A Latency-levelling Load Balancing Algorithm for Fog and Edge Computing

Gabriele Proietti Mattia
Marco Magnani
Roberto Beraldi
proiettimattia@diag.uniroma1.it
beraldi@diag.uniroma1.it
Department of Computer, Control and Management,
Engineering "Antonio Ruberti", Sapienza University of Rome
Rome, Italy

ABSTRACT

When deploying a distributed application in the Fog or Edge computing environments, the average service latency among all the involved nodes can be an indicator of how much a node is loaded with respect to the other. 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 perspective and they cannot be used for a QoS-oriented load balancing of the system. Moreover, due to the displacement of the nodes and the heterogeneity of the computing devices the necessity of a load balancing algorithm is clear. In this paper, we propose a load balancing approach that is focused on the service latency with the objective to level it across all the nodes in a fully decentralized 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, we show both in simulation and in a real-world deployment that our approach is able to level the service latency among a set of heterogeneous nodes organized in different topologies.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**; **Modeling and simulation**; • **Networks** → In-network processing.

KEYWORDS

edge computing; fog computing; load balancing; service latency

1 INTRODUCTION

Service latency plays a crucial role in modern distributed applications [9]. In particular, in the Edge and Fog Computing environments, due to 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, what we can say about the application that is running on both devices? Will the users that make task requests to Node A experience the same service latency as the ones that will make requests to Node B? Yes, but only in one case, the CPU A must be equal to 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 decentralized 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 summarized as follows.

- A continuous-time model which describes the dynamics of the system by using a system of differential equations

that reaches the stability when all the nodes experience the same service latency;
- 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;
- 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 organized as follows. In Section 2 we present some related work, then in Section 3 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 4 that is tested in a simulator. Then in Section 5 we show results of the proposed heuristic in a real environment and finally, the conclusions will be drawn in Section 6.

2 RELATED WORK

The main area in which this work lies is the problem of load balancing in Edge and Fog computing [10] [12] [8] [7]. 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 [11] 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 [5]. Then, Tripathy et al. in [23] 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 [18], 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 [20] 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 [25], however, no simulation or real implementation results are provided. Sthapit et al, in [21] 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 [24].

A set of works, instead, focus on healthcare [19] and the "internet of healthcare things" [4]. For example, [15] proposes a load balancing framework which is able to avoid any failure in responsiveness and [13] which targets a smart city. Both the 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 [17] 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, [1] proposes an Edge-Fog-Cloud algorithm for distributing the traffic in all of the three layers but the focus is not the latency optimisation, [9] provides a model based on queuing theory, [3] 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 [14] proposes a scheduling approach based on blockchain and [6] a strategy to cope with failures by using Software-Defined Networks (SDN).

In conclusion, the last set of works worth mentioning focuses on load balancing by using intelligent approaches like reinforcement learning [22], [2], [16]. 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.

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
λ_i	Traffic to node i (in reqs/s)
μ_i	Service rate of node i (in reqs/s)
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 (of λ_i) of tasks forwarded from node i to node j at time t
	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 of the average 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

Table 1: List of symbols used

3 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 requests λ_i req/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 λ_i to a given neighbour 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 also stress the fact that a node i cannot forward the load

that it receives from other nodes and it can only forward the one from the clients, that is λ_i.

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

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

where the following conditions must be followed

$$0 \leq m_{ij}(t) \leq 1, \quad \tilde{\sum_{j}} m_{ij}(t) \leq 1 \qquad \forall\, i, j, t \tag{2}$$

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

$$x_i(0) = \lambda_i \qquad \forall\, i \tag{3}$$

Equation 1 can be interpreted as follows. A node i, receives constant traffic by the clients that are connected to it, that is λ_i, then a part of this traffic can be forwarded to the neighbour nodes (for which $a_{ij} < 0$) and it is subtracted, but neighbour nodes may also decide to forward part of their traffic to i and this part is added to the total load of the node. For any node i, the functions $m_{ij}(t)\ \forall j$ describe the portions of incoming traffic λ_i that are forwarded to the neighbour nodes and they are our unknowns. By knowing the $m_{ij}(t)$, we will then need to find a time t^* where $m_{ij}(t) = m_{ij}(t^*)$, $\forall i, j, t > t^*$, and the values $m_{ij}(t^*)\ \forall i, j$ will be the final migration ratios that each node will need to apply to reach the final goal. At this point, we need to model this final goal: the levelling of latencies. For finding the functions $m_{ij}(t)$, instead of trying to define them directly, it is easier to describe their variation over time, for this reason, we calculate the derivative with respect to the time of Equation 1 that is:

$$\dot{x}_i(t) = -\tilde{\sum_{j \in V}} a_{ij}\lambda_i \dot{m}_{ij}(t) + \tilde{\sum_{j \in V}} a_{ji}\lambda_j \dot{m}_{ji}(t) \tag{4}$$

