Collaboration Strategies for Fog Computing under Heterogeneous Network-bound Scenarios

Claudia Canali, Riccardo Lancellotti, Simone Mione
Department of Engineering "Enzo Ferrari",
University of Modena and Reggio Emilia,
Email: claudia.canali@unimore.it, riccardo.lancellotti@unimore.it, 205212@studenti.unimore.it

***Abstract*—The success of IoT applications increases the number of online devices and motivates the adoption of a fog computing paradigm to support large and widely distributed infrastructures. However, the heterogeneity of nodes and their connections requires the introduction of load balancing strategies to guarantee efficient operations. This aspect is particularly critical when some nodes are characterized by high communication delays. Some proposals such as the Sequential Forwarding algorithm have been presented in literature to provide load balancing in fog computing systems. However, such algorithms have not been studied for a wide range of working parameters in an heterogeneous infrastructure; furthermore, these algorithms are not designed to take advantage from highly heterogeneous network delays that are common in fog infrastructures. The contribution of this study is twofold: first, we evaluate the performance of the sequential forwarding algorithm for several load and delay conditions; second, we propose and test a *delay-aware* version of the algorithm that takes into account the presence of highly variable node connectivity in the infrastructure. The results of our experiments, carried out using a realistic network topology, demonstrate that a delay-blind approach to sequential forwarding may determine poor performance in the load balancing when network delay represents a major contribution to the response time. Furthermore, we show that the delay-aware variant of the algorithm may provide a benefit in this case, with a reduction in the response time up to 6%.**

***Index Terms*—Fog Computing, Load-balancing, Simulation, Sequential Forwarding algorithm, Delay-aware algorithm**

I. Introduction

The increasing popularity of Internet of Things (IoT) applications based on the processing of data coming from geographically distributed sensors and online devices has lead to the adoption of the fog computing paradigm as a promising approach for the supporting infrastructures [1], [2]. In a fog computing infrastructure we place an intermediate layer of fog nodes located at the edge of the network. As shown in Fig. 1, these fog nodes act as intermediary between a plethora of sensors that collect data from the a wide range of activities following the IoT vision (bottom of the figure) and one or more cloud data centers that store the data and provide added values information extraction on them. See [3] for a more in-depth description of IoT-based innovative services. The fog nodes can perform several useful tasks such as filtering, aggregation, and alert triggering. These pre-processing steps occur close to the end-devices, thus reducing the volume of data exchanged with the cloud infrastructure and the response time experiences by the IoT applications tasks. The presence of an intermediate level of fog nodes is particularly advantageous in case of applications with critical requirements in terms of low and predictable latency such as gaming and augmented reality, control of autonomous vehicles, vehicular traffic monitoring [4]. The use of fog computing systems, however, opens new research issues in terms of strategies for resource allocation due to finite resources at the fog level, increasing number and complexity of applications, and heterogeneity of incoming load due to mobile traffic [5], [6]. Load balancing, in particular, represents a critical feature for the system performance in complex scenarios where the network delays between the constituting elements of the infrastructure can be highly heterogeneous and the workload intensity can vary over time.

978-1-7281-8326-8/20/$31.00 ©2020 IEEE

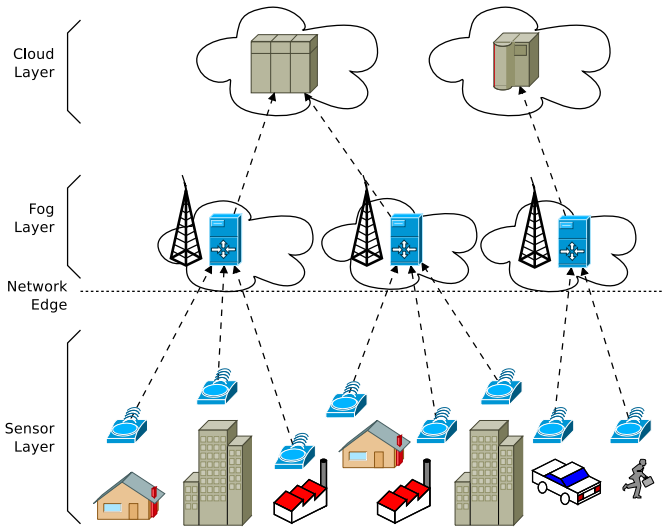


Fig. 1: Fog computing infrastructure

Several solutions for load balancing in fog computing systems available in literature rely on a centralized component that acts as load orchestrator [7], [8]. A fully centralized approach can achieve competitive performance, but has the weakness of high computational complexity and huge reporting overhead. Other solutions, based on a distributed approach, typically assume that each node has some knowledge of the load on (all or a subset of) other nodes to make an informed selection of the fog node to forward the incoming task [9]. This approach requires some specific protocol to send updates

