A Load Balancing Algorithm for Equalising Latency Across Fog or Edge Computing Nodes

Gabriele Proietti Mattia, Antonio Pietrabissa, and Roberto Beraldi

***Abstract*—When dealing with distributed applications in Edge or Fog computing environments, the service latency that the user experiences at a given node can be considered an indicator of how much the node itself is loaded with respect to the others. Indeed, only considering the average CPU time or the RAM utilisation, for example, does not give a clear depiction of the load situation because these parameters are application- and hardware-agnostic. They do not give any information about how the application is performing from the user's perspective, and they cannot be used for a QoS-oriented load balancing. In this article, we propose a load balancing algorithm that is focused on the service latency with the objective of levelling it across all the nodes in a fully decentralised manner. In this way, no user will experience a worse QoS than the other. By providing a differential model of the system and an adaptive heuristic to find the solution to the problem in real settings, we show both in simulation and in a real-world deployment, based on a cluster of Raspberry Pi boards, that our approach is able to level the service latency among a set of heterogeneous nodes organised in different topologies.**

***Index Terms*—Edge computing, fog computing, load balancing, service latency.**

I. Introduction

SERVICE latency plays a crucial role in modern distributed applications [1]. In particular, in the Edge and Fog Computing environments, due to the geographic displacement of the nodes, some of them can be subjected to more traffic than others. In these situations, for designing an effective and QoS-oriented load balancing algorithm, it is not possible to consider only the typical hardware parameters that regard, for example, the CPU load, the RAM utilisation or the network traffic. This is because all of these performance indicators are both hardware and application-agnostic, they do not consider that the devices may be heterogeneous, and the same application on different hardware performs differently. Suppose that we have two Edge or Fog nodes Node A and Node B with two different CPUs, CPU A and CPU B respectively. Suppose that we designed an algorithm that enables nodes to cooperate, and some nodes can forward part of their flow of tasks to be executed to another node. Also, suppose that we designed an algorithm which is able to level the CPU time and in the end both CPU A and B are levelled to 50%. If there are no differences in network delays, we now wonder if the users that will make requests to Node A will experience the same latency of the users which will make requests to Node B. The answer is yes, but only in one case, the performance of CPU A must be exactly equal to one of CPU B, a characteristic of the system which is not common in Edge or Fog computing and even if we deploy the same hardware, we will never have exactly the same performances, due to background processes of the OS and intrinsic hardware differences. Given these conditions, it is necessary to change the performance indicators which drive the balancing, we need to design an algorithm which is able to balance the QoS that each user will experience: each user, independently from the node at which it will request the service, will have to experience the same service latency. The latency can be intended as a performance parameter which best describes how the application is behaving, independently of the effective load situation. Therefore by levelling the latency of the service, we will probably not balance the CPU load. Indeed, slower devices will be, in general, less loaded than the faster ones because they will saturate when the load is lesser than the faster ones. But in general, we will be sure that each user will experience the same QoS as the others since there will be no user that will experience a higher or a lower service latency than the other. The motivation of this work is clear, and our principal focus is designing, in a fully decentralised environment (that particularly fits the Fog and Edge Computing models) with no central entity, a load balancing algorithm that is able to level the service latency across all the nodes by tuning the percentage of tasks that a node can forward to another, a percentage that we call the *migration ratio*. In other words, each node can decide if and at which level it can cooperate with others offloading part of its work for reducing its service latency until it reaches a stable value that is equal across all the neighbours when this is possible, or at least closer to the value of the others.

The contributions of the papers can be summarised as follows.

- A continuous-time model which describes the dynamics of the system by using a system of differential equations that reaches stability when all the nodes experience the same service latency;
- Mathematical proof of convergence to a Wardrop equilibrium of the continuous-time model;
- An heuristic algorithm which tries to find a solution to the problem in a real environment by continuously adapting the migration ratios in rounds of fixed duration;

Manuscript received 17 November 2022; revised 8 February 2023; accepted 4 April 2023. Date of publication 10 April 2023; date of current version 8 October 2023. This work was partially supported by the SERICS under Grant PE00000014, in part by the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. Recommended for acceptance by S. Deng. *(Corresponding Author: Gabriele Proietti Mattia.)*

The authors are with the Department of Computer, Control and Management Engineering "Antonio Ruberti", Sapienza University of Rome, 00185 Rome, Italy (e-mail: proiettimattia@diag.uniroma1.it; pietrabissa@diag.uniroma1.it; beraldi@diag.uniroma1.it).

Digital Object Identifier 10.1109/TSC.2023.3265883

TABLE I
LIST OF SYMBOLS USED

Symbol	Meaning
	Model
$\mathcal{N}$	Set of nodes
A	Adjancency matrix
a_{ij}	Cell of the adjacency matrix that is 1 if node i can communicate with node j, otherwise 0
$x_i(t)$	Net load (in req/s) of node i at time t
λ_i	Traffic to node i (in reqs/)
μ_i	Service rate of node i (in reqs/s)
ρ_i	Utilization rate of node i (defined as λ_i/μ_i)
K_i	Maximum queue length for node i
$l_i(t)$	Service latency of node i at time t
$l_{a_i}(t)$	Average service latency between node i and its neighbours at time t
$m_{ij}(t)$	Percentage (over λ_i) of tasks forwarded from node i to node j at time t
$\bar{m}_{ij}(t)$	Percentage (over $x_i(t)$) of tasks forwarded from node i to node j at time t
$\psi_{ij}(\hat{t})$	Amount of traffic (in req/s) that node i forwards to node j at time t
$\Phi(\mathbf{x})$	Beckmann potential
$V(\mathbf{x})$	Candidate Lyapunov function
	Adaptive Heuristic (Algorithm 1)
M	Matrix of migration ratios
m_{ij}	Current percentage (of λ_i) of tasks forwarded from node i to node j
α	Step size
ϵ	Tolerance on the average latency for which the algorithm stops the updating of the migration ratios (balance zone)
T	Round duration
	Trajectories and Experiments
d_t	Average service latency
d_a	Average service latency among all the nodes

- Simulation results of the proposed heuristic algorithm;
- Results of the implementation of the proposed algorithm in a testbed of Raspberry Pis which shows the efficacy of the solution even in a real setting.

The rest of this paper is organised as follows. In Section II we present some related work, then in Section III we define the system model by describing its dynamic relying on differential equations. This model does not give us an algorithm for finding the solution in a real deployment, and therefore we propose a heuristic in Section IV that is tested in a simulator. Then in Section V we show the results of the proposed heuristic in a real environment and finally, the conclusions will be drawn in Section VI. In Table I we listed all the symbols used in the paper.

II. Related Work

The main area in which this work lies is the problem of load balancing in Edge and Fog computing [2], [3], [4], [5]. In our work, we design a load balancing algorithm that is QoS oriented, which targets the delay that users experience when using the deployed application. Similar works, like [6] propose the (OLBA) framework, which takes into account turn-around time and service delay and relies on Particle Swarm Optimization (PSO) for finding the best load balancing strategy but the approach is not fully decentralized, the same approach is followed by [7]. Then, Tripathy et al. in [8] focus on the QoS parameters but in a smart city setting and a smart allocation scheme is performed through a genetic algorithm. However, the approach is not "online", and the scheduling decision is not taken for every task. More technological approaches instead, like the one proposed in [9], design algorithms specifically targeting well-known frameworks like Kubernetes. In that case, the authors propose a proxy-based approach that periodically monitors the pods' state, and according to the load, it forwards the requests to balance it; however, the approach does not consider node heterogeneity which can have the same load but generates different service latency. Similarly, Singh et al. in [10] propose a container-as-a-service (CaaS) load balancing strategy that is focused on energy efficiency, however, the approach is based on two steps service level agreement, while our tries to use only one, moreover the results are only provided in simulations. A game theory-based approach is proposed by [11], however, no simulation or real implementation results are provided. Sthapit et al, in [12] propose a modelling of Edge computing layer as a set of queues and design a load balancing strategy which targets the job completion rate and the energy consumption, however, only simulation results are provided, like in [13].

A set of works, instead, focus on healthcare [14] and the "internet of healthcare things" [15]. For example, [16] proposes a load balancing framework which is able to avoid any failure in responsiveness and [17] which targets a smart city. Both approaches focus on the quality of service but they do not directly target the service latency, which is critical when having heterogeneous computing nodes.

By introducing even the Cloud layer [18] we increase the computation capability, although the cloud is not used in this work, we can still refer to the load balancing strategies offered by different works. For example, [19] proposes an Edge-Fog-Cloud algorithm for distributing the traffic in all of the three layers but the focus is not the latency optimisation, [1] provides a model based on queuing theory, [20] studies a load balancing approach for the Fog-Cloud environment classifying requests in real-time, important and time-tolerant but again the approach is not focused on latency levelling, then [21] proposes a scheduling approach based on blockchain and [22] a strategy to cope with failures by using Software-Defined Networks (SDN).