Equation 4, 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 $\dot{m}_{ij}(t)$ 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 $\dot{m}_{ij}(t)$ 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 $\rho_i(t) = x_i(t)/\mu_i$):

$$l_i(t) = \frac{1 - (K_i + 1)\rho_i(t)^{K_i} + K_i\rho_i(t)^{(K_i+1)}}{\mu_i(1 - \rho_i(t))(1 - \rho_i(t)^{K_i})} \tag{5}$$

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 we can level them to the average? There are three sub-strategies that we need to adopt to reach the goal and they concretize in 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 neighbors, called l_{a_i}, for this reason the first factor is:

$$\dot{m}^{\alpha}_{ij}(t) = \max\ 0, \frac{l_i(t) - l_{a_i}(t)}{l_i(t)} \tag{6}$$

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

$$\dot{m}^{\beta}_{ij}(t) = \max\ 0, \frac{l_i(t) - l_j(t)}{l_{h_i}(t)} \tag{7}$$

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

$$\dot{m}^{\gamma}_{ij}(t) = \max\ 0, \frac{l_{a_i}(t) - l_j(t)}{l_{k_i}(t)} \tag{8}$$

The final dynamic of the migration ratios is therefore

$$\dot{m}_{ij}(t) = m^{\alpha}_{ij}(t) \cdot m^{\beta}_{ij}(t) \cdot m^{\gamma}_{ij}(t) \tag{9}$$

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 neighbors:

$$l_{a_i}(t) = \frac{l_i(t) + \prod_{j \in V; i<j} a_{ij} l_j(t)}{1 + \prod_{j \in V} a_{ij}} \tag{10}$$

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

$$l_{h_i}(t) = \max\left\{0, \tilde{\sum_{j \in V}} l_i(t) - l_j(t)\right\} \tag{11}$$

$$l_{k_i}(t) = \max\left\{0, \tilde{\sum_{j \in V}} l_{a_i}(t) - l_j(t)\right\} \tag{12}$$

We will resort to numerical calculus to find the time trajectories of the system of non-linear ODE described in Equation 4 with initial conditions $x_i(0) = \lambda_i,\ \forall i$ but unconstrained for simplicity. Then, after finding the numerical solution $x_i(t), \forall i$, we can easily find the effective behaviour of migration ratios over time considering that:

$$m_{ij}(t) = \int_0^t \dot{m}_{ij}(\xi)\, d\xi \tag{13}$$

We will consider the trajectory of the solution valid until the condition expressed in Equation 2 is respected.

3.1 Latency-levelling property

We now prove that when the trajectories of the solution of the system at Equation 4 converge, then the latencies are aligned to the same value. In the Appendix at Section 7 we instead prove the existence and the uniqueness of a set of steady states $x_i \forall i$ for which the latencies are levelled.

Theorem 3.1. *If the solution's trajectories of the O.D.E. system at Equation 4 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. $l_i(t) < l_j(t), \forall j < i, \forall t > t*$. We can distinguish two possible cases, for any $t > t^*$:

(a) $l_i(t) > l_{a_i}$, 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 neighbour 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. From this fact we have that the $\dot{m}^{\alpha}_{ij}(t) < 0$ by definition, $\dot{m}^{\beta}_{ij}(t) < 0$ and $\dot{m}^{\gamma}_{ij}(t) < 0$ because there exist 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 4 the negative part is not zero, the positive part instead is zero since i is the only one node with latency higher than the average it will not receive traffic from any neighbour. Therefore we showed that $\dot{x}(t) < 0$ for some $t > t^*$, and this is a contradiction ※;

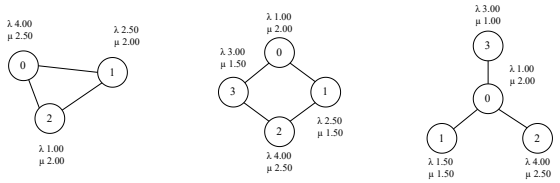
(b) $l_i(t) < l_{a_i}$, i.e. the service latency of node i is *lower* than the average latency between i and its neighbours. As in the case (a) if the node i's latency is below than the average latency then there exists at least one neighbour j whose latency is higher than the average. The consequences are exactly the ones of case (a) 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}.

