Job
  $Q \leftarrow$ System.QueueLen()
  $\Theta \leftarrow \lceil$Job.Steps * $Q/M \rceil$
  SequentialForwarding(M, Θ, Job)

Algorithm 2 illustrates the behavior of the Adaptive Forwarding Algorithm. The threshold Θ is computed in such a way that it grows linearly with the number of steps. If a job has never been forwarded (or has been forwarded just a few times), the job is not processed locally unless the local load is very low. On the other hand, if a job has already been forwarded several times we assume a more relaxed attitude towards the search of a fog node with a low load.

The first lines of the algorithm compute the threshold Θ that grows linearly with the number of times the job has been forwarded. In particular, we tune the growth of the threshold in such a way that, after M steps, the threshold Θ is equal to the maximum queue length of a fog node, thus guaranteeing that the job will be accepted unless this last visited node has no room in its queue.

C. Baseline algorithm

In the performance evaluation, we consider also the *No LB* algorithm, that is the case where no load balancing occurs, as a baseline.

For the algorithms previously described, and given Q as the maximum length of queue (as in Algorithm 2), the behavior corresponds to the case where $\Theta > Q$ or to the case where $M < 1$. This algorithm is expected to be characterized by a high loss rate (that is jobs dropped because the queue is full), unbalanced load (especially in scenarios with heterogeneous load distribution among the fog nodes) and, generally by poor performance.

III. Model

Having described the algorithms overview, we now propose a mathematical model to describe the system performance. We focus on the Sequential Forwarding algorithm operating in a homogeneous and simplified scenario.

We consider a system of N identical fog nodes in the limit of $N \to \infty$, each receiving a Poisson flow of job requests, with rate λ request per unit of time. The duration of a job is exponentially distributed with an average processing rate of μ. A single fog node is abstracted as a FIFO queue with Q places included the server. The key assumption is that as N grows these queues become independent from each other so that we can focus on and study a single queue [10]. The rates of the Continuous Time Markov Chain (CTMC) describing the queue dynamic are:

$$\mu_i = 1 \quad \lambda_i = \lambda \begin{cases} \frac{1-\pi_\Theta^{M+1}}{1-\pi_\Theta} & i \leq \Theta \\ \pi_\Theta^M & i > \Theta \end{cases} \tag{1}$$

where π_i is the steady-state probability that there are $j > i$ jobs in the queue, that is the state of the queue. The reason for the above formula is the following. Let i be the current state of the queue. When $i \leq \Theta$, the node accepts any incoming job. The flow of such jobs is λ. In addition, it may accept a job J forwarded by other fog nodes. This occurs when the job J was received by another fog node n, whose state was higher than Θ, and: (i) n sent J to our tagged node – the flow of this type of jobs is $\lambda\pi_\Theta$ or (ii) J is forwarded k times to other nodes whose state was higher than the threshold as well, and then the k-th node sent J to our tagged node. Summing up these flows we get:

$$\lambda + \lambda\pi_\Theta \sum_{k=0}^{M-1} \pi_\Theta^k = \lambda \frac{1-\pi_\Theta^{M+1}}{1-\pi_\Theta} \tag{2}$$

For $i > \Theta$, the queue only accepts a job if the job has already visited other M queues having already more than Θ jobs waiting for service.

The actual values that solve the equation can be determined numerically using a fixed point algorithm. We assume the steady-state solution of the CTMC is a unique fixed point $F(\vec{P}) = \vec{P}$, where $\vec{P}$ is the state vector and F is a function that computes the stationary solution of an M/M/1/Q queue whose rates are defined by Eq. 1. We assume that $\vec{\mu}$ is known (we assume $\mu_i = 1, \forall i = 0, \ldots, Q$) while $\vec{\lambda}$ takes the role of a parameter. Considering $\vec{\lambda}$ as a parameter is a reasonable assumption due to the independence among nodes [10]. The steady-state solution is numerically computed starting with a state vector where $p_0 = 1$ and $p_i = 0, \forall i = 1, \ldots, Q$, this because at the beginning every node is obviously at the initial state. From this initial vector, the first $\vec{\lambda}$ is computed and then state vector updated. This scheme keeps iterating until the euclidean distance between $\vec{P}$ and $\vec{P'}$ of two consecutive iterations ϵ is less than a pre-defined value (in our case $\epsilon \leq 10^{-13}$). The solution of the queue is the base for the definition of the following metrics: (i) average number of steps of a job, (ii) average queue length of a fog node, (iii) response time of a job and, (iv) loss probability of a job.

