A randomized low latency resource sharing algorithm for Fog Computing

Roberto Beraldi, Gabriele Proietti Mattia

DIAG, La Sapienza University of Rome, Italy

***Abstract*—In this paper, we propose and report a study of a low latency resource sharing protocol for Fog Computing. The protocol has its root in the power-of-random choices family of randomization protocol. The protocol, dubbed $LL_g(T)$ is designed to cope with a not homogeneous set of nodes and dealing with a communication latency comparable with the task execution, a characteristic of time-constrained applications supported by this service delivery model. The protocol allows to determine when a task can be moved from the origin fog node that receives the task to another node, where it can be executed faster. This task handoff is controlled via a threshold T. The remote node is selected uniformly at random.**

I. Introduction

In the last decade we have seen a progressively *cloudfication* of software applications, with the business code and data running and stored in the cloud. This