The technique used for modelling the system comes from the control systems theory [23]. In particular, the literature on the Wardrop equilibrium to which we prove the convergence of the model is quite extensive.

In conclusion, the last set of works worth mentioning focuses on load balancing by using intelligent approaches like reinforcement learning [24], [25], [26]. The heuristic proposed in this work (Section 1) is not explicitly using reinforcement learning but it follows a strategy that mimics a learning process since the migration ratios are continuously adapted to meet a goal by using a learning rate α.

III. Performance Model

In our model, we suppose to have a set $\mathcal{N}$ of nodes whose network topology is described by the adjacency matrix A. In

particular, given any two nodes i and j, they can communicate only if $a_{ij} = a_{ji} = 1$ since we always suppose that the communication between nodes is bi-directional. Each node i receives a fixed traffic rate of λ_i requests/s from the underlying clients, and it is able to execute μ_i req/s. Moreover, a node i is able to forward part of its load (in terms of requests/s) to a given neighbouring node j, and we do not consider the network communication latencies. We call the percentage of forwarded requests from node i to j the "migration ratio", and it is expressed as m_{ij}.

We now want to model the system mathematically and, to do so, we define which is the total load of a node i over time. We call this function $x_i(t)$ since it models the state of node i in a given time t:

$$x_i(t) = \lambda_i - \sum_{j \in V} a_{ij}\psi_{ij}(t) + \sum_{j \in V} a_{ji}\psi_{ji}(t) \tag{1}$$

where the initial condition, at $t = 0$, since $\psi_{ij}(0) = 0\ \forall i, j$ is

$$x_i(0) = \lambda_i \qquad \forall\, i \tag{2}$$

Equation (1) can be interpreted as follows. The total net load that a node i sees over time is made up of three addends. The first (i) component represents the constant traffic coming by the clients that are directly connected to the node, and it is called λ_i. The second (ii) addend is subtracted since it represents the sum of the traffic that the node i forwards to any neighbouring node j (for which $a_{ij} \neq 0$) that is $\psi_{ij}(t)$. However, (iii) even neighbouring nodes may also decide to forward part of their traffic to i, and this part, $\psi_{ji}(t)$ again summed for all the neighbours, is added to the total load of the node $x_i(t)$ and represents the last addend. As it will be now clear, for any node i, the functions $\psi_{ij}(t)$ describe the traffic (in terms of req/s) that node i forwards to the neighbouring j at any time t and they are our unknowns. By knowing the $\psi_{ij}(t)$, we will then need to find a time t^* where $\psi_{ij}(t) = \psi_{ij}(t^*),\ \forall i, j, t > t^*$ and the values $\psi_{ij}(t^*)\ \forall i, j$ will be the final traffic that each node will need to forward to each neighbour to reach the final goal. However, for applying the solution to a real setup, it is not enough to know which are the forwarded rates $\psi_{ij}(t)$. Indeed, we need to find the migration ratio from a node i to a node j, expressed in the percentage of the clients' incoming load (λ_i) that a node must forward. This is because, for simplicity and for having a term of comparison with further simulations and experimental tests, we hypothesise that a node can only forward the traffic that is coming from the clients and not the one that is coming from the neighbours. Therefore we impose that the total traffic from node i to node j that is $\sum_j \psi_{ij}(t)$ must be lower or equal to λ_i. When this is true, we call this strategy "single-hop", and we can derive the migration ratio from a node i to node j as the portion of the incoming traffic from the clients to i:

$$m_{ij}(t) = \frac{\psi_{ij}(t)}{\lambda_i} \tag{3}$$

At this point, we need to model this final goal: the levelling of latency among all the nodes. For finding the functions $m_{ij}(t)$, instead of trying to define them directly, it is easier to describe their variation over time, and for this reason, we calculate the derivative with respect to the time of Equation 1 that is:

$$\dot{x}_i(t) = -\sum_{j \in \mathcal{N}} a_{ij}\dot{\psi}_{ij}(t) + \sum_{j \in \mathcal{N}} a_{ji}\dot{\psi}_{ji}(t) \tag{4}$$

The variation over time of the forwarded traffic from i to j can be defined as:

$$\dot{\psi}_{ij}(t) = x_i(t)\dot{\tilde{m}}_{ij}(t) \tag{5}$$

That is the product between the load of the node i, $x_i(t)$, and the variation on the percentage of this load to be forwarded to j, that we call $\dot{\tilde{m}}_{ij}(t)$ to differentiate them from the effective migration ratio $m_{ij}(t)$. We can then rewrite the derivative as:

$$\dot{x}_i(t) = -\sum_{j \in \mathcal{N}} a_{ij}x_i(t)\dot{\tilde{m}}_{ij}(t) + \sum_{j \in \mathcal{N}} a_{ji}x_j(t)\dot{\tilde{m}}_{ji}(t) \tag{6}$$

Equation (6) describes the dynamic of the state of node i, that is how the load that every node i sees at time t changes over time. The formulation can be repeated for every node. Thus we have a system of $|\mathcal{N}|$ Ordinary Differential Equations (O.D.E.). Before solving the system, we need to define the functions $\dot{\tilde{m}}_{ij}(t)$ that are still unknown, but we remind that the solution to the system will allow us to know the original $m_{ij}(t)$.

Basically, we define the $\dot{\tilde{m}}_{ij}(t)$ as the multiplication of three factors logically derived from the fact that our objective is that, in every node, every task must have the same duration, and therefore the average service latency of each node must be the same. Moreover, we need to keep in mind two essential behaviours of the entire system: (i) when a node i migrates a portion of the incoming traffic to another node j, the node i will see its average service latency decrease, while in the node j the average task service latency will increase. This is because the service latency function is a monotonically increasing function with respect to the load of a node. In our case, we suppose, for simplicity, that nodes can be modelled as M/M/1/K queues and the service latency at time t of node i can be expressed as (given $\rho_i(t) = x_i(t)/\mu_i$):

$$l_i(t) = \frac{1 - (K_i + 1)\rho_i(t)^{K_i} + K_i\rho_i(t)^{(K_i+1)}}{\mu_i(1 - \rho_i(t))(1 - \rho_i(t)^{K_i})} \tag{7}$$

Then (ii) the average delay between neighbours nodes plays a crucial role because the average service latency of a given node can be higher or lower than the average, and trying to level them to the average proved to be the key strategy to solving the problem. But how can we level them to the average? There are three sub-strategies that we need to adopt to reach the goal, and they concretise into three factors:

1) the tasks migration must be performed only if the delay of the current node i, $l_i(t)$, is greater than the average delay between itself and its neighbours, called l_{a_i}, for this reason, the first factor is:

$$\dot{\tilde{m}}^{\alpha}_{ij}(t) = \max\left[0, \frac{l_i(t) - l_{a_i}(t)}{l_i(t)}\right]. \tag{8}$$

2) the tasks migration must be performed only if the delay of the current node i, $l_i(t)$, is greater than the delay of its

neighbour j, $l_j(t)$, and therefore:

$$\dot{\bar{m}}_{ij}^{\beta}(t) = \max\left[0, \frac{l_i(t) - l_j(t)}{l_{h_i}(t)}\right]. \tag{9}$$

3) the tasks migration must be performed only if the delay of the neighbour node j, $l_j(t)$, is lesser than the average delay between node i and itself, and therefore:

$$\dot{\bar{m}}_{ij}^{\gamma}(t) = \max\left[0, \frac{l_{a_i}(t) - l_j(t)}{l_{k_i}(t)}\right]. \tag{10}$$

The final dynamic of the migration ratios is, therefore

$$\dot{\bar{m}}_{ij}(t) = \dot{\bar{m}}_{ij}^{\alpha}(t) \cdot \dot{\bar{m}}_{ij}^{\beta}(t) \cdot \dot{\bar{m}}_{ij}^{\gamma}(t). \tag{11}$$

and the idea behind the formulation is that the dynamic of the state $\dot{x}(t)$ stops when at least one of them becomes zero, both for the received load and the forwarded one. As already mentioned, the $l_{a_i}(t)$ is the average delay between the current node i and its neighbours:

$$l_{a_i}(t) = \frac{l_i(t) + \sum_{j \in V; i \neq j} a_{ij} l_j(t)}{1 + \sum_{j \in V} a_{ij}} \tag{12}$$