The average number of steps before a job is served S can be defined as follows:

$$S = \begin{cases} 0 & i \leq \Theta, \text{ or } M = 0 \\ (1 - \pi_\Theta) \sum_{k=1}^{M} k \pi_\Theta^k & i > \Theta \end{cases} \quad (3)$$

In fact, due to the independence among nodes, a job is served by the first queue out of the M possible trials, whose state is less than Θ.

The second relevant metric, that is the average queue length can be defined as:

$$W_Q = \sum_{k=1}^{Q} k \pi'_k \quad (4)$$

where π'_i is the steady state probability of the state i, $\pi'_i = \pi_{i-1} - \pi_i$.

Knowing that the average time for a job to be forwarded and allocated to another fog node is δ and that the average service time is $1/\mu$, we can determine the average response time from the time spent waiting in the queue defined in Eq. (4) and the number of steps S as in Eq. (3)

$$T_r = \frac{1}{\mu} + \delta S + \frac{1}{\mu} W_Q \quad (5)$$

Finally, a job is discarded if, over the M steps, no node was found with a load below Θ. The loss probability can be defined as:

$$p_B = \pi_\Theta^M \pi'_Q \quad (6)$$

IV. Experimental results

Throughout this section, we evaluate the performance of the proposed algorithms. We start describing the scenarios considered in our performance evaluations, next we discuss the main findings concerning the sequential forwarding algorithm, using both the numerical solutions of the model and a stimulative approach in a simplified scenario. After this preliminary evaluation, we focus on a highly heterogeneous scenario, based on a realistic geographic setup for a smart city and we evaluate in detail both the Sequential forwarding and the adaptive sequential algorithms.

A. Scenarios definition

In our experiments we consider both a simplified scenario, used for the initial evaluation of the sequential forwarding algorithm, and a realistic scenario based on a fog infrastructure aiming to support a smart city application. We anticipate that, in the simplified scenario, we use for the evaluation both the model proposed in Sec. III and a simulation. On the other hand, due to the complexity of the realistic scenario, we rely only on the simulator to analyze both sequential forwarding and adaptive forwarding algorithms.

The first scenario, namely *simplified scenario*, models the fog nodes as M/M/1/Q queuing systems, with an exponential distribution of both incoming jobs from the sensors and job service in the fog nodes. Concerning the workload, we assume $\lambda = 0.9$ jobs/sec (average rate of incoming jobs), while $\mu = 1$ job/sec (average processing rate), resulting in an expected system utilization $\rho = 0.9$. In our fog nodes, we assume a queue length $Q = 10$. Each fog node is connected to just one sensor and the network delay between each pair of fog nodes is $\delta = 0.1$s, that is the delay experienced every time a job is forwarded. In this scenario the model assumes an infinite number of fog nodes, while for the simulation, we consider a set of 20 fog nodes, that should be enough to capture the main characteristics of the algorithm performance.

The second scenario, namely the *realistic scenario*, is based on a case study carried out in Modena, a city in northern Italy of roughly 180'000 inhabitants. The proposed fog infrastructure uses a set of fog nodes to collect data on the vehicular traffic and on the air quality (a goal of the study is to contribute to the development of detailed models on the air pollution). The sensors for this application should be located on the main city streets, while a set of fog nodes are located in facilities belonging to the municipality. Long-range wireless links (such as IEEE 802.11ah/802.11af [11]) are used to transfer data from the sensors to the fog nodes and among the fog nodes themselves. Each sensor transmits the data to the nearest fog node, as in [8], and we assume the delay among fog nodes to be proportional to the distance between them.