3.2 Trajectories and Topologies

We will now explore some configuration 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 behaviour of the latency over time $d_t(t)$, computed by using the Equation 5 and the migration ratios $m_{ij}(t)$ computed by using Equation 13.

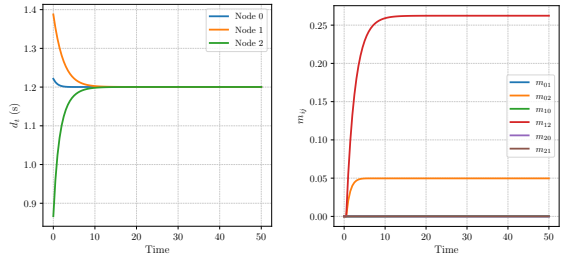
We tested different network topologies, the first three are shown in Figure 1. These small topologies are taken into consideration because it is easy to have a direct comparison with the behaviour shown in simulations and in the real deployment. Finally, we tested a fully connected topology with 15 nodes.



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

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

Topology A (Figure 1a) is composed of three nodes arranged in a fully connected graph, the Figure 2a 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 1.2s. From the migration ratio, Figure 2b we can observe that the 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.

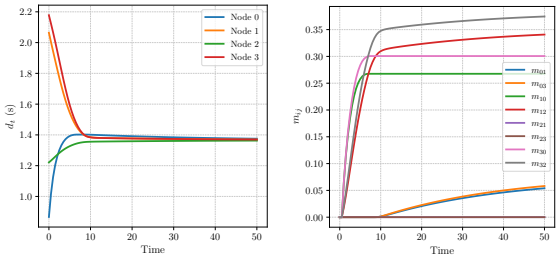


(a) Service latency d_t (s) (b) Migration ratios m_{ij}

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

Topology B (Figure 1b) comprehends four nodes connected as a ring, the Figure 1b shows the numerical trajectories of the the performance parameters. Each node, from 0 to 3, starts with service latency, respectively, 0.86s, 2.06s, 1.22s and 2.18s and the end of the transient (Figure 3a) is levelled to 1.38s. At steady state and we can observe how (Figure 3b) 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 the 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 for diminishing them is making a node to give back the load to the sender.



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

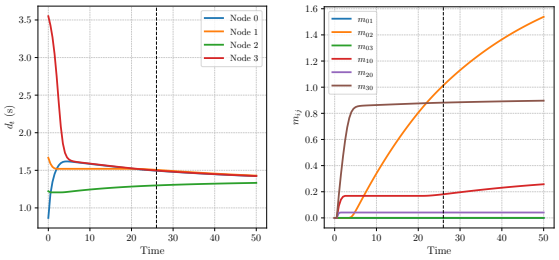
Figure 3: Trajectories of the average latency d_t and the migration ratios m_{ij} for the four nodes described by Topology B (Figure 1b).

Topology C (Figure 1c) 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 1.4s (Figure 4a) but the solution that is reached is actually not achievable because the condition expressed at Equation 2 is no more respected (Figure 4b), 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 $0 \leq \lambda_i \leq 4$, $0 \leq \mu_i \leq 4$ and $2 \leq K_i \leq 6$. All of these parameters are picked at random, but the purpose of this is to understand how the system behaves with many nodes. The SageMath[1] Python ODE solver took about 20 hours to derive the trajectories up to $t = 100$ with the numeric solver Runga-Kutta-Felhberg on a Ryzen 9 5800X processor. Figure 5a shows the behaviour of the latency for all the nodes and as we can see the system reduce 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 $t \geq 31$ (5b).

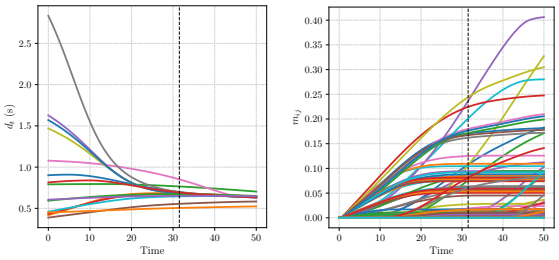
This last result shows how the model scales with the number of nodes, however, we do not envision modelling a system of more than 20 Edge or Fog nodes, because aligning the latencies in a very large set of nodes may not be the best strategy for balancing the load.

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



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

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



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

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

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 latencies, without counting the difficulties of implementing the algorithm in the real world where the network latencies have a significant impact. Instead, it is more efficient to create groups of a maximum of 20 nodes and try levelling the latencies within the groups, these groups can, for example, represent neighbourhoods of a smart city.