Finally, $l_{h_i}(t)$ and $l_{k_i}(t)$ are the summations of the differences over time:

$$l_{h_i}(t) = \max\left[0, \sum_{j \in V} l_i(t) - l_j(t)\right] \tag{13}$$

$$l_{k_i}(t) = \max\left[0, \sum_{j \in V} l_{a_i}(t) - l_j(t)\right] \tag{14}$$

We will resort to numerical calculus to find the time trajectories of the system of non-linear ODE described in (4) with initial conditions $x_i(0) = \lambda_i$, $\forall i$. The numerical solution describes the trajectory of the state $x_i(t)$ of every node i, but our objective is to find the final migration ratios that, if wired within a real system, allow us to reach the latency-levelling goal. For doing this, from the definition of the effective migration ratios (3), we first need to compute $\psi_i j(t)$ as:

$$\psi_{ij}(t) = \int_0^t \dot{\psi}_{ij}(u)du = \int_0^t x_i(u)\dot{\bar{m}}_{ij}(u)du \tag{15}$$

Now, the solution to the system at (4) does not distinguish between single and multi-hop strategies. However, we can easily understand if the solution can be implemented in a single-hop manner by taking into consideration that if a node i is obliged to forward requests coming from the neighbours, then necessarily, there will exist a time $\hat{t}$ for which

$$\sum_j \psi_{ij}(\hat{t}) > \lambda_i \tag{16}$$

By using (3) we have that the total effective migration ratio for a node i will be greater than one

$$\sum_j \lambda_i m_{ij}(\hat{t}) > \lambda_i \tag{17}$$

$$\sum_j m_{ij}(\hat{t}) > 1 \tag{18}$$

This means that if (18) does not hold for every node in the system, then the final solution at convergence can be implemented in a single-hop manner by applying the forwarding policy for which nodes can only forward the requests from the clients. Otherwise, we can still practically use the solution that we find at time $\hat{t}$ that is single-hop, but if $\hat{t} < t^*$, then we probably do not have levelled the latency across all the nodes (see Topology C at Section III-B).

A. *Latency-Levelling Property*

We now prove that when the trajectories of the solution of the system at Equation 6 converge, then the service latency is aligned to the same value in every node. Then we prove that the same system asymptotically converges to a unique Wardrop equilibrium.

Lemma III.1. If the solution's trajectories of the O.D.E. system at Equation 6 converges, i.e. $\exists\, t^*$ s.t. $\dot{x}_i(t) = 0\ \forall t > t^*, \forall i$ then all the nodes have the same service latency, i.e. $l_0(t) = l_1(t) = \ldots = l_i(t)\ \forall i$ and this latency is the average latency $l_{a_i}(t^*)$ among all the neighbours of each node i at time t^*.

Proof. We can prove the theorem by contradiction. Suppose that the system solution converged at t^* but there exists one node i that has not the same service latency as the other nodes, i.e. $l_i(t) \neq l_j(t), \forall j \neq i, \forall t > t^*$. We can distinguish two possible cases, for any $t > t^*$:

1) $l_i(t) > l_{a_i}$, i.e. the service latency of node i is *higher* than the average latency between i and its neighbours, we point out that every other neighbouring node's latency can be higher, equal or lower than the average latency but at least one node must have the latency below the average, for the property of the average of a set of numbers. From this fact, we have that the $\dot{\bar{m}}_{ij}^{\alpha}(t) \neq 0$ by definition, $\dot{\bar{m}}_{ij}^{\beta}(t) \neq 0$ and $\dot{\bar{m}}_{ij}^{\gamma}(t) \neq 0$ because there exists at least one node with average service latency below the average and the same node's latency is also lower than the latency of node i. This means that from Equation 6 the negative part is not zero, the positive part instead is zero since i is the only node with latency higher than the average it will not receive traffic from any neighbour. Therefore we showed that $\dot{x}(t) \neq 0$ for some $t > t^*$, and this is a contradiction ※;
2) $l_i(t) < l_{a_i}$, i.e. the service latency of node i is *lower* than the average latency between i and its neighbours. As in the case 1, if the node i's latency is below than the average latency then there exists at least one neighbour j whose latency is higher than the average. The consequences are exactly the ones of case 1, and we proved the contradiction ※;

From these two cases emerges that the only possible case is that $l_i(t) = l_{a_i}$ and no other node can have a service latency that is higher or lower than the average l_{a_i}.

Definition III.1. Under flow rate vector $\boldsymbol{\lambda} = [\lambda^i]_{i \in \mathcal{I}} \in \Lambda$, the feasible state space is defined as

$$\mathcal{X} := \left\{ \mathbf{x} = [x_i]_{i \in \mathcal{N}} \middle| x_i \geq 0, \sum_{i \in \mathcal{N}} x_i = \sum_{i \in \mathcal{N}} \lambda_i \right\}. \tag{19}$$

The flow vectors $\mathbf{x} \in \mathcal{X}$ are the feasible flow vectors.

Assumption III.1. The latency functions $l_i : [0, \bar{\lambda}] \to \mathbb{R}_{\geq 0}$, for all $i \in \mathcal{N}$, are Lipschitz continuous, strictly increasing over the interval $[0, \bar{\lambda}]$, where $\bar{\lambda} := \sum_{i \in \mathcal{N}} \lambda_i$, and such that $l_i(0) = 0$.

In the Wardrop literature, a *stable* flow is a state in which no agent may improve its strategy unilaterally. This network equilibrium takes the name of Wardrop equilibrium.

Definition III.2. ([27]) A flow vector $\mathbf{x} \in \mathcal{X}$ is at a Wardrop equilibrium under flow rate $\boldsymbol{\lambda}$ if it holds that $l_i(\mathbf{x}) \leq l_j(\mathbf{x})$ for all $i \in \mathcal{N}$ such that $x_i > 0$ and $\forall j \in \mathcal{N}$.

One may interpret the Wardrop equilibria as the set of states in which the latency of all the loaded nodes (i.e., the nodes $i \in \mathcal{N}$ such that $x_i > 0$) are equalised.

As customary in Wardrop theory, we resort to the Beckmann potential [28], defined as

$$\Phi(\mathbf{x}) = \sum_{i \in \mathcal{N}} \int_0^{x_i} l_i(\xi) d\xi, \tag{20}$$

whose properties are summarised by by Property III.1 below.

Property III.1. Under Assumption III.1, the Beckmann potential (20) is continuous and the following properties hold:

1) there exists a unique feasible flow, denoted by $\mathbf{w} \in \mathcal{X}$, that minimises $\Phi(\mathbf{x})$, with $\mathbf{x} \in \mathcal{X}$;
2) correspondingly, there exist a unique, positive minimum of $\Phi(\mathbf{x})$, denoted by $\Phi_{min} := \Phi(\mathbf{w}) > 0$;
3) $\mathbf{w}$ is the unique Wardrop equilibrium.

Thus, under Assumption III.1 a unique equilibrium point exists, where the latencies of all the nodes are equalised. Furthermore, given that that the l_i's are monotonically increasing and null in zero, all the nodes are loaded at the equilibrium, i.e., $x_i > 0$ for all $i \in \mathcal{N}$ if $\mathbf{x} = \mathbf{w}$.

For the convergence proof, the Beckmann potential (20) is used to build the candidate Lyapunov function $V : \mathcal{X} \to \mathbb{R}_{\geq 0}$ as the continuously differentiable function

$$V(\mathbf{x}) := \Phi(\mathbf{x}) - \Phi_{\min}. \tag{21}$$

The following Theorem holds.

Theorem III.1. Under Assumption III.1, the trajectories of the nonlinear system (7), (8)–(11) with initial condition $x(0) \in \mathcal{X}$ asymptotically converge to the Wardrop equilibrium $\mathbf{w}$.

Proof. Firstly, we show that $\mathcal{X}$ is a positive invariant set for the nonlinear system (7), (8)–(11), i.e., that if $x(0) \in \mathcal{X}$ it holds that $x(t) \in \mathcal{X}$ for all $t > 0$:

1) It follows from (4) and by the symmetry of the adjacency matrix A that $\sum_{i \in \mathcal{N}} \dot{x}_i(t) = 0$ and, therefore, that $\sum_{i \in \mathcal{N}} x_i(t) = \sum_{i \in \mathcal{N}} x_i(0) = \lambda$, for all $\mathbf{x}(0) \in \mathcal{X}$.
2) (4), (8)–(11) yield that, for all $i \in \mathcal{N}$, $\dot{x}_i(t) \geq 0$ if $x_i(t) = 0$; given that $x_i(0) \geq 0$, it holds that $x_i(t) \geq 0$.

Relying on standard Lyapunov theory, we prove now that the Function (21) is a Lyapunov function for the system (7), (8)–(11) by showing that V and $-\dot{V}$ are positive definite with respect to the equilibrium point $\mathbf{w}$, which is unique by Property III.1.

(V *positive definite*)

Given Property III.1 and (21), it holds that $V(\mathbf{x}) = 0$ if $\mathbf{x} = \mathbf{w}$ and $V(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathcal{X} \setminus \{\mathbf{w}\}$.