The fog node behavior is modeled based on preliminary prototypes of the project, where a sensor captures images using a camera when movement is detected. The frames are sent to the fog node that should identify and count vehicles within each frame to create a real-time map of the traffic intensity throughout the city. The process of producing images is modeled using an exponential distribution. The processing time of a frame has been measured using a prototype and it can be described using a Gaussian probability distribution with an average value depending mainly from the image resolution. In our experiments, we consider that the average processing time $1/\mu = 10$ms (and with a standard deviation of 1ms), comparable with the average network delay, set to the same value of $\delta = 10$ms (that would correspond to a QVGA frame with a high JPEG compression factor transmitted over a long range IEEE 802.11ah link with a bandwidth in the order of 4Mbps). The network is based on a realistic setup of the fog infrastructure created using the PAFFI framework [9], with 100 sensors and 20 fog nodes. Due to the geographic placement of elements, the workload intensity is heterogeneous among the nodes, the workload intensity λ for each fog node ranging from 250 jobs/sec (2.5 times the processing rate of a fog node) to some fog nodes that are almost idle. The average load over the whole infrastructure is such that the average utilization $\overline{\rho} = \overline{\lambda}/\mu = 0.9$.

From a software tools point of view, the fixed point algorithm is implemented using Matlab[2], while the simulation is based on the Omnet++ framework[3], with additional modules developed *ad-hoc* to support the two proposed load balancing algorithms.

[2]https://www.mathworks.com/

[3]https://omnetpp.org/

Throughout the performance evaluation, the main performance metrics considered are:

- *Loss rate*, that is the probability of a job being discarded because the queue of the selected fog node is completely full. This condition is described for the model in Eq. (6).
- *Average Number of steps*, that is the average number of forwarding for a job before the job is processed – for the model, refer to Eq. (3)
- *Response time*, that is the time occurring between the moment the Job is received from the first fog node, to the moment the processing ends on the final fog node. The response time model is described in Eq. (4). For the simulator, in some cases, we consider useful to provide a breakdown of the response time components: that are *service time* (the time spent being processed), *balancer time* (the time spent being forwarded among the fog nodes), and *queuing time* (the time spent in the fog node ready queue waiting to be processed).
- *Fairness*, that is the measure of load balancing effectiveness. To quantify the fairness in the heterogeneous system, we consider the Jain index applied to the fog node utilization ρ. This measure is particularly interesting in the realistic scenario, characterized by high heterogeneity.

We recall that the Jain index for N resources is defined as:

$$\mathcal{J}(\rho) = \overline{\rho}^2 / \overline{\rho^2} = \frac{1}{1 + \mathrm{cv}(\rho)^2} \tag{7}$$

where $\rho = \{\rho_i, \ldots, \rho_N\}$ is the set of utilization values for the N fog nodes, and $\mathrm{cv}(\cdot)$ is the coefficient of variation, that is the ratio between the standard deviation and the average.

B. Evaluation in the simplified scenario

We now evaluate the impact of the values of the parameters Θ and M on the performance of the Sequential forwarding algorithm using the simplified scenario. In this set of experiments, we also compare the results from the model described in Sec. III with the results obtained using the simulator as cross-validation of the two methodologies.

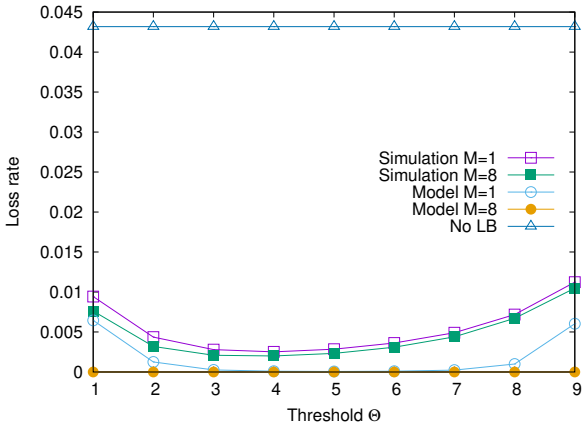

Fig. 2. Loss rate vs. Threshold Θ

The first analysis, shown in Fig. 2 concerns the loss rate as a function of Θ for different values of M. The performance of the No LB case is clearly poor, with a loss rate close to 5%. The sequential forwarding algorithm provides a clear benefit, with a loss rate significantly lower. Depending on Θ, the loss rate shows a cup-shaped curve, where the (relatively) high loss rate for low values of Θ are related to the case where a Job, after being forwarded M times due to the highly selective threshold, arrives at an overloaded node and is therefore dropped. For high values of Θ the opposite occurs and the scarcely effective load balancing determined by the high threshold results in a generalized risk of overload in the nodes. In this behavior, the parameter M plays a major role as it changes the number of chances a job has before being dropped. Clearly, the higher is M the lower is the loss rate.