on the load state of each fog node and may cause high traffic overload on the network, with the risk of delayed and stale information about the nodes load. Moreover, the load balancing mechanisms currently available in literature are typically unaware of the heterogeneous network delay that can characterize the fog infrastructures. For these reasons, our study focuses on a fully distributed load balancing approach, that does not require any exchange of load information among fog nodes, and investigates the impact of delay awareness on load balancing under different load and network conditions.

The contribution of this paper is twofold. First, we carry out a comprehensive evaluation of an existing alternative approach based on random walks, namely sequential forwarding [10]. Specifically, we consider a wide range of scenarios by varying the system load and the ratio between the network delay and the task service time (thus creating CPU- and Network-bound scenarios) to fully understand the behavior of this delay-blind load balancing algorithm. Second, we propose an improved delay-aware version of the sequential forwarding algorithm that is able to take advantage from the knowledge of the infrastructure delays among fog nodes. We evaluate the effects of the delay-aware approach on the system performance by means of simulation considering the main elements of the fog infrastructure placed according to a realistic topology. The results of our experiments show that a delay-aware mechanism is effective in reducing the time spent to forward jobs among neighbors. Moreover, our experiments evidence how delay awareness may in some cases decrease the effectiveness of the load balancing due to a less randomized choice of the destination fog node. However, delay awareness provides significant benefits when the scenario is network-bound, that is when network delay represents a major contribution to the response time, leading to a reduction in response time up to 6%.

The rest of this paper is organized as following. Section II describes the proposed algorithm. In Section III we provide an experimental evaluation of the proposed algorithm. Section IV discusses related works while conclusions and future research directions are provided in Section V.

II. Algorithm

In this section we present the main algorithms considered in this research for load balancing in a fog scenario. First, we present the basic *Sequential Forwarding* algorithm introduced in [10]. Considering that a heterogeneous fog structure also from a connectivity point of view is common in the real world, we consider scenarios where the network delay can have a major impact on the overall system response time and where network delay between each couple of fog nodes cab be highly variable. This motivates the proposal of a *Delay-Aware Sequential Forwarding* algorithm that forwards the jobs just to a set of the closest neighbors and with a probability that favors the closest ones.

A. Sequential forwarding algorithm

We now discuss the *Sequential Forwarding* algorithm (originally proposed in [10]) that is a basis for our analysis. The algorithm uses a threshold Θ to decide if an incoming job must be forwarded to a neighbor or not. Furthermore, the algorithm relies on an additional parameter H (that is the a maximum number of forwarding hops), in order to guarantee an upper limit on the delay experienced by each job during the load balancing phase. A detailed description of the sequential forwarding algorithm is provided in [10]. In this paper we explore in detail the performance of this algorithm and we use this analysis as a basis for the proposal of a delay-aware algorithm.

Algorithm 1 Sequential Forwarding Algorithm

```
Require: H, Θ, Job
  if Job.Hops() ≥ H then
    ProcessLocally(Job)
  else
    if System.Load() ≤ Θ then
      ProcessLocally(Job)
    else
      Neigh ← SelectNeighbor()
      Job.IncrementSteps()
      Forward(Job, Neigh)
    end if
  end if
```

The Sequential Forwarding is detailed in Alg. 1. The two nested *if* statements are used to handle the maximum number of hops H and the threshold Θ. We assume that the data structure describing the job is enriched with meta-data to keep track of the number of times a job is forwarded. The job is forwarded to a neighbor if the local load exceeds a threshold Θ and if the number of forwarding hops is less than H. Otherwise the job is processed locally.

Algorithm 2 Local processing: *ProcessLocally*()

```
Require: Job
  if System.Queue() < System.MaxQueue() then
    Enqueue(Job)
  else
    Drop(Job)
  end if
```

Algorithm 2 details the case where a node is processed locally (that is when the *ProcessLocally*() procedure is called in Alg. 1). In this case, the Job should be enqueued in the ready queue of the server. However, since this queue is finite in size, if the queue is already full, the job is dropped, resulting in a loss.