($-\dot{V}$ *positive definite*)

By equations (20) and (21), $\dot{V}$ is written as

$$\dot{V}(\mathbf{x}(t)) = \dot{\Phi}(\mathbf{x}(t)) = \frac{d}{dt}\Phi(\mathbf{x}(t)) = \sum_{i \in \mathcal{N}} \dot{x}_i(t) l_i(x_i(t)). \tag{22}$$

From (4) and (22), it follows that

$$\begin{aligned} -\dot{V}(\mathbf{x}(t)) &= \sum_{i \in \mathcal{N}} \left(\sum_{j \in \mathcal{N}} a_{ij} x_i(t) \dot{\tilde{m}}_{ij}(t) \right. \\ &\quad \left. - \sum_{j \in \mathcal{N}} a_{ji} x_j(t) \dot{\tilde{m}}_{ji}(t) \right) l_i(x_i(t)) \\ &= \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} a_{ij} x_i(t) \dot{\tilde{m}}_{ij}(t) l_i(x_i(t)) \\ &\quad - \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} a_{ji} x_j(t) \dot{\tilde{m}}_{ji}(t) l_i(x_i(t)) \\ &= \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} a_{ij} x_i(t) \dot{\tilde{m}}_{ij}(t) \left(l_i(x_i(t)) - l_j(x_j(t)) \right). \end{aligned} \tag{23}$$

Each term of the last summation is either positive or null, given that the following relations hold:

- If $l_i(x_i(t)) \leq l_j(x_j(t))$ Lemma III.1 yields $\dot{\tilde{m}}_{ij}(t) = 0$ and, therefore, the term is null.
- If $l_i(x_i(t)) > l_j(x_j(t))$, equations III.1 yields $\dot{\tilde{m}}_{ij}(t) \leq 0$ and, given that a_{ij} and $x_i(t)$ are non-negative, the term is either positive or null.

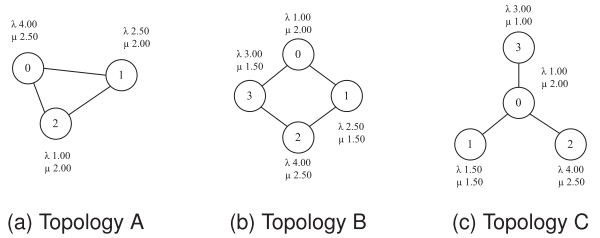
The relations above show that $\dot{V}(\mathbf{x}) = 0$ if $\mathbf{x} = \mathbf{w}$ since, by Definition (III.2), it holds that $l_j(x_j) = l_i(x_i)$ for all $i, j \in \mathcal{N}$.

To conclude that $-\dot{V}$ is positive definite, it remains to show that there is at least one positive term when the system is not at the Wardrop equilibrium. If $\mathbf{x} \in \mathcal{X} \setminus \{\mathbf{w}\}$, by Definition III.2 there exists at least one pair $i', j' \in \mathcal{N}$ such that $l_{i'}(x_{i'}) > l_{j'}(x_{j'}(t))$. We consider, without loss of generality, that i' and j' are neighbouring nodes: in fact, if i' and j' were not neighbouring nodes, since the graph is connected, there would a path between the nodes and, therefore, at least one pair of consecutive nodes in the path with different latency. Being $i', j' \in \mathcal{N}_{i'}$ two nodes with different latency, there exists a pair of nodes $i, j \in \mathcal{N}_{i'}$ (possibly i' and/or j' themselves) such that $l_i(x_i) > l_{a_{i'}}(\mathbf{x}) > l_j(x_j)$. Therefore, Lemma III.1 yields $\dot{\tilde{m}}_{ij} > 0$. Finally, given that $l_i(x_i) > l_j(x_j) \geq 0$, Assumption III.1 yields $x_i > 0$.

Summarising: if $\mathbf{x} \in \mathcal{X} \setminus \{\mathbf{w}\}$, there exists a pair of nodes $i, j \in \mathcal{N}$ such that $a_{ij} = 1$, $x_i > 0$, $\dot{\tilde{m}}_{ij} > 0$ and $l_i(x_i) > l_j(x_j)$, i.e., there exists at least a positive term in the last summation of (23). □

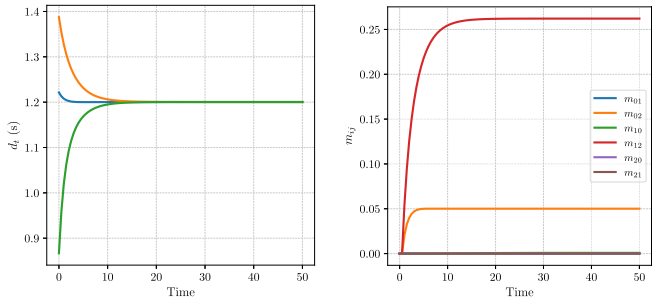
B. Trajectories and Topologies

We will now explore some configurations of nodes and parameters that we will reuse later in the simulations and in the experimental setting, the general idea is to show how this model can predict quite well the behaviour of a real system. The crucial point for the results to match is the alignment of the service latency, but the alignment value and the migration ratios may differ, as will be clearer later. In this section, we study the



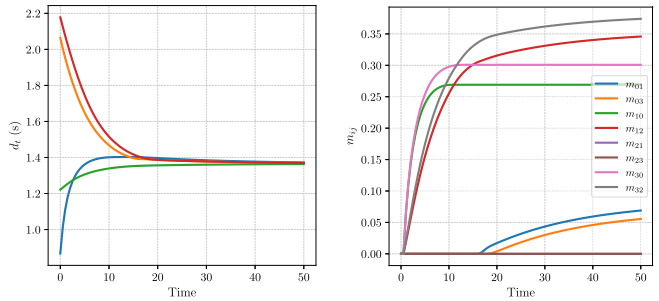
(a) Topology A (b) Topology B (c) Topology C

Fig. 1. The nodes topology and parameters configuration used across the mathematical model, the simulations and the final experimental setting.



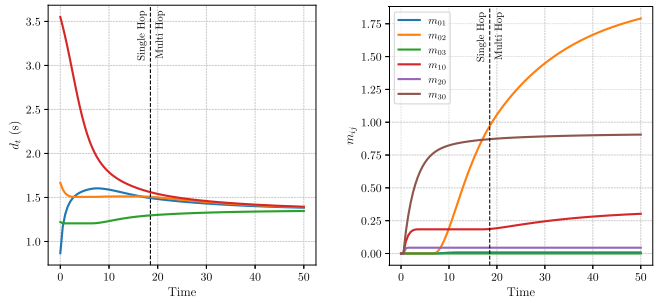
(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 2. Trajectories of the average latency d_t and the migration ratios m_{ij} for the three nodes described by Topology A [Fig. 1(a)].



(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 3. Trajectories of the average latency d_t and the migration ratios m_{ij} for the four nodes described by Topology B [Fig. 1(b)].



(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 4. Trajectories of the average latency d_t and and the migration ratios m_{ij} for the four nodes described by Topology C [Fig. 1(c)].

behaviour of the latency over time $d_t(t)$, computed by using the (7) and the migration ratios $m_{ij}(t)$ computed by using (3).

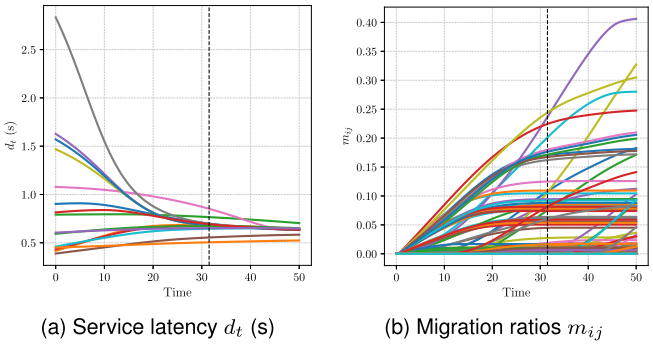
We tested different network topologies, the first three are shown in Fig. 1. These small topologies are taken into consideration because it is easy to have a direct comparison with the behaviour shown in simulations and in a real deployment. Finally, we tested a fully connected topology with 15 nodes.

Topology A Fig. 1(a) is composed of three nodes arranged in a fully connected graph, the Fig. 2(a) shows the trajectories of the latency and the migrations ratios m_{ij} of the nodes. As we can observe, after the transient the system reaches the steady-state at about $t = 15$ where the latencies are levelled at 1.2 s. From the migration ratio, Fig. 2(b), we can observe that Node 1 gives 25% of its load λ_1 to Node 2 since it has the higher service latency at $t = 0$ and part of its load is forwarded to the node that is below the average service latency, that is Node 2. Node 2 only has to receive load while Node 0 and Node 1 have to lose their load in order to balance the service latency, indeed even Node 0 forward exactly the 5% of its load to slightly reduce the service latency.