4 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 decentralized 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 latencies that exist 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.

The Figure 6 summarizes 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 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:
    (a) 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;
    (b) if the average latency of neighbour node is below the balance zone, then we increase the migration ration towards it of the step size α, this will cause our latency to be reduced and its one to increase, approaching the balancing zone.

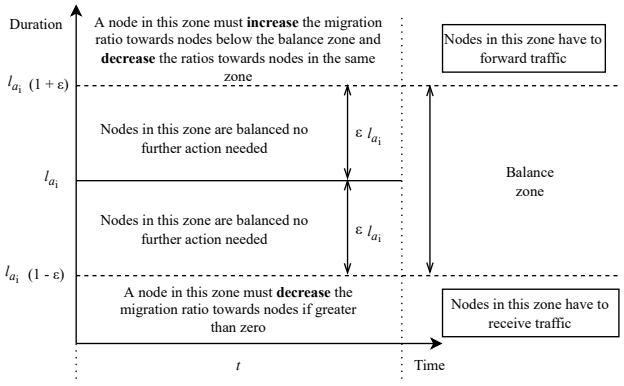


Figure 6: Representation of the logic behind the adaptive heuristic for a node i in a given time t. We suppose the average delay l_{a_i} between the node i and its neighbours to be fixed during an instant time t.

4.1 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 (Figure 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 $T = 60s$ 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.

Figure 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 1.2s (Figure 7a), 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

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

Algorithm 1 Adaptive Heuristic for leveling latencies

Require: $i, \alpha, \mathcal{N}, \vec{M}_i, \epsilon, \mathcal{I}_i$
currentNode ← $\mathcal{N}[i]$
[*1. Compute the average latency among all the neighbour nodes*]
averageLatency ← currentNode.latency
numberOfNeighbours ← 0
for all j in $|\mathcal{N}|$ **and** $\mathcal{I}_{ij} < 0$ [*Loop over the neighbours*] **do**
averageLatency ← node.latency **and**
numberOfNeighbours ← numberOfNeighbours + 1
end for
averageLatency ← averageLatency / numberOfNeighbours
averageLatencyLow ← averageLatency · $(1.0 + \epsilon)$
averageLatencyHigh ← averageLatency · $(1.0 - \epsilon)$
totalRatiosGiven ← $\sum_j m_{ij}$
[*2. If under average and migrating, then reduce migration*]
if currentNode.getAverageLatency() ≤ averageLatencyLow **and** totalRatiosGiven > 0 **then**
for all j in $|\mathcal{N}|$ **and** $\mathcal{I}_{ij} < 0$ **do**
if $\mathcal{N}[j]$.getAverageLatency() ≥ averageLatencyHigh **then**
if canBeSubtractedToNode(j, α) **and** canSubtract(α) **then**
$m_{ij} \leftarrow m_{ij} - \alpha$
totalRatiosGiven ← 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.getAverageLatency() < 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** $\mathcal{I}_{ij} < 0$ **do**
[*4a. Reduce the ratio to neighbour above the average high limit*]
if $\mathcal{N}[j]$.getAverageLatency() ≥ averageLatencyHigh **then**
if canBeSubtractedToNode(j, α) **and** canSubtract(α) **then**
$m_{ij} \leftarrow m_{ij} - \alpha$
totalRatiosGiven ← totalRatiosGiven - α
end if
end if
[*4b. Increase the ratio to neighbour below the average low limit*]
if $\mathcal{N}[j]$.getAverageLatency() ≤ averageLatencyLow **then**
if canBeGiven(α) **then**
$m_{ij} \leftarrow m_{ij} + \alpha$
totalRatiosGiven ← totalRatiosGiven + α
end if
end if
end for