Finally, in Alg. 3 we discuss the function *SelectNeighbor*() used in Alg. 1. In the Sequential Forward algorithm, the function simply returns a random node among the list of neighbor nodes. This function will be extended in the *Delay-Aware*

Algorithm 3 Neigh. Selection (delay-blind): *SelectNeighbor*()

$\mathcal{N} \leftarrow$ System.*Neighbors*()
return *Random*($\mathcal{N}$)

version of the Sequential forwarding algorithm described in the next subsection.

B. Delay-Aware Sequential Forward

We now discuss the *Delay-Aware Sequential forwarding* algorithm. The algorithm introduces two principles to take advantage from the knowledge of the infrastructure delays between fog nodes. Let i and j be two nodes of the infrastructure, We define as $\delta_{i,j}$ the delay between them. First, we limit the number of possible neighbors that can receive a job from a generic node i to the value of K. To this aim, we introduce a set of nodes $\mathcal{M}_i$ (where $|\mathcal{M}_i| = K$) that are the K nearest neighbors to node i. The neighbor to which a job is forwarded is selected from this set. Second, the probability to select a neighbor node j node is inversely proportional to the communication delay $\delta_{i,j}$. We can define the probability $p_{i,j}$of selecting node j from node i for job forwarding as in Eq. (1)

$$p_{i,j} = \begin{cases} \frac{\delta_{i,j}^{-1}}{\sum_{l \in \mathcal{M}_i} \delta_{i,l}^{-1}}, & \text{if} j \in \mathcal{M}_i \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where $\delta_{i,j}$ is the delay between node i and node j, and $\mathcal{M}_i$ is the set of the K best connected neighbors to node i.

It is worth to note that the mechanism of weighted random selection of the node may produce paths including the same node more than once. However, the randomness involved in the choice of the next hop determines an almost negligible probability of having requests performing cycles among the fog nodes. Moreover, a request reaching again a previous node at a different time will find it in a different load condition. These effects and their impact on the load balancing performance are discussed in detail in [11].

Algorithm 4 Neigh. Selection (delay-aware): *SelectNeighbor*()

Require: K
$\mathcal{N} \leftarrow$ System.*Neighbors*()
$\mathcal{D} \leftarrow$ System.*NeighborDistances*()
$\mathcal{M} \leftarrow$ *SelectNearestK*($\mathcal{N}$, $\mathcal{D}$, K)
$\mathcal{P} \leftarrow$ *ComputeProbability*($\mathcal{M}$, $\mathcal{D}$)
return *WeightedRandom*($\mathcal{M}$, $\mathcal{P}$)

Alg. 4 shows how the *SelectNeighbor*() function (introduced in Alg. 1) is re-defined for the Delay-Aware Sequential Forward algorithm. We retrieve the information about the neighbors and their distances ($\mathcal{N}$ and $\mathcal{D}$ sets). As we previously said, a neighbor is selected in the set of the K nearest neighbors, so we restrict our search to the set $\mathcal{M}$ (we omit the suffix i for a leaner notation). Furthermore, we define the probability of being selected as in Eq. (1). This set of probabilities is then used for the actual neighbor selection.

III. Experimental results

In this section, we evaluate the performance of the Sequential Forwarding algorithm [10], and we compare it with its evolution Delay-Aware Sequential Forwarding. We start describing the considered scenarios in our performance evaluations, next we discuss the main findings concerning the Sequential Forwarding analyzing its performance over a wide set of scenarios. To this aim, we use a simulative approach to explore different network configurations and load conditions. After this preliminary evaluation, we analyze the performance of the Delay-Aware Sequential Forwarding algorithm focusing on a set of the most significant scenarios.

A. Scenarios definition

In our experiments we consider a realistic scenario based on a fog infrastructure, aiming to support a smart city application. This infrastructure is highly heterogeneous, so some nodes are near one to each other while others are characterized by significantly higher network delays.