Topology B Fig. 1(b) comprehends four nodes connected as a ring, the Fig. 1(b) shows the numerical trajectories of the performance parameters. Each node, from 0 to 3, starts with service latency, respectively, 0.86 s, 2.06 s, 1.22 s and 2.18 s and the end of the transient Fig. 3(a) is levelled to 1.38 s. At the steady state and we can observe how Fig. 3(b) Node 3 forwards about the 65% of its traffic to nodes 0 and 2 for lowering the latency, the same is done by Node 1, which forwards a total of about 60% of its load to Nodes 2 and 1, then Node 2 does not forward tasks because already close to the average latency while starting from $t = 10$ Node 0 starts to forward tasks to its neighbours up to the 10%. This means that Node 1 must give back part of the load to Nodes 1 and 3, but these nodes already forwarded part of their load to Node 0, this behaviour is justified by the fact that the derivative of migration ratios functions $\dot{m}_{ij}(t)$ are always positive. Therefore the only way to diminish them is to make a node to give back the load to the sender.

Topology C Fig. 1(c) is a star topology and includes four nodes, but this particular configuration of the nodes is more challenging because one single node is connected to all the others while the others are only connected to the same node, and therefore the node at the centre can be overwhelmed by the load of the others. However, the model converges to a levelled latency of 1.4 s Fig. 4(a) but the solution that is reached is actually not achievable because the condition expressed at Equation 18 is no more respected Fig. 4(b)), since the model is unconstrained. This does not mean that we cannot use the solution. Indeed, it is sufficient to consider the transient as long as the condition is still met, i.e. at $t = 26$ and consider the migration ratios there. What is clear is that the exact levelling of the latency is not feasible, but considering the solution, at $t = 26$ we still reached a point in which the latencies are closer, even if they do not exactly match. In particular, we recall that in this solution, the node m_{02} is required to forward all of its traffic λ_0 and execute only the traffic forwarded by the other nodes.

The last topology that we tested comprehends instead 15 nodes in a fully connected topology with $1 \leq \lambda_i \leq 4, 1 \leq \mu_i \leq 4$ and $2 \leq K_i \leq 6$. All of these parameters are picked at random,



(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 5. Trajectories of the average latency d_t and the migration ratios m_{ij} for 15 nodes in a fully connected topology.

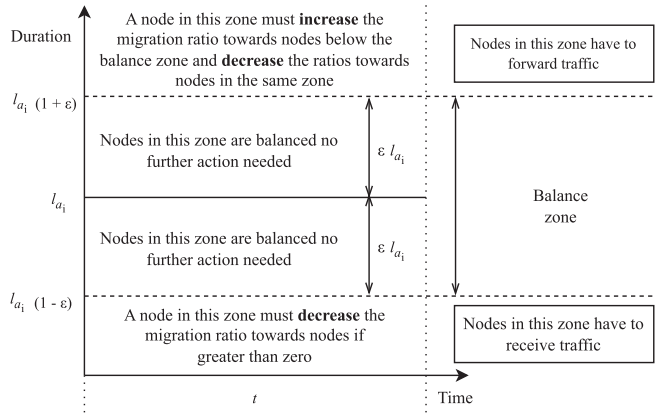


Fig. 6. Representation of the logic behind the adaptive heuristic for a node i in a given time t. We suppose the average delay l_{a_i} between the node i and its neighbours to be fixed during an instant time t.

but the purpose of this is to understand how the system behaves with many nodes. We used SageMath[1] Python ODE solver to derive the trajectories up to $t = 100$ with the numeric solver Runga-Kutta-Felhberg on a Ryzen 9 5800X processor. Fig. 5(a) shows the behaviour of the latency for all the nodes, and as we can see, the system reduces their variance, but again we need to cut the solution at time $t = 31$ because $\sum_j m_{ij}(t) \geq 1$ for some i when $t \geq 31$ Fig. 5(b).

This last result shows how the model scales with the number of nodes, however, we do not envision modelling a system of more than 20 Edge or Fog nodes because aligning the latency in a very large set of nodes may not be the best strategy for balancing the load. As we can see, some nodes can be obliged to forward all of their traffic, and if the parameters λ_i and μ_i are particularly different, then it would not be possible to level the latency without counting the difficulties of implementing the algorithm in the real world where the network latency have a significant impact. Instead, it is more efficient to create groups of a maximum of 20 nodes and try levelling the latency within the groups, these groups can, for example, represent neighbourhoods of a smart city.

IV. Adaptive Heuristic

We now want to effectively implement a strategy for levelling the latency among the nodes. The mathematical model tells us what are, at steady state, the migration ratios m_{ij} $\forall i, j$ but calculating them requires finding the trajectories of the model. Moreover, there are other 3 points that motivate the design of an algorithm. First of all, (1) the mathematical model assumes that we know the state of every node but in the real world, we want to have a fully decentralised approach, each node should be able to see the only state of its neighbours and tune the migration ratios accordingly, also that state must be explicitly requested when needed. Then (2) real nodes may be subject to variation in load conditions over time, thus the algorithm should react and re-tune the migration ratios to cope with the changes. As the last point, (3) the model does not take into account the communication latency that exists between the nodes. Therefore, we now propose an adaptive strategy which follows a heuristic approach to find the most suitable set of migration ratios for every node in such a way the latency is made equal when it is possible or at least closer when it is not feasible.

Fig. 6 summarises the logic behind the heuristic. Firstly, we suppose to divide the time into rounds of T seconds each. The Algorithm 1 is run every time a round ends and has as a final objective the one of modifying the migration ratios when it is needed. We also divide the algorithm into steps for describing the rationale behind its design. The input that it takes comprehends the index of the current node i in which the algorithm is executed (we remind that the algorithm is fully distributed, there is no central entity or coordinator), the step size α, the set of nodes $\mathcal{N}$, the vector of migration ratios $\vec{M}_i$ which describe the percentage of tasks that is forwarded to each (neighbour) node, percentage on the average latency that defines the balancing zone ϵ and the incidence vector for node i that is $\mathcal{I}_i$ and describes which are the neighbours of the current node. Suppose that the round time T just elapsed, and the algorithm does the following:

1) first of all, the node computes the average latency between itself and all the neighbours, moreover, it computes the upper and lower average limits by multiplying the average latency by $1 \pm \epsilon$, these limits allow us to relax the constraint that each node must exactly match the latency of each other, which in real scenarios is very unlikely due to the arrivals' distribution. As the last step, it is also computed the sum of all the migration ratios, which cannot exceed 1.0;
2) once the average is known, we proceed to the adjusting of the migration ratios; the first check that we perform is to see if the current node is below the average and if it is migrating tasks to other nodes. Indeed, if this happens, then it means that the node is forwarding too much traffic to the others. We remind from the mathematical model, that the strategy for making the algorithm work is that a node can only receive or give traffic to others at the same time, and, in general, only the nodes that are above the average must forward tasks to the ones that are below. Thus, a node that is below the average and it is giving

1.https://www.sagemath.org/

traffic to others must reduce the ratios in such a way its average returns the balance zone ($d_{a_i} \pm \epsilon$). This is what the algorithm does in during this step for all the neighbours nodes by previously checking if the ratio given to the node does not reach negative numbers and this is done by using the auxiliary functions described in Algorithm 2. If the adjustment is done, the function returns with no further steps;

3) at this point, we check if the average latency of the current node is below the high level of the average zone, because if this is true then it means that the node latency is in the average zone, then no further action is needed;
4) if we reach this step, then the node's latency is out of balance, i.e. it is above the high level of the zone, then we need to adjust the migration ratios for every neighbour node, but we can distinguish the following two cases:
1) if the average latency of neighbour node j is above the balance zone, then we reduce the migration ratio towards it of the step size α since it means that we are forwarding too much traffic;
2) if the average latency of the neighbour node is below the balance zone, then we increase the migration ratio towards it of the step size α, this will cause our latency to be reduced and its one to increase, approaching the balancing zone.

A. Simulations Results

We now show some results of the proposed algorithm in a discrete event simulator written in Python by using the library "Simpy"[2] and published as open source.[3] We will use the same topologies and parameters (Fig. 1) used in for computing the trajectories of the mathematical model in order to have terms of comparison.

In the simulator, we again assume no communication delays between the nodes and the same nodes are modelled as M/M/1/K queues since the objective of simulations is to understand if the migration ratios found by the heuristic match the model. All the tests are done with the simulator to use a round time $T = 60\tilde{}s$ and the behaviour of the average latency are filtered with a Savitzy-Golay filter with window size 20 and polynomial degree of 4. Moreover, the balance zone uses $\epsilon = 0.05$, the step size $\alpha = 0.01$ and $K_i = 4 \; \forall \; i$. A peculiar characteristic of the simulator is that the average latency is computed as the average of the last 10 rounds, this is done in order to stabilize the curves, otherwise due to the exponential distribution of the inter-arrival times and of the execution times the average latency may be subjected to significant variations.