Finally, comparing the results of the model and the simulator, we observe that both approaches capture the main characteristics of the algorithm. The discrepancy in the numeric value is likely due to the relatively low number of nodes used in the simulation.

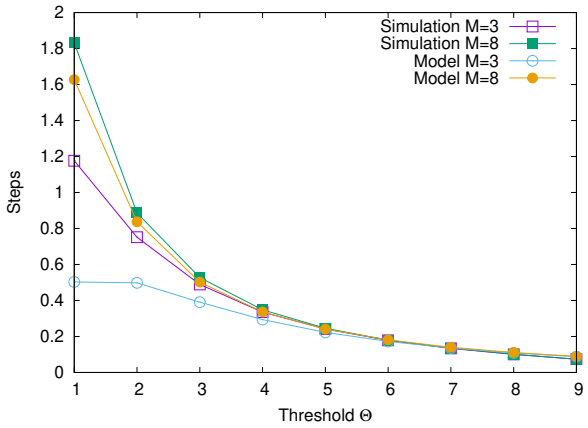

Fig. 3. Average Number of steps vs Threshold Θ.

Fig. 3 shows the average number of steps of a job before being processed. Again, we observe a clear reduction in the number of steps as the threshold Θ decreases due to the less stringent requirement on the number of queued jobs (load) in the sampled node. Moreover, even the maximum number of steps M plays a role as it may force a stop in the search for a fog node with a low load. The impact of M is more evident when Θ is low: indeed reaching a node with a very low load is a condition that may hard to fulfill if the number of steps is limited. The curve also shows that when this limitation is removed, i.e. M increases, the number of steps increases of a small amount, demonstrating that the random walk is a powerful mechanism for distributing the load. Again the main findings are confirmed by both the numerical results and by the simulations.

Finally, Fig. 4 shows the total service time T_r of a job. We observe that in most cases (that is, as long as $\Theta < 8$ and for every value of M), the proposed algorithm outperforms the No LB alternative from both from the response time and loss rate points of view. It is worth noting that, as Θ grows, the load balancing action becomes less effective, resulting in higher

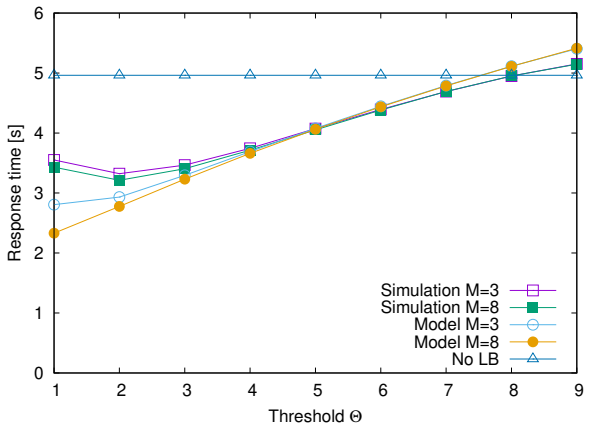

Fig. 4. Response time vs Threshold Θ.

response time. Again the results from both the simulation and the model are comparable, thus offering good cross-validation of the findings.

C. Evaluation in the realistic scenario

We now focus on the realistic scenario, that is a case where loads are unevenly distributed over the fog nodes, the network link delays are uneven and where the processing time is no longer described as an exponential (that is the fog nodes are described as M/G/1/Q queuing network elements). In this analysis, we consider both the sequential forwarding algorithm and its adaptive sequential extension, using only the simulator to evaluate the algorithm performance.

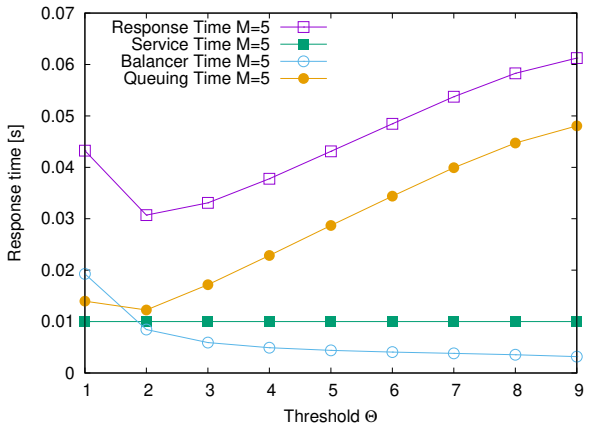

Fig. 5. Performance of Sequential forwarding algorithm

Fig. 5 shows the breakdown of the contributions to the response time: service time, balancer time and queuing time. The values are provided for $M = 5$, but are representative of the typical behavior of the algorithm.