All the considered scenarios are 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. The sensors for this application are placed on the main city streets, while a the fog nodes are in facilities belonging to the municipality. We use long-range wireless links (for example, IEEE 802.11ah/802.11af [12]) to transfer data from the sensors to the fog nodes and between pairs of fog nodes. Each sensor sends the data to the nearest fog node, that is common choice in smart sensing deployment [13]. In most long-range wireless links the achievable data rate (that is the available bandwidth) is inversely proportional to the communication distance. From this observation, we assume the delay among fog nodes to be proportional to the distance between them. It is worth to note that the considered model is a simple but effective approach to introduce distance-related delays in a geographic infrastructure that has been used in literature [14]. Other, non linear, distance-to-delay relationships (e.g, quadratic or piece-wise interpolations) have been evaluated in preliminary tests with limited differences with respect to the setup used in these experiments. For this reasons we opted for the simplest model available. The fog node behavior is modeled based on a preliminary project prototype. In the prototype a sensor captures images of the street when movement is detected. The frames are sent to the fog node that uses a neural network-based to identify and count vehicles. This creates a real-time map of the traffic intensity throughout the city. The process of sending images is modeled using an exponential distribution. The time to process a frame has been characterized on the prototype and can be described using a Gaussian probability distribution with an average value depending mainly from the image resolution. In our experiments, we define the average processing time $1/\mu = 10$ ms (and with a standard deviation of 1ms). For the average network delay $\overline{\delta}$ we consider multiple values. In particular, we introduce a scenario parameter $\delta\mu$ that is the

ratio between the average network delay $\overline{\delta}$ and the average processing time $1/\mu$. We consider values of the $\delta\mu$ scenario parameter ranging form 0.1 (CPU-bound scenario) to 2 (network bound scenario). The network topology is based on a realistic setup of the fog infrastructure created using the PAFFI framework [14], with 100 sensors and 20 fog nodes. Due the geographic placement of elements, the workload intensity is heterogeneous among the nodes, with the workload intensity λ for each fog node ranging from 2.5 times the processing rate of a fog node to some fog nodes that are almost idle. We run the experiments with different average utilization $\rho = \overline{\lambda}/\mu$ (with $\overline{\lambda}$ being the average incoming job rate in the set of fog nodes belonging to the infrastructure). The average load over the whole infrastructure ρ ranges from 0.5 to 0.9, with a step of 0.1. Furthermore, in our sequential forwarding algorithm, we set the parameter $H = 5$ that is a value proved in previous experiments to provide low response time and low drop rate [10]. The maximum queue length is limited to 10 jobs, so $\Theta \in [1, 10]$. In order to avoid an explosion of the parameter space of our experiments, we omit some results of our sensitivity analyses. In particular, we do not present results whenever our tests show that a parameter has not a major impact for our proposal or when our tests confirm results already available in literature (especially in [10], [11]).

The simulation is based on the Omnet++ framework[1], with additional modules developed ad-hoc to support the sequential forward algorithm and the nearest neighbors mechanism.

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

- *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.
- *Loss rate*, that is the probability of a job being discarded because the queue of the selected fog node is completely full or the job is expired.

In some cases, we consider useful to provide a breakdown of the response time components that are:

- *Service time* that is the time spent being processed (average service time is $1/\mu$),
- *Balancer time* that is the time spent being forwarded among the fog nodes for load balancing,
- *Queuing time* that is the time spent in the fog node ready queue waiting to be processed.

B. Evaluation of Sequential Forwarding Algorithm

We now evaluate the performance of the Sequential Forwarding algorithm in different scenarios. This analysis aims to understand how the load ρ and the impact of network delay $\delta\mu$ affect the performance of the load balancing algorithm.

Fig. 2 shows the response time as a function of the threshold Θ for $\rho = 0.8$ and $\delta\mu = 1.0$. Furthermore we show a breakdown of the response time components: balancer time, queuing time and service time. While the service time is not dependent from the threshold Θ, we observe that the balancer

[1]https://omnetpp.org/

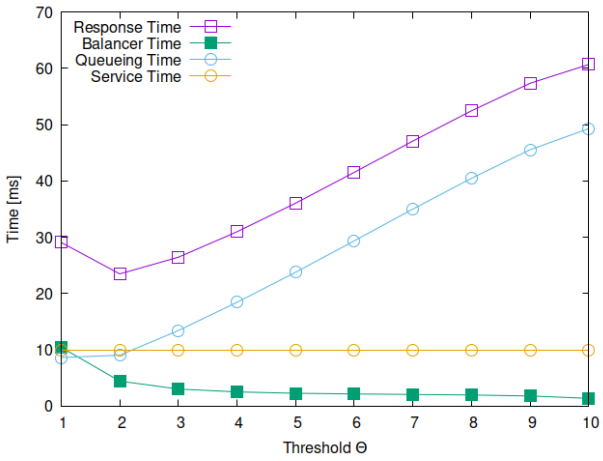


Fig. 2: Breakdown of Response Time *vs.* Θ ($\delta\mu = 1.0$ $\rho = 0.8$)