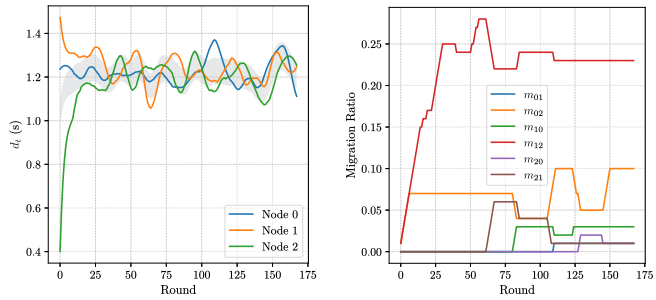
Fig. 7 shows the results of the simulations of Topology A. First of all, we can observe how after 25 rounds, the average latency starts to stabilize at about 1.2 s Fig. 7(a), we have highlighted in grey the balance zone that is the average delay $d_a \pm \epsilon$ and in the chart the average it is computed across all of the nodes. We can notice how the latency result is perfectly matching the model compared to Fig. 2(a), the fluctuations around the average is due to the exponential inter-arrival times and execution times. For levelling the latency the migration ratios found by the algorithm are represented in Fig. 7(b). In particular, we can observe that m_{12} stabilizes at around 0.24 and m_{02} at around 0.07 while the others are less than 0.03. Again these result matches the ones of the model shown in Fig. 2(b), in which m_{12} and m_{02} stabilizes at 0.26 and 0.05 respectively, while the others are set to 0.

Topology B results are shown in Fig. 8. As far as regards the average service latency Fig. 8(a) we can observe how it stabilizes at about 1.4 s which is in line with the mathematical model shown Fig. 3(a). The same holds for the migrations ratios, for example, the Node 0 gives 5% of the λ_0 to its two neighbours respectively that match the model, Node 1 gives about 20% of its traffic to Node 0 but the model 26% and about 45% to Node 2 while the

Algorithm 1: Adaptive Heuristic for Levelling Latencies.

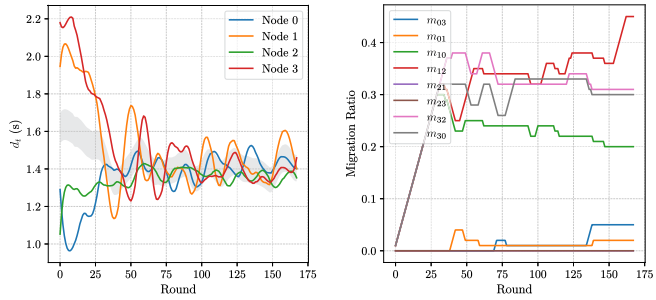
Require: i, α, $\mathcal{N}$, $\vec{M}_i$, ϵ, $\mathcal{I}_i$
currentNode $\leftarrow \mathcal{N}[i]$
[1. Compute the average latency among all the neighbour nodes]
averageLatency $\leftarrow$ currentNode.getLatency()
numberOfNeighbours $\leftarrow 0$
for all j in $|\mathcal{N}|$ **and** $\mathcal{I}_{ij} \neq 0$ [*Loop over the neighbours*] **do**
  averageLatency $\leftarrow$ node.getLatency() **and**
  numberOfNeighbours $\leftarrow$ numberOfNeighbours + 1
end for
averageLatency $\leftarrow$ averageLatency / numberOfNeighbours
averageLatencyLow $\leftarrow$ averageLatency $\cdot\ (1.0 + \epsilon)$
averageLatencyHigh $\leftarrow$ averageLatency $\cdot\ (1.0 - \epsilon)$
totalRatiosGiven $\leftarrow \sum_j m_{ij}$
[*2. If under average and migrating, then reduce migration*]
if currentNode.getLatency() $\leq$ averageLatencyLow **and**
totalRatiosGiven > 0 **then**
  for all j in $|\mathcal{N}|$ **and** $\mathcal{I}_{ij} \neq 0$ **do**
    if $\mathcal{N}[j]$.getLatency() $\geq$ averageLatencyHigh **then**
      if canBeSubtractedToNode(j, α) **and** canSubtract(α) **then**
        $m_{ij} \leftarrow m_{ij} - \alpha$
        totalRatiosGiven $\leftarrow$ totalRatiosGiven - α
      end if
    end if
  end for
  return
end if
[*3. If latency below the high zone limit, then the node is balanced*]
if currentNode.getLatency() $<$ averageLatencyHigh **then**
  return
end if
[*4. If latency greater or equal the high limit we need to migrate*]
for all j in $|\mathcal{N}|$ **and** $\mathcal{I}_{ij} \neq 0$ **do**
  [*4a. Reduce the ratio to neighbour above the average high limit*]
  if $\mathcal{N}[j]$.getLatency() $\geq$ averageLatencyHigh **then**
    if canBeSubtractedToNode(j, α) **and** canSubtract(α) **then**
      $m_{ij} \leftarrow m_{ij} - \alpha$
      totalRatiosGiven $\leftarrow$ totalRatiosGiven - α
    end if
  end if
  [*4b. Increase the ratio to neighbour below the average low limit*]
  if $\mathcal{N}[j]$.getLatency() $\leq$ averageLatencyLow **then**
    if canBeGiven(α) **then**
      $m_{ij} \leftarrow m_{ij} + \alpha$
      totalRatiosGiven $\leftarrow$ totalRatiosGiven + α
    end if
  end if
end for

2.https://pypi.org/project/simpy/
3.https://gitlab.com/gabrielepmattia/simulator-2022-tsc



(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 7. Behaviour of the average latency d_t and migration ratios for Topology A (Fig. 1(a)) in the simulated environment.



(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 8. Behaviour of the average latency d_t and migration ratios for Topology B (Fig. 1(b)) in the simulated environment.

Algorithm 2: Auxiliary Functions.

Require: i, α, $\mathcal{N}$, $\vec{M}_i$, $\bar{z}$, $\underline{z}$, $\mathcal{I}_i$, totalRatiosGiven

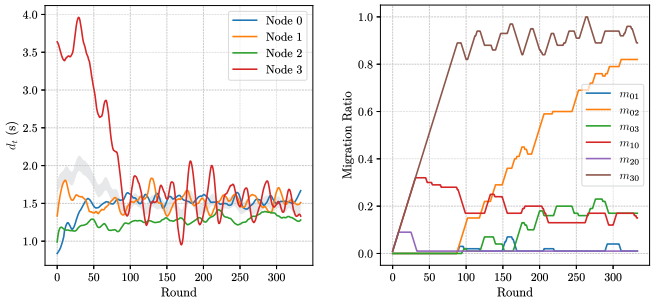
[*Check if the specified amount of ration can be given*]
def canBeGiven(alpha: float): boolean
  return totalRatiosGiven + alpha ≤ 1.0
end def

[*Check if the specified amount of ratio can be subtracted*]
def canSubtract(alpha: float)
  return totalRatiosGiven - alpha > 0.0
end def

[*Check if the specified amount of ration can be subtracted to a node*]
def canBeSubtractedToNode(j: int, alpha: float)
  return m_{ij} − alpha > 0.0
end def

model 34%. The same slight differences hold for Nodes 3 and 4 and are due to the traffic variability.

Topology C results are shown in Fig. 9. Regarding the service latency Fig. 9 we can see how it does not converge to the same value for each node, and this behaviour is the same presented in the model in Fig. 4(a) where we truncated the trajectory at $t = 26$. Indeed, the same values are obtained in the simulation, Node 0, 1 and 3 align at about 1.5 s while Node 2 stabilizes to 1.3 s because it cannot receive enough traffic from Node 0 in order to increase its latency to match 1.5 s. This does happen in the model



(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 9. Behaviour of the average latency d_t and migration ratios for Topology C (Fig. 1(c)) in the simulated environment.

after $t = 26$ but Node 0 would forward more traffic than the one that is available. Regarding instead the migration ratios, shown in Fig. 9(b), we can observe that as the latency, they match with the truncated solution of the model Fig. 4(b) with slight differences. In particular, m_{30} reaches 0.9, m_{02} reaches 0.9 while in the model 1.0, then m_{03} and m_{10} reach 0.2 respectively while in the model 0.0 and 0.2.