Service time $(1/\mu)$ is constant, as expected, because it is a fixed parameter of the scenario. The balancer time (that is the time spent being forwarded) decreases as Θ grows. This behavior is consistent with the number of hops already shown in Sec. IV-B for the simple scenario.

For the queuing time, we observe a non-monotone behavior. For the first part $(\Theta = 1)$ we have a non-negligible impact of the last step (that is the M-th step), where the threshold is not considered. This determines a non-negligible amount of cases where the selected final fog node has a high load, thus explaining the higher queuing time. For the remaining part $(\Theta \geq 2)$ the queuing time grows with Θ because the threshold is less selective when we select whether we want to jump or not. As a consequence, we accept to have a job processed on nodes with a higher load, resulting in a longer queuing time (again, this behavior has already been described for the simple scenario in Sec.IV-B.

Hence, the response time is the sum of the three contributions. For low values of the threshold, we observe a high response time that is determined by both the higher number of steps and by the slightly longer queuing time. As a final remark, it is worth to note that the response time has a minimum for $\Theta = 2$

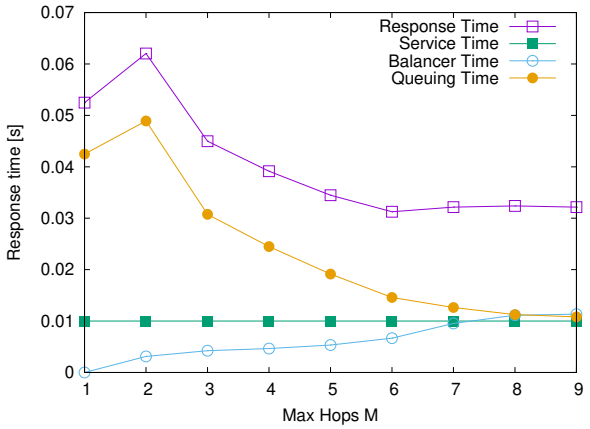

Fig. 6. Performance of Adaptive forwarding algorithm

Having understood the behavior of the sequential forwarding algorithm, we now evaluate the adaptive forwarding algorithm. To this aim, we recall that this algorithm aims at providing stable results without the need to tune Θ carefully. Indeed, at every step, the threshold Θ is increased by a value equal to Q/M so that, after M steps, the threshold is equal to Q and the forwarding is guaranteed to end. In this analysis, we consider M as the main parameter.

Fig. 6 shows the breakdown of the response time for the adaptive forwarding algorithm. As in Fig. 5 the service time remains constant, as expected. For the other contributions to the response time, we observe that when M is very low (e.g. $M = 1$) the algorithm cannot perform any adaptation: with the first step the threshold is already Q and the behavior is identical to the No Load Balancing case. As M grows we observe the emergence of the adaptive behavior: the balancer time grows with M because the increase in Θ is Q/M, hence it is inversely proportional to M. As consequence as M grows, more steps are required to tune the threshold. On the other hand, with the increase of M, the queuing time is reduced because we have more chances to find a fog node with a

sufficiently low threshold value. The final result is a confirmation that the algorithm shows stable performance. Indeed, while the sequential forwarding algorithm is characterized by a small range of Θ values where the minimum response time is found (shown in Fig. 5 for $\Theta = 2$) in the adaptive forwarding algorithm we have a large plateau of similar performance (for $M \geq 6$) where the adaptive algorithm guarantees stable performance.

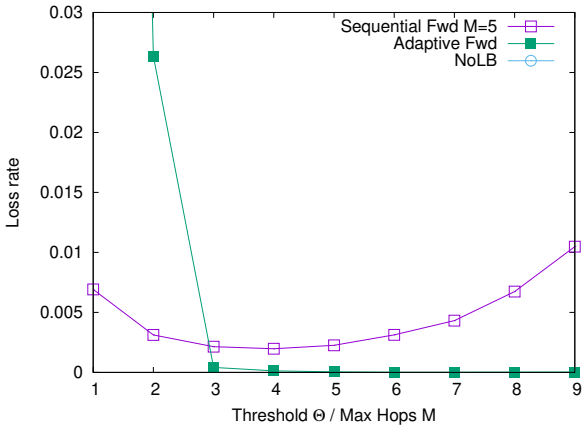

Fig. 7. Loss rate comparison of considered algorithms

Aiming to compare the main algorithms considered in this paper, in Fig. 7 and 8 we present the loss rate and the response time for the sequential forwarding, adaptive sequential a No LB approaches. As we are comparing algorithms with different parameters, we place on the X-axis both the threshold Θ, meaningful for the sequential forwarding algorithm and the maximum number of hops M for the adaptive sequential algorithm.