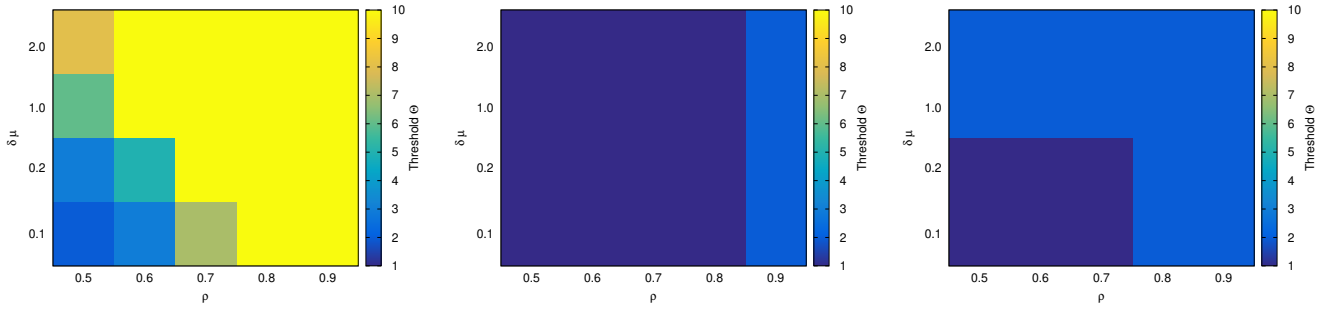
time decreases as Θ grows. Indeed, with an higher threshold we activate less frequently the balancing mechanism. At the same time, when we seldom use the load balancing, we are more likely to observe longer queues, as testified by the increase in the queuing time. The graph shows an optimal point for $\Theta = 2$ that minimizes the response time.

To better understand how the components of the response time are affected by the scenario parameters $\delta\mu$ and ρ, we now present, in Fig. 3, the threshold values which minimizes, respectively, balancer time, queuing time and response time (we recall that service time is not affected by Θ so we don't graph this component). In each heatmap, the color scale (on the right of each figure) uses blue hues when the best value of Θ is close to 1 and yellow hues when the optimal Θ approaches the maximum value of 10.

Fig. 3a shows the threshold that minimizes the balancer time. In most cases the time spent in load balancing is minimized by the maximum threshold value $\Theta = 10$ (as suggested by Fig. 2). However, when the load is quite low (e.g., $\rho \leq 0.7$), the balancer time becomes negligible also for lower threshold values, because the load balancing is rarely needed. The reduction of the impact of network delay ($\delta\mu$) further reduces the absolute value of the balancer time, accelerating the situation when the balancer time become negligible for low values of Θ.

Conversely, Fig. 3b shows that value of Θ that minimizes the queuing time depends mainly on the system load ρ. When $\rho \leq 0.8$ we observe the shape of the queuing time curve shown in Fig. 2. On the other hand, when load is high, we there is a minimum for the queuing time when $\Theta = 2$.

To summarize, we have two contributions (balancer and queuing times, respectively) that are reduced by opposite values of the threshold. Fig. 3c shows that the threshold Θ that minimizes the response time is 1 for $\delta\mu < 1$ and $\rho < 0.8$. When the network delay is low, the impact of the balancer time is less evident and the queuing time dominates completely the threshold choice. In the other hand, when $\delta\mu \geq 1$ or $\rho \geq 0.8$, we have a more evident impact of the balancer time



(a) Minimize Balancer Time (b) Minimize Queuing Time (c) Minimize Response Time

Fig. 3: Threshold Values to optimize different objectives

and avoiding the high number number of balancer activation that characterized the case where $\Theta = 1$ minimizes the overall response time.

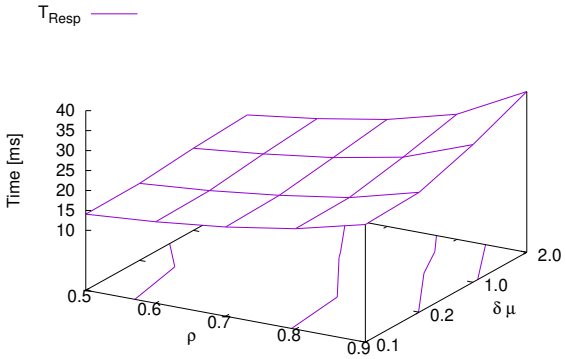


Fig. 4: Minimum T_{Resp} w.r.t. $\delta\mu$ and ρ

To summarize the evaluation on the response time, Fig. 4 presents the best response time (considering the optimal values of Θ in Fig. 3c) for every considered value of $\delta\mu$ and ρ. We observe that the system utilization ρ determines an increase in response time (due to longer queuing times in the nodes) and we observe that this increase is more evident as $\delta\mu$ grows (amplifying the contribution of the time spent in the load balancer due to the higher network delays).