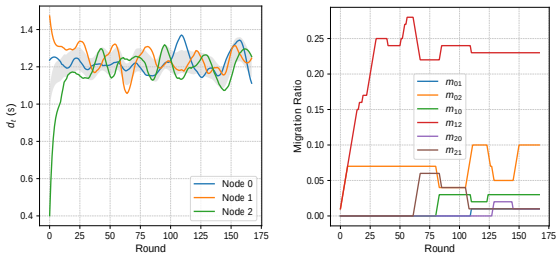
Algorithm 2 Auxiliary functions

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

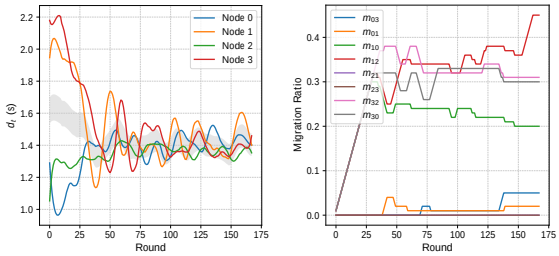
compared to Figure 2a, the fluctuations around the average is due to the exponential inter-arrival times and execution times. For levelling the latency the migration ratios found by the algorithm are represented in Figure 7b. In particular, we can observe that m_{12} stabilizes at around 0.24 and m_{02} at around 0.07 while the others are less than 0.03. Again these result matches the ones of the model shown in Figure 2b, in which m_{12} and m_{02} stabilizes at 0.26 and 0.05 respectively, while the others are set to 0.



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

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

Topology B results are shown in Figure 8. As far as regards the average service latency (Figure 8a) we can observe how it stabilizes at about 1.4s which is in line with the mathematical model shown Figure 3a. 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 model 34%. The same slight differences hold for Nodes 3 and 4 and are due to the traffic variability.

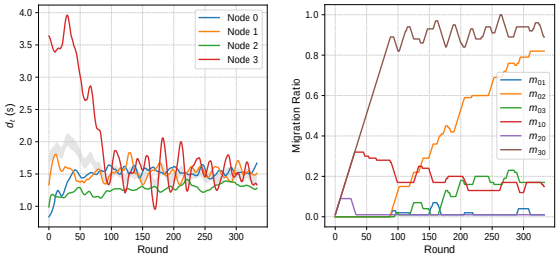


(a) Service latency d_t (s) (b) Migration ratios m_{ij}

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

Topology C results are shown in Figure 9. Regarding the service latency (Figure 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 Figure 4a 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 1.5s while Node 2 stabilizes to 1.3s because it cannot receive enough traffic from Node 0 in order to increase its latency to match 1.5s. This does happen in the model after $t = 26$ but Node 0 would forward more traffic than the one that is available. Regarding instead the migration ratios, shown in Figure 9b, we can

observe that as the latency, they match with the truncated solution of the model (Figure 4b) 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.



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

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

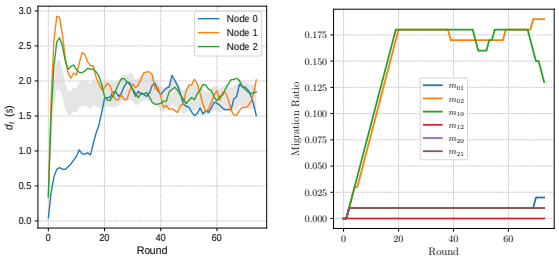
5 EXPERIMENTAL SETTING

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.

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

Results. All of the topologies shown in Figure 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. The Figure 10 shows the behaviour of the average service latency and of the migration ratios for the Topology A (Figure 1a). Regarding the latency (Figure 10a) we can observe how the alignment value is slightly different from the model (Figure 2a) and the simulations (Figure 7a), in particular, the average service latency is levelled to 1.7s and this represents an increase of 0.5s 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 tasks that we are sampling. However, the algorithm manages to level the latencies among all the nodes but with migration ratios that are different from the model. Indeed, in Figure 10b we can observe how the 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.



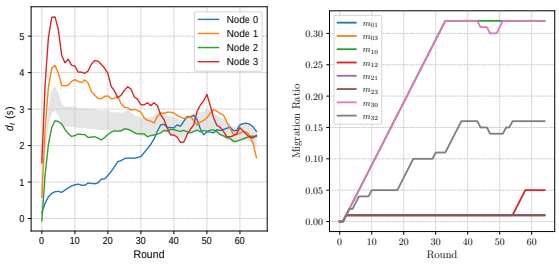
(a) Service latency d_t (s) (b) Migration ratios m_{ij}

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

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

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

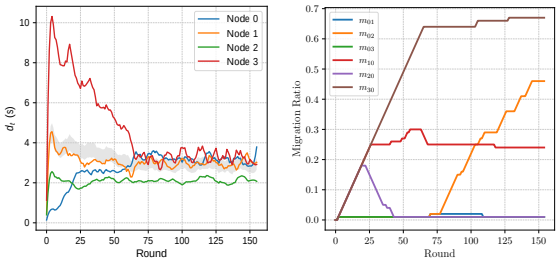
and 3 forward about the 30% of their traffic to Node 0, and Node 3 forward the 15% of its traffic to Node 2 at steady state.