We recall that the considered scenario is characterized by a highly heterogeneous workload distribution, due to the geographic placement of sensors and fog nodes. The workload distribution can cause overload in part of the infrastructure while other fog nodes are badly underutilized. This explains the bad performance of the No LB algorithm where the loss rate is in the order of 25% (curve not shown in Fig 7 where the *Y*-axis is limited to 3%). On the other hand, the loss rate is below 0.5% for the sequential forwarding, with a cup-shaped curve similar to the one already discussed in Sec. IV-B. For the adaptive forwarding, we do nor register any loss in the range $M \geq 6$, providing the best performance in a load balancing algorithm. Even more important, the good performance are stable over a large range of values, confirming the ability of the algorithm to self-tune its parameters.

The performance gap is also confirmed by the comparison of the response time in Fig. 8, with the No LB algorithm, providing an average response time of 52 ms. On the other hand, the sequential forwarding algorithms can reach a response time of just 31 ms (for $\Theta = 2$), with a reduction of 40% compared to the No LB alternative. The adaptive forwarding reaches the same value of 31 ms, but this performance level is available

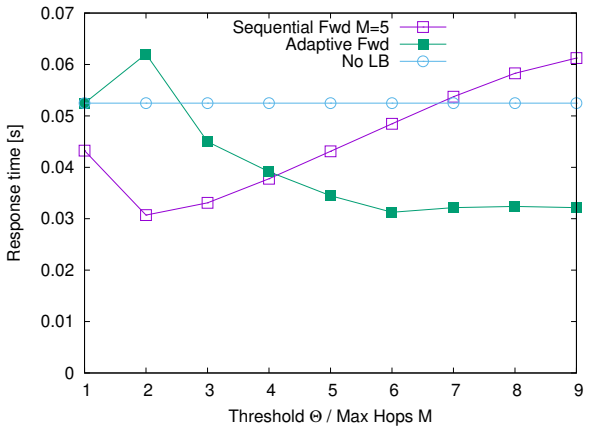

Fig. 8. Response time comparison of considered algorithms

for a larger range of values (for $M \geq 6$), confirming the more stable performance.

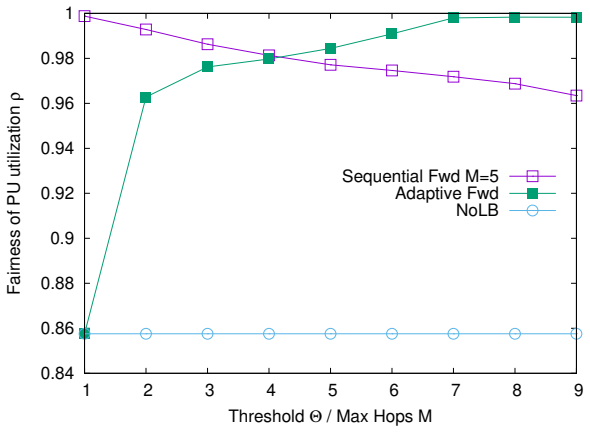

Fig. 9. Fairness comparison of considered algorithms

Given the uneven load distribution that characterizes the experimental setup, we also consider the ability of the algorithms to provide a load sharing among the nodes. We quantify this ability using the Jain fairness index applied to the fog nodes utilization ρ (the index has been introduced in Eq. (7) in Sec. IV-A). The results are shown in Fig. 9.

The No LB algorithm shows the highly unbalanced load distributions that characterize the considered experimental scenario, with an index close to 0.86.

The Sequential forwarding algorithm provides a very good load balancing when the threshold Θ is low (due to the aggressive load forwarding carried out). As the threshold increases, the fairness decreases as we accept more often to process jobs on a node with an already high load.