Another critical performance metric is the drop rate, shown in Fig. 5 as a function of Θ. From the figure we observe that the impact of $\delta\mu$ is almost negligible on this parameter. On the other hand, we observe that when the system load is high (e.g., $\rho = 0.8$), the drop rate increases because the average queue length grows, with a consequent increase in the probability of saturating the queue. Furthermore, when the threshold is too high (e.g., $\Theta \geq 7$), the packets loss is caused by the presence of long queues in some nodes, because the load balancing mechanism is not activated with a sufficient frequency to be effective in avoiding overload.

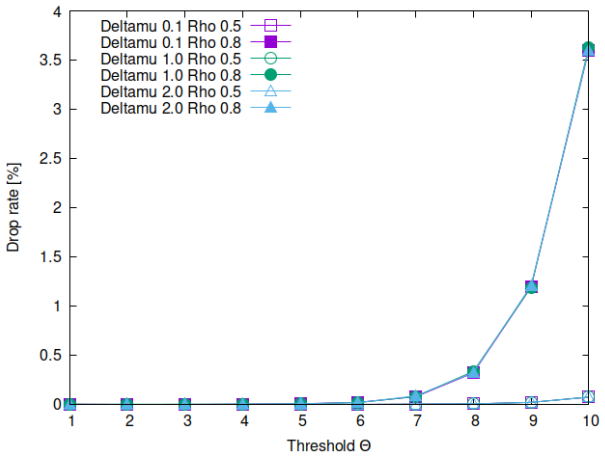


Fig. 5: Drop Rate vs Threshold Θ.

C. Evaluation of Delay-Aware Sequential Forwarding Algorithm

We now introduce the delay-aware mechanism in the load balancing algorithm. This mechanism serves to take advantage of the heterogeneity in the fog structure that is common in real applications. However, it is worth to note that the delay aware mechanism may have a twofold effect. On one hand, it reduces the time related to network delays, reducing the load balancer time. On the other hand, it reduces the randomness of the neighbor selection mechanism (because we limit the neighbor nodes and because we prefer closer nodes to more distant ones). This may have a detrimental effect on the load balancing. Depending on the values of ρ and $\delta\mu$, we expect one of the two effect to dominate the other. It is then important to understand when the delay-aware mechanism can provide an actual benefit.

Fig. 6 presents the performance of the delay-aware algorithm for three significant examples: low load and CPU-bound scenario ($\rho = 0.5$, $\delta\mu = 0.1$), high load and balanced network/CPU contributions scenario ($\rho = 0.8$, $\delta\mu = 1.0$), high load and network-bound scenario ($\rho = 0.8$, $\delta\mu = 2.0$). For each scenario we present the case where $K = 1$ (purple), that is forwarding occurs only to the closest neighbor, $K = 10$