V. Experimental Setting

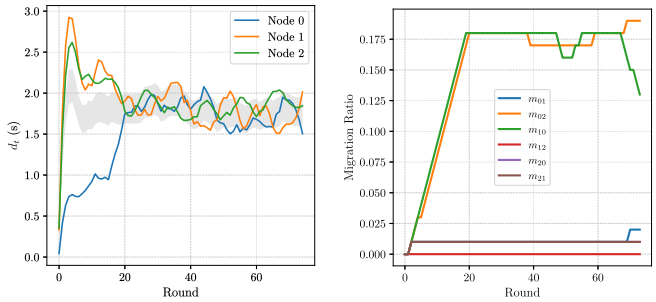
After testing the proposed adaptive heuristic in simulations, we finally implemented it in a testbed of Raspberry Pi 4[4] connected with Gigabit Ethernet to a dedicated subnet. Each node implements a Python web server based on the Flask[5] library, that once deployed with Docker, receives the traffic from a machine that acts as a traffic generator. The source of the application is published as open source[6]. The webserver implements the scheduling decision, indeed, when a new task arrives, it decides to execute it locally or forward it to another neighbour node according to the current configuration of migration ratios. Migration ratios are updated according to Algorithm 1 every T seconds

For implementing the tasks of variable duration, we used a loop that performed the same operation repeated a fixed amount of times, we measured the duration of a single iteration, and from there, we compute the number of iterations to match the desired μ_i parameter for each node. The operation carried out in the loop is the computation of the SHA-512 hash of the same (20 bytes) string. We measured that the operation in question, in a Raspberry Pi 4, has an average duration of $4.721\mu s$ (on 30'000 iterations repeated 10 times). Therefore, for example, setting $\mu = 2$ is equal to perform $(1/2)/4.721^{-6} \approx 105'900$ loop iterations.

A. Deployment

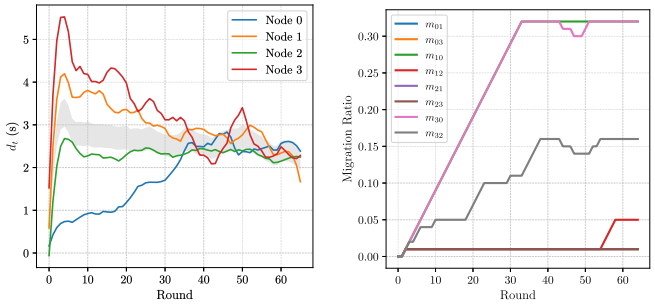
The deployment process involves two phases. (I) After the container is started in every node, the webserver is put on wait

4.https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
5.https://pypi.org/project/Flask/
6.https://gitlab.com/gabrielepmattia/framework-2022-tsc



(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 10. Behaviour of the average latency d_t and migration ratios for Topology A (Fig. 1(a)) in the experimental setting.



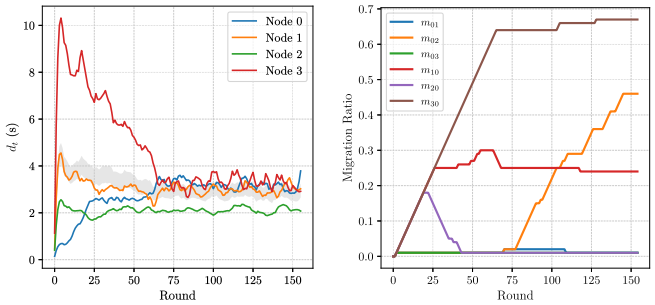
(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 11. Behaviour of the average latency d_t and migration ratios for Topology B (Fig. 1(b)) in the experimental setting.

for the configuration that is passed via POST. The configuration is a JSON file where the main parameters are declared, for example, the queue length K, how many rounds are used for computing the average latency, the round duration T and the balance zone size ϵ. This structure also contains some parameters that regard the identification of the node: the IP, the ID, the name, μ, the step size α and λ. The last part regards the topology of the network that defines with which nodes the communication is possible. After the configuration is received (II) each node starts 2 threads: the *update* thread that is in charge of updating the migration ratios at every round and collecting all statistics parameters used by the algorithm as service latency, the number of executed tasks and the queue length; and the *worker* thread that is in charge executing a service execution request by picking the first available from the internal queue. Now the node is ready to receive the requests from the task generator and the adaptive heuristic (Algorithm 1) updates the migration ratios accordingly every T seconds.

B. Results

All of the topologies shown in Fig. 1 have been run in the above-mentioned framework, we will now illustrate the results obtained. In all the experiments, we set $K_i = 4, \forall\, i$, the round time $T = 30$s, the tolerance $\epsilon = 0.1, \alpha = 0.01$ and all the curves have been filtered with the Savitzy-Golay filter by using window size 20 and polynomial degree 4. Fig. 10 shows the behaviour of the average service latency and of the migration ratios for Topology A (Fig. 1(a)). Regarding the latency Fig. 10(a) we can observe how the alignment value is slightly different from the model Fig. 2(a) and the simulations Fig. 7(a), in particular, the average service latency is levelled to 1.7 s, and this represents an increase of 0.5 s with respect the other tests, but as we can notice the latency at round 1 is not matching the simulations, nor the model and this is justified by the fact that the model of the queue M/M/1/K is not representing well the behaviour of a real node. Moreover, we ignore the eventual background work of the CPU that may interfere with the tasks that we are sampling. However, the algorithm manages to level the latency among all the nodes but with migration ratios that are different from the model. Indeed, in Fig. 10(b) we can observe how Node 0 forwards about the 17.5% of its traffic to Node 2 and the Node 1 forwards about the same amount of traffic to Node 0. This solution found by the heuristic is quite different from the one predicted because we point out that the solution, i.e. the combination of m_{ij} ratios, may not be unique.



(a) Service latency d_t (s) (b) Migration ratios m_{ij}

Fig. 12. Behaviour of the average latency d_t and migration ratios for Topology C (Fig. 1(c)) in the experimental setting.

The Fig. 10 shows the behaviour of the average service latency and of the migration ratios for the Topology B (Fig. 1(b)). As the previous result, the final alignment latency is again different, we pass from 0.8 s, 4.1 s, 2.6 s, 5.5 s (respectively from Node 0 to 3) to 2.5 s for each node with respect 1.5 s in the model and in the simulations. The algorithm manages to level the latency by making Nodes 1 and 3 forward about the 30% of their traffic to Node 0, and Node 3 forward the 15% of its traffic to Node 2 at steady state.

The final test on the real deployment regards Topology C (Fig. 1(c)) and its result is shown in Fig. 10. As we can observe, latencies (Fig. 12(a)) are higher than the ones predicted of 1.5 s, however, the final result is the same since Nodes 0, 1 and 3 are aligned while Node 2 instead cannot reach the alignment latency (see Figs. 5(a) and 8(a)). This is also reflected in the migration ratios (Fig. 12(b)) in which we have Node 3 which forwards the 70% of its load to Node 0 while Node 0 will try to forward all of its traffic to Node 2, even if the Figure is cut to $t = 120$.

Concluding, the results in a real testbed of Raspberry Pi showed how the adaptive heuristic algorithm allows reaching the

final goal of levelling latency with a behaviour that was predicted both in the model and in the simulations. However, due to the absence of a more precise model of a real node, the predicted alignment latency and migration ratios are not the same but this does not limit the applicability of the proposed heuristic, rather the tests showed how it can work even in a real deployment.

VI. Conclusion

In this article, we showed a mathematical modelling of a system of n Fog or Edge nodes which envisions a dynamic in which the service latency is levelled among all the nodes in a given topology. We also provided proof of the convergence of the model to a Wardrop equilibrium. Then, even if from the model we are able to derive the solution, that is, the migration ratios m_{ij} from any node i to a node j, we designed a fully decentralised and adaptive heuristic which is able to reach the same solution but without the need to have a centralised entity (which is able to run the model) and with potential capability to adapt when the load varies over time. We run the algorithm both in simulations and in a real deployment of Raspberry Pi boards, and we showed how the solution is very similar to the one predicted by the mathematical model. However, further research directions are needed to improve the proposed approach. First of all, the communication latency has to be included in the model, while in our case, we only consider them in the final Raspberry Pi deployment, which justifies the differences in the results. Moreover, a more precise model for a real node must be studied since the M/M/1/K does not approximate exactly a real computer node, and this again justifies the discrepancy between the model and the final deployment results. Then, as the last improvements points, a load that varies over time can be introduced in the model, instead of having a fixed λ_i we can suppose to have a $\lambda_i(t)$ function, and we can also consider to jointly level even other performance parameters beyond the single service latency.

Acknowledgments

The authors would also like to thank the former master student Marco Magnani for his preliminary implementation of the proposed heuristic in a cluster of Raspberry Pi [29].

References

[1] R. Fantacci and B. Picano, "Performance analysis of a delay constrained data offloading scheme in an integrated cloud-Fog-edge computing system," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12 004–12 014, Oct. 2020.

[2] M. H. Kashani and E. Mahdipour, "Load balancing algorithms in Fog computing: A systematic review," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1505–1521, Mar./Apr. 2022.