The adaptive forwarding algorithm presents a level of fairness that increases with the parameter M, confirming the general behavior of this parameter on the algorithm. It is worth noting that, for $M \geq 7$ the fairness index is equal to 1, meaning that the adaptive algorithm can provide a highly effective load balancing.

V. Related Work

A short summary of relevant research papers related to our problem is provided below.

In [12] 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. Our solution is fully distributed.

In [13], 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, again a centralized workload balancer is required by the solution.

In [14], An approach similar to the sequential forwarding algorithm is used. However, the proposed solution requires either a centralized repository to store the load state of each fog node or needs a specific protocol to send updates on the load state of each node. Our approach, based on a blind forwarding provides good performance without complex coordination structures.

In [15] an approach is presented to periodically distribute the 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 batch of tasks to be assigned is available, i.e., the tasks are not processed online as in our algorithm.

Finally, the idea of randomly select nodes to offload a task is used in the class of power-of-choices algorithms, adapted in [16], [17] to work in the fog deploy model. The key difference with the algorithm proposed in this paper is that tasks are forwarded without making any selection among alternatives and that self-adaption is absent.

VI. conclusions and future work

Throughout this paper, we proposed two new algorithms, namely *sequential forwarding* and *adaptive forwarding*, for the load balancing in a fog computing infrastructure. Our proposals aim to provide fair load sharing in a scenario characterized by a highly heterogeneous workload distribution as it the case of a realistic fog deployment. The algorithms are designed to be simple and fully decentralized without relying on any probing or reservation mechanisms.

We tested our proposal using both a mathematical model and a simulator. Our experiments in a realistic scenario prove how the algorithms outperform the case of no load balancing (No LB). The experimental evidence shows a loss rate dropping from 25% for the No LB case to less than 0.5% for the sequential forwarding algorithm, and up to 0% for the adaptive sequential forwarding. Furthermore, the response time is reduced by 40% by both the proposed algorithm w.r.t. the No LB case while achieving a perfect fairness condition (Jain index close to 1). In addition, we remark how the self-tuning adaptation mechanism can provide stable performance in terms of low response time and low loss rate for a wider range of configuration load parameters and load conditions. This paper is just a first step in a broader research line. We are currently investigating an extension of the current algorithms taking into account fog nodes with heterogeneous processing rates and mechanisms for the adaptive algorithm that consider non-linear relationship between the threshold and the number of steps.

References

[1] M. Satyanarayanan, W. Gao, and B. Lucia, "The computing landscape of the 21st century," in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '19, 2019.

[2] O. C. A. W. Group, "Openfog reference architecture for fog computing," in *https://www.openfogconsortium.org/ra/*. OpenFog Consortium, 2017.

[3] G. P. A. W. Group *et al.*, "View on 5g architecture," *White Paper, December*, 2017.

[4] J. J. et al., "Problem statement of iot integrated with edge computing," in *draft-hong-t2trg-iot-edge-computing-00, Work in Progess*, Expire Jan 2020.

[5] 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.

[6] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: Stateof-the-art and research challenges," in *IEEE Communications Surveys*. IEEE, 2017.

[7] R. Beraldi and H. Alnuweiri, "Sequential randomization load balancing for fog computing," in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Sep. 2018, pp. 1–6.

[8] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1171–1181, Dec 2016.

[9] C. Canali and R. Lancellotti, "Paffi: Performance analysis framework for fog infrastructures in realistic scenarios," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, Oct 2019, pp. 1–8.

[10] C. Graham, "Chaoticity on path space for a queueing network with selection of the shortest queue among several," *Journal of Applied Probability*, vol. 1, no. 37, pp. 198–211, 2000.

[11] E. Khorov, A. Lyakhov, A. Krotov, and A. Guschin, "A survey on IEEE 802.11 ah: An enabling networking technology for smart cities," *Computer Communications*, vol. 58, pp. 53–69, 2015.

[12] 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.

[13] 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.

[14] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog Computing: Towards Minimizing Delay in the Internet of Things," *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017*, pp. 17–24, 2017.

[15] 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)*, June 2017, pp. 32–39.

[16] 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.

[17] R. Beraldi and H. Alnuweiri, "Exploiting power-of-choices for load balancing in fog computing," in *2019 IEEE International Conference on Fog Computing (ICFC)*, June 2019, pp. 80–86.