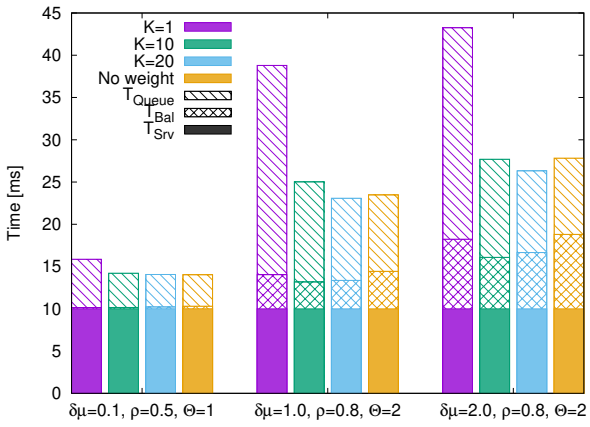


Fig. 6: Impact of Delay-Awareness

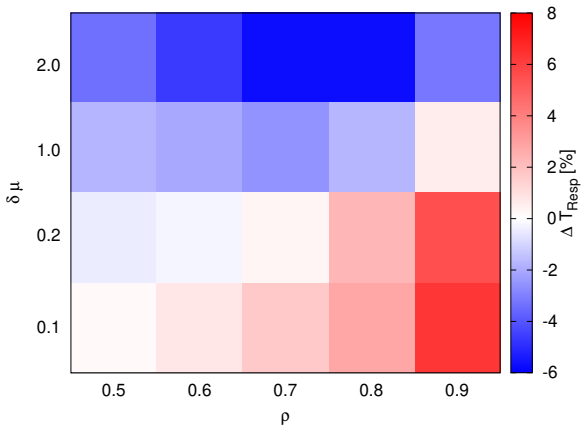


Fig. 7: Response time variation

(green), that is when only the 50% closest neighbors can be selected for the forwarding. The last two columns (blue and yellow) involve every neighbor in the selection, but in one case we use weights in the selection ($K = 20$, blue color), while the last case (namely "No weight", yellow color) is the traditional, not weighted, Sequential Forwarding algorithm. Furthermore, in every column we present the breakdown of response time into service time (solid color), balancer time (grid pattern) and Queuing time (oblique lines pattern). From the comparison we observe that the case where $K = 1$ result in poor performance in every scenario because the forwarding is deterministic and is not effective in sharing the load among the nodes. We observe how the increase of the response time is driven by an explosion of the queuing time, proof of the poor load balancing. A similar effect on queuing time, even if not so evident, occurs also when $K = 10$. However, in this case we observe how the increase of the queuing time is (partially) mitigated by a reduction in the balancer time (at least when $\delta\mu \geq 1.0$). Finally, we observe that, when we use all the neighbors, but we add weights in their selection, the reduction in the balancer time when $\delta\mu$ is high (e.g. $\delta\mu = 2.0$) overweights the slightly higher queuing time.

For this reason in the following of the section we focus just on the comparison of the case where $K = 20$ with respect to the standard Sequential Forwarding algorithm. Repeating the analyses carried out in the previous section and shown in Fig. 3, we obtain that the optimal threshold value Θ are the same for both considered algorithms. Hence, in the following comparison we will refer to the same threshold value Θ that is optimal for both alternatives.

Fig. 7 explores the benefits delay-aware algorithm $K = 20$ over Sequential Forwarding solution, presenting the relative difference in the response time for every value of ρ and $\delta\mu$. In the heat map blue hues are used when the proposed delay-aware algorithm outperforms the standard Sequential Forwarding solution, while red hues are used when the the delay-awareness is not increasing the response time (likely due poor load balancing). Finally, faded colors (close to white) are used when the two options provide similar performance. To evaluate the difference in response time we rely on a metric defined as:

$$\Delta T_{Resp} = \frac{T_{Resp}^{K=20} - T_{Resp}^{NW}}{T_{Resp}^{NW}}$$

Where T_{Resp}^{NW} is the response time of the Sequential Forwarding algorithm ("No Weight" version) and $T_{Resp}^{K=20}$ is the Delay-Aware Sequential Forwarding with K=20. We observe that, when the load is high (e.g., $\rho = 0.9$), even a minor reduction in load balancing effectiveness determines a growth in queuing time that is hard to compensate, as testified by the red hues in the rightmost part of Fig. 7. At the same time, when the scenario is network-bound (e.g., $\delta\mu \geq 1.0$) a more efficient selection of the neighbor that reduces the balancer time provides a non-negligible benefit. In Fig. 7 this is evident in the upper part of the graph, where the reduction in response time can be up to 6%.

IV. Related Work

In this section we briefly summarize the relevant research papers related to the issue of load balancing in distributed fog computing systems.

In several studies in literature, the proposed solutions consider a centralized component that has the role of workload balancer with a complete knowledge of the load status of the fog nodes of the infrastructure. For example, in [7] 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. This load balancing scheme adopts a centralized fog management layer that receives all the state information about the fog nodes. The solution proposed in [8] considers tasks that are characterized according to their computational nature and are consequently 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. Also in this case a centralized workload balancer is required by the solution. A last centralized solution, based on genetic algorithms for the design

of load-balanced solution for fog computing infrastructures, is presented in [15]. The approach assumes a stationary load, such that a centralized optimization algorithm can run once during the initialization phase to map the sensors over the fog nodes. The presence of a centralized balancer and the required exchange of information to provide this component with the complete knowledge about the nodes load may hinder the performance of the fog system and cause network traffic overload when the load changes over time. On the other hand, the load balancing mechanism proposed in this paper is fully distributed and does not require any load information exchange.

The solution presented in [16] is based on a periodic distribution of the incoming tasks among the nodes of the edge computing network so to increase the number of tasks that can be processed, while satisfying the quality-of-service (QoS) requirements for the completion of the tasks. The assumption behind this model, however, is about a different context: indeed, the authors assume that a batch of tasks to be assigned is always available, i.e., the tasks are not processed online as in our solution.