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

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

The final test on the real deployment regards Topology C (Figure 1c) and its result is shown in Figure 10. As we can observe, latencies (Figure 12a) are higher than the ones predicted of 1.5s, 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 Figures 5a and 8a). This is also reflected in the migration ratios (Figure 12b) in which we have the Node 3 which forwards the 70% of its load to Node 0 while the Node 0 will try to forward all of its traffic to Node 2, even if the Figure is cut to $t = 120$.



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

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

Concluding, the results in a real testbed of Raspberry Pi showed how the adaptive heuristic algorithm allows reaching the final goal of levelling latencies 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 latencies 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.

6 CONCLUSIONS

In this paper, we started with mathematical modelling of a system of n Fog or Edge nodes for designing a dynamic which is able to level the service latency among all the nodes in a given topology. 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 decentralized and adaptive heuristic which is able to reach the same solution but without the need to have a centralized 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.

7 APPENDIX

In the following we consider a completely connected topology, and a generic load-delay relationship $f_i(\lambda)$ which is a monotonically increasing continuous function, with $f(0) = 0$ and $\lim_{\lambda\to\infty} f_i(\lambda) = d_{M_i}$. The transmission delay is not considered, but the same proof sketch can be used by adding the transmission delay to the definition of f_i.

Property 7.1 (Existence of balanced loads). *Given a vector $\Lambda = (\lambda_1, \lambda_2, \ldots, \lambda_N)$ of loads, $|\Gamma| \stackrel{def}{=} \sum_i \lambda_i = \lambda_T$ it there exists another vector $\Lambda' = (\lambda'_1, \lambda'_2, \ldots, \lambda'_N)$ such that:*

$$f_1(\lambda'_1) = f_2(\lambda'_2) = \ldots = f_N(\lambda'_N) = d, |\Gamma| = |\Gamma'|$$

Proof. Let consider the function:

$$\lambda'_T(d) = f_1^{-1}(d) + f_2^{-1}(d) + \ldots + f_N^{-1}(d)$$

and let $d_M = min\{d_{M_i}\}$. Due to the property of f, this function is continuous and increases monotonically with d; moreover, $\lambda'_T(0) = 0, \lambda'_T(d_M) = \infty$. Since $\lambda'_T(0) - \lambda_T < 0, \lambda'_T(d_M) - \lambda_T > 0$, due to the Bolzano's theorem, it there exits a value $d < d_M$ such that $\lambda'_T(d) - \lambda_T = 0$, i.e. $\lambda'_T(d) = \lambda_T$.

Property 7.2 (Unicity of balanced loads). *For a given d, the vector Λ' is unique.*

Proof. Follows from the properties of f_i.

Property 7.3 (Migration). *Given two load vectors of size N, Λ and Λ', $|\Lambda| = |\Lambda'|$, where Λ' is such that $f_i(\lambda'_i) = d > 0$, it there exists at least an $N \times N$ migration matrix M such that:*

$$\Lambda' = \mathbf{M}\Lambda$$

where $0 \le m_{ij} \le 1$, $\sum_i m_{ij} = 1$.

Proof. Since $|\Lambda| = |\Lambda'|$, Λ can be partitioned in two sets, $\mathcal{A}$ and $\mathcal{B}$, namely the set of nodes such that $\lambda'_i \leq \lambda_i$ and the set of nodes such that $\lambda'_i > \lambda_i$ - unless $\Lambda = \Lambda'$ in which case $\mathbf{M} = \mathbf{I}$.

First of all, observer that all nodes in $\mathcal{B}$ have $\lambda'_j > \lambda_j$, so we can set $m_{ij} = 0, j \in \mathcal{B}, i \in \mathcal{A}$, $m_{jj} = 1$ (these nodes only receive load from others).

The other values $m_{ij}, j \in \mathcal{A}, i \in \mathcal{B}$ are constrained as following:

$$\lambda_i = \lambda_i + \sum_{j \in \mathcal{A}} m_{ij}\lambda_j \quad i \in \mathcal{B}$$

There are $B = |\mathcal{B}|$ of these equations, each with A unknowns, $A = |\mathcal{A}|$. In addition there are other A constrains on the coefficients, i.e. $(1 - \sum_i m_{ij})\lambda_j = \lambda'_j$. Of these equations one is redundant since it must be $\Sigma_i \lambda_i = \Sigma_i \lambda'_i$, so only $A + B - 1$ are truly independent. Overall, we have AB unknowns and $A + B - 1$ equations. Since $A + B = N, AB \geq A + B - 1$, i.e. the unknowns are at least equal to the number of constrains. The system of equations has then either one solution or it is undetermined and infinity solutions exist. Since each node $j \in \mathcal{A}$ migrates a fraction $m_j < 1$ of λ_j towards nodes of $\mathcal{B}$ (this is true since $f_i(0) = 0$ and $d > 0$) $m_j = \frac{\lambda'_j}{\lambda_j} < 1$ and all the coefficients are <1.