[3] M. Kaur and R. Aron, "A systematic study of load balancing approaches in the Fog computing environment," *J. Supercomputing*, vol. 77, no. 8, pp. 9202–9247, Aug. 2021, doi: 10.1007/s11227–020-03600-8.

[4] A. Chandak and N. K. Ray, "A review of load balancing in Fog computing," in *Proc. Int. Conf. Inf. Technol.*, 2019, pp. 460–465.

[5] R. Beraldi, C. Canali, R. Lancellotti, and G. Proietti Mattia, "Randomized load balancing under loosely correlated state information in Fog computing," in *Proc. 23rd ACM Int. Conf. Model. Anal. Simul. Wireless Mobile Syst.*, Alicante, Spain, 2020, pp. 123–127.

[6] S. Harnal, G. Sharma, N. Seth, and R. D. Mishra, *Load Balancing in Fog Computing Using QoS*. Singapore: Springer, 2022, pp. 147–172, doi: 10.1007/978–981-16-3448-2_8.

[7] D. Baburao, T. Pavankumar, and C. S. R. Prabhu, "Load balancing in the Fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method," *Appl. Nanoscience*, vol. 13, pp. 1045–1054, Jul. 2021, doi: 10.1007/s13204–021-01970-w.

[8] R. K. TripathyBarik and D. S. Roy, "Secure-M2FBalancer: A secure mist to Fog computing-based distributed load balancing framework for smart city application," in *Advances in Communication Devices Networking*, S. Dhar, S. C. Mukhopadhyay, S. N. Sur, and C.-M. Liu Eds., Singapore: Springer, 2022, pp. 277–285.

[9] Q.-M. Nguyen, L.-A. Phan, and T. Kim, "Load-balancing of kubernetes-based edge computing infrastructure using resource adaptive proxy," *Sensors*, vol. 22, no. 8, 2022, Art. no. 2869, [Online]. Available: https://www.mdpi.com/1424--8220/22/8/2869

[10] A. Singh, G. S. Aujla, and R. S. Bali, "Container-based load balancing for energy efficiency in software-defined edge computing environment," *Sustain. Comput.: Inform. Syst.*, vol. 30, 2021, Art. no. 100463. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210537920301876

[11] F. Zhang, R. Deng, X. Zhao, and M. M. Wang, "Load balancing for distributed intelligent edge computing: A state-based game approach," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 4, pp. 1066–1077, Dec. 2021.

[12] S. Sthapit, J. Thompson, N. M. Robertson, and J. R. Hopgood, "Computational load balancing on the edge in absence of cloud and Fog," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1499–1512, Jul. 2019.

[13] X. Xu et al., "Dynamic resource allocation for load balancing in Fog environment," *Wireless Commun. Mobile Comput.*, vol. 2018, Apr. 2018, Art. no. 6421607, doi: 10.1155/2018/6421607.

[14] H. M. Shakir and J. Karimpour, "Systematic study of load balancing in Fog computing in IoT healthcare system," in *Proc. Int. Conf. Adv. Comput. Appl.*, 2021, pp. 132–137.

[15] M. Asif-Ur-Rahman et al., "Toward a heterogeneous mist, Fog, and cloud-based framework for the internet of healthcare things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4049–4062, Jun. 2019.

[16] S. Malik et al., "Intelligent load-balancing framework for Fog-enabled communication in healthcare," *Electronics*, vol. 11, no. 4, 2022, Art. no. 566, [Online]. Available: https://www.mdpi.com/2079--9292/11/4/566

[17] H. A. Khattak et al., "Utilization and load balancing in Fog servers for health applications," *EURASIP J. Wireless Commun. Netw.*, vol. 2019, no. 1, Apr. 2019, Art. no. 91, doi: 10.1186/s13638–019-1395-3.

[18] A. Mijuskovic, A. Chiumento, R. Bemthuis, A. Aldea, and P. Havinga, "Resource management techniques for cloud/Fog and edge computing: An evaluation framework and classification," *Sensors*, vol. 21, no. 5, 2021, Art. no. 1832, [Online]. Available: https://www.mdpi.com/1424--8220/21/5/1832

[19] N. Agrawal, "Dynamic load balancing assisted optimized access control mechanism for edge-Fog-cloud network in Internet of Things environment," *Concurrency Computation: Pract. Experience*, vol. 33, no. 21, 2021, Art. no. e6440. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6440

[20] F. Alqahtani, M. Amoon, and A. A. Nasr, "Reliable scheduling and load balancing for requests in cloud-Fog computing," *Peer-to-Peer Netw. Appl.*, vol. 14, no. 4, pp. 1905–1916, Jul. 2021, doi: 10.1007/s12083–021-01125-2.

[21] W. Li, S. Cao, K. Hu, J. Cao, and R. Buyya, "Blockchain-enhanced fair task scheduling for cloud-Fog-edge coordination environments: Model and algorithm," *Secur. Commun. Netw.*, vol. 2021, Apr. 2021, Art. no. 5563312, doi: 10.1155/2021/5563312.

[22] E. Batista, G. Figueiredo, and C. Prazeres, "Load balancing between Fog and cloud in Fog of things based platforms through software-defined networking," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, pp. 7111–7125, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157821002901

[23] A. Pietrabissa and V. Suraci, "Wardrop equilibrium on time-varying graphs," *Automatica*, vol. 84, pp. 159–165, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109817303643

[24] F. M. Talaat, M. S. Saraya, A. I. Saleh, H. A. Ali, and S. H. Ali, "A load balancing and optimization strategy (LBOS) using reinforcement learning in Fog computing environment," *J. Ambient Intell. Humanized Comput.*, vol. 11, pp. 4951–4966, 2020.

[25] A. AlOrbani and M. Bauer, "Load balancing and resource allocation in smart cities using reinforcement learning," in *Proc. IEEE Int. Smart Cities Conf.*, Piscataway, NJ, USA: IEEE, 2021, pp. 1–7.
[26] G. P. Mattia and R. Beraldi, "On real-time scheduling in Fog computing: A reinforcement learning algorithm with application to smart cities," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Other Affiliated Events*, Piscataway, NJ, USA: IEEE, 2022, pp. 187–193.
[27] S. Fischer, L. Olbrich, and B. Vöcking, "Approximating wardrop equilibria with finitely many agents," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer, 2008, pp. 238–252.
[28] M. Beckmann, C. B. McGuire, and C. B. Winsten, "Studies in the economics of transportation," *Econometrica*, vol. 26, no. 1, Jan. 1958, Art. no. 183.
[29] G. Proietti Mattia, M. Magnani, and R. Beraldi, "A latency-levelling load balancing algorithm for Fog and edge computing," in *Proc. 25th ACM Int. Conf. Model. Anal. Simul. Wireless Mobile Syst.*, Montreal, Canada, 2022, pp. 5–14, doi: 10.1145/3551659.3559048.

Gabriele Proietti Mattia received the BSc, MSc, and PhD degrees in computer science from the Sapienza University of Rome, Italy, in 2017, 2019, and 2023, respectively. He is currently a postdoc researcher in computer science, and his research interests include Fog and Edge computing, distributed systems and algorithms, microservices, mobile applications, and machine learning.

Antonio Pietrabissa received the MSc degree in electronics engineering and the PhD degree in systems engineering from the University of Rome Sapienza, in 2000 and 2004, respectively. He is associate professor with the Department of Computer, Control and Management Engineering "Antonio Ruberti" (DIAG), University of Rome Sapienza, where he teaches Automatic Control and Process Automation. Since 2000, he has participated in about 30 EU and National research projects, playing the role of scientific coordinator of the projects 5G-ALLSTARS, on 5 G communications, funded within the H2020 Europe-South Korea cooperation, ARIES, on fire emergency prevention, funded by ESA, DAAS, on safe communications in explosive risk areas, funded by MIUR-FILAS, and FedMedAI, on medical applications of federated learning, funded by Regione Lazio (IT). He serves as associate editor for Control Engineering Practice (Elsevier) and for *IEEE Transactions on Automation Science and Engineering*. His research interests include the application of systems and control theory to the analysis and control of networks. He is the author of about 60 journal papers and 85 conference papers.

Roberto Beraldi received the laurea degree from the University of Calabria, in 1991, the master's degree from CEFRIEL (Politecnico di Milano), in 1992, and the PhD degree in computer science, in 1996. From 1996 he worked with the Italian's National Institute of Statistics (ISTAT), and since 2002, works with the Department of Computer, Control and Management Engineering "Antonio Ruberti" of Sapienza University of Rome, Italy, where he is currently an associate professor. His research interests include mobile networking, Fog/Edge computing, and distributed systems. He regularly serves as TPC member of international conferences and journals in these fields.