An approach more similar to the sequential forwarding algorithm is used in [9]. However, in this case the proposed solution requires either a centralized repository to store the load state of each fog node or a specific protocol to send updates on the load state of each node. On the other hand, our approach is based on a blind forwarding not requiring the knowledge of the nodes load, and may provide good performance without the need for a complex coordination structures with load information exchange.

The idea of randomly selecting fog nodes to offload a task is typically used in the class of power-of-choices algorithms, that have been adapted in [17], [18] to work in fog computing systems. The key difference with the algorithm proposed in this paper is that tasks are forwarded without making any selection based on the load of the possible alternatives and that self-adaptation mechanisms are absent.

Finally, we need to mention the previous study where the sequential forwarding algorithm was initially proposed [10]. This paper represents a clear step ahead with respect to the previous one because we explore in detail the performance of the sequential forwarding algorithm and, on the basis of the results of this analysis, we propose the delay-aware algorithm that is able to take advantage from the knowledge of the infrastructure delays between fog nodes.

V. CONCLUSIONS AND FUTURE WORK

In this paper we pointed out the critical role of the fog computing in modern IoT applications and we discussed how load balancing is a critical feature to cope with fluctuating workloads that are likely to occur in such widespread and complex scenario. Furthermore, we pointed out that currently available load balancing mechanism are typically unaware of the heterogeneous network delay that can characterize Fog infrastructures. Throughout this paper we address this issue with a twofold contribution. First, we propose a comprehensive study of how the main parameters of load and ratio between network delay and service time affect the behavior of a delay-blind load balancing algorithm, namely Sequential Forwarding. Second, we propose an improved delay-aware version of the algorithm and we evaluate its performance.

Our study, carried out using a realistic topology based on a prototype testbed, shows that introducing a delay-aware mechanism has two effects. First, it reduces the contribution to the response time related to forwarding jobs among neighbors. Second, as the selection of the neighbor is less randomized compared to the original sequential forwarding algorithm, the effectiveness of the load balancing is reduced, possibly resulting in a growth of the response time. Our experiments show that, when the overall system load is high and the scenario is CPU-bound, delay awareness results in a performance degradation. On the other hand, when the scenario is network-bound, the benefit from delay awareness overweights the reduced effectiveness of the load balancing, providing a reduction in response time up to 6%.

This paper is just a first step in a wider research line. We aim to explore new cooperation mechanism and to evaluate the impact of delay awareness on them. Furthermore, we are currently investigating adaptive mechanisms that can automatically select or disable delay awareness based on the sensing of network delays and queuing times.

REFERENCES

[1] M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 1826–1857, 2018.

[2] M. Mahmud and R. Buyya, "Fog computing: A taxonomy, survey and future directions," *Internet of Everything - Algorithms, Methodologies, Technologies and Perspectives, Springer*, pp. 103–130, Nov. 2018.

[3] 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*, Paris, France, May 2019.

[4] H. Inaltekin, M. Gorlatova, and M. Chiang, "Virtualized control over fog: Interplay between reliability and latency," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5030–5045, 2018.

[5] S. Chen, T. Zhang, and W. Shi, "Fog computing," *IEEE Internet Computing*, vol. 21, no. 2, pp. 4–6, 2017.

[6] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: A network perspective," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 783–791.

[7] E. C. Pinto Neto, G. Callou, and 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.

[8] 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.

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

[10] R. Beraldi, C. Canali, R. Lancellotti, and G. Proietti Mattia, "A random walk based load balancing algorithm for fog computing," in *The Fifth International Conference on Fog and Mobile Edge Computing (FMEC 2020)*, Jul. 2020, pp. 46–53.

[11] R. Beraldi, C. Canali, R. Lancellotti, and G. Proietti, "Randomized load balancing under loosely correlated state information in fog computing," in *23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'20)*, Alicante, Spain, Nov. 2020.

[12] 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.

[13] 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.

[14] 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.

[15] C. Canali and R. Lancellotti, "A fog computing service placement for smart cities based on genetic algorithms," in *Proc. of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, Heraklion, Greece, May 2019.

[16] Y. Song, S. S. Yau, R. Yu, X. Zhang, and G. Xue, "An approach to qos based task in edge computing networks for iot applications," in *International Conference on Edge Computing*, 2017.

[17] 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.

[18] 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.