Example of migration matrix with $N = 4, \mathcal{A} = \{1, 2\}$ showing 4 unknowns:

$$\begin{pmatrix} \lambda'_1 \\ \lambda'_2 \\ \lambda'_3 \\ \lambda'_4 \end{pmatrix} = \begin{pmatrix} 1 - m_{31} - m_{41} & 0 & 0 & 0 \\ 0 & 1 - m_{32} - m_{42} & 0 & 0 \\ m_{31} & m_{32} & 1 & 0 \\ m_{41} & m_{42} & 0 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{pmatrix}$$

REFERENCES

[1] Neha Agrawal. 2021. Dynamic load balancing assisted optimized access control mechanism for Edge-Fog-Cloud network in Internet of Things environment. *Concurrency and Computation: Practice and Experience* 33, 21 (2021), e6440. https://doi.org/10.1002/cpe.6440 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6440

[2] Aseel AlOrbani and Michael Bauer. 2021. Load Balancing and Resource Allocation in Smart Cities using Reinforcement Learning. In *2021 IEEE International Smart Cities Conference (ISC2)*. IEEE, 445 Hoes Lane, Piscataway, NJ 08854-4141 USA, 1–7. https://doi.org/10.1109/ISC253183.2021.9562941

[3] Fayez Alqahtani, Mohammed Amoon, and Aida A. Nasr. 2021. Reliable scheduling and load balancing for requests in cloud-fog computing. *Peer-to-Peer Networking and Applications* 14, 4 (01 Jul 2021), 1905–1916. https://doi.org/10.1007/s12083-021-01125-2

[4] Md. Asif-Ur-Rahman, Fariha Afsana, Mufti Mahmud, M. Shamim Kaiser, Muhammad R. Ahmed, Omprakash Kaiwartya, and Anne James-Taylor. 2019. Toward a Heterogeneous Mist, Fog, and Cloud-Based Framework for the Internet of Healthcare Things. *IEEE Internet of Things Journal* 6, 3 (2019), 4049–4062. https://doi.org/10.1109/JIOT.2018.2876088

[5] D. Baburao, T. Pavankumar, and C. S. R. Prabhu. 2021. Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method. *Applied Nanoscience* (21 Jul 2021). https://doi.org/10.1007/s13204-021-01970-w

[6] Ernando Batista, Gustavo Figueiredo, and Cassio Prazeres. 2021. Load balancing between fog and cloud in fog of things based platforms through software-defined networking. *Journal of King Saud University - Computer and Information Sciences* (2021). https://doi.org/10.1016/j.jksuci.2021.10.003

[7] Roberto Beraldi, Claudia Canali, Riccardo Lancellotti, and Gabriele Proietti Mattia. 2020. Randomized load balancing under loosely correlated state information in fog computing. In *23rd ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'20)*. Alicante, Spain. https://doi.org/10.1145/3416010.3423244

[8] Ashish Chandak and Niranjan Kumar Ray. 2019. A Review of Load Balancing in Fog Computing. In *2019 International Conference on Information Technology (ICIT)*. 460–465. https://doi.org/10.1109/ICIT48102.2019.00087

[9] Romano Fantacci and Benedetta Picano. 2020. Performance Analysis of a Delay Constrained Data Offloading Scheme in an Integrated Cloud-Fog-Edge Computing System. *IEEE Transactions on Vehicular Technology* 69 (2020), 12004–12014.

[10] Mostafa Haghi Kashani and Ebrahim Mahdipour. 2022. Load Balancing Algorithms in Fog Computing: A Systematic Review. *IEEE Transactions on Services Computing* (2022), 1–1. https://doi.org/10.1109/TSC.2022.3174475

[11] Shilpi Harnal, Gaurav Sharma, Nidhi Seth, and Ravi Dutt Mishra. 2022. *Load Balancing in Fog Computing Using QoS*. Springer Singapore, Singapore, 147–172. https://doi.org/10.1007/978-981-16-3448-2_8

[12] Mandeep Kaur and Rajni Aron. 2021. A systematic study of load balancing approaches in the fog computing environment. *The Journal of Supercomputing* 77, 8 (01 Aug 2021), 9202–9247. https://doi.org/10.1007/s11227-020-03600-8

[13] Hasan Ali Khattak, Hafsa Arshad, Saif ul Islam, Ghufran Ahmed, Sohail Jabbar, Abdullahi Mohamud Sharif, and Shehzad Khalid. 2019. Utilization and load balancing in fog servers for health applications. *EURASIP Journal on Wireless Communications and Networking* 2019, 1 (08 Apr 2019), 91. https://doi.org/10.1186/s13638-019-1395-3

[14] Wenjuan Li, Shihua Cao, Keyong Hu, Jian Cao, and Rajkumar Buyya. 2021. Blockchain-Enhanced Fair Task Scheduling for Cloud-Fog-Edge Coordination Environments: Model and Algorithm. *Security and Communication Networks* 2021 (05 Apr 2021), 5563312. https://doi.org/10.1155/2021/5563312

[15] Swati Malik, Kamali Gupta, Deepali Gupta, Aman Singh, Muhammad Ibrahim, Arturo Ortega-Mansilla, Nitin Goyal, and Habib Hamam. 2022. Intelligent Load-Balancing Framework for Fog-Enabled Communication in Healthcare. *Electronics* 11, 4 (2022). https://doi.org/10.3390/electronics11040566

[16] Gabriele Proietti Mattia and Roberto Beraldi. 2022. On real-time scheduling in Fog computing: A Reinforcement Learning algorithm with application to smart cities. In *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, 445 Hoes Lane, Piscataway, NJ 08854-4141 USA, 187–193. https://doi.org/10.1109/PerComWorkshops53856.2022.9767498

[17] Adriana Mijuskovic, Alessandro Chiumento, Rob Bemthuis, Adina Aldea, and Paul Havinga. 2021. Resource Management Techniques for Cloud/Fog and Edge Computing: An Evaluation Framework and Classification. *Sensors* 21, 5 (2021). https://doi.org/10.3390/s21051832

[18] Quang-Minh Nguyen, Linh-An Phan, and Taehong Kim. 2022. Load-Balancing of Kubernetes-Based Edge Computing Infrastructure Using Resource Adaptive Proxy. *Sensors* 22, 8 (2022). https://doi.org/10.3390/s22082869

[19] Hayder Makki Shakir and Jaber Karimpour. 2021. Systematic Study of Load Balancing in Fog Computing in IOT Healthcare system. In *2021 International Conference on Advanced Computer Applications (ACA)*. 132–137. https://doi.org/10.1109/ACA52198.2021.9626813

[20] Amritpal Singh, Gagangeet Singh Aujla, and Rasmeet Singh Bali. 2021. Container-based load balancing for energy efficiency in software-defined edge computing environment. *Sustainable Computing: Informatics and Systems* 30 (2021), 100463. https://doi.org/10.1016/j.suscom.2020.100463

[21] Saurav Sthapit, John Thompson, Neil M. Robertson, and James R. Hopgood. 2019. Computational Load Balancing on the Edge in Absence of Cloud and Fog. *IEEE Transactions on Mobile Computing* 18, 7 (2019), 1499–1512. https://doi.org/10.1109/TMC.2018.2863301

[22] Fatma M. Talaat, Mohamed S. Saraya, Ahmed I. Saleh, Hesham Arafat Ali, and Shereen H. Ali. 2020. A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment. *Journal of Ambient Intelligence and Humanized Computing* (2020), 1–16.

[23] Subhranshu Sekhar Tripathy, Rabindra K. Barik, and Diptendu Sinha Roy. 2022. Secure-M2FBalancer: A Secure Mist to Fog Computing-Based Distributed Load Balancing Framework for Smart City Application. In *Advances in Communication, Devices and Networking*, Sourav Dhar, Subhas Chandra Mukhopadhyay, Samarendra Nath Sur, and Chuan-Ming Liu (Eds.). Springer Singapore, Singapore, 277–285.

[24] Xiaolong Xu, Shucun Fu, Qing Cai, Wei Tian, Wenjie Liu, Wanchun Dou, Xingming Sun, and Alex X. Liu. 2018. Dynamic Resource Allocation for Load Balancing in Fog Environment. *Wireless Communications and Mobile Computing* 2018 (26 Apr 2018), 6421607. https://doi.org/10.1155/2018/6421607

[25] Fenghui Zhang, Ruilong Deng, Xinsheng Zhao, and Michael Mao Wang. 2021. Load Balancing for Distributed Intelligent Edge Computing: A State-Based Game Approach. *IEEE Transactions on Cognitive Communications and Networking* 7, 4 (2021), 1066–1077. https://doi.org/10.1109/TCCN.2021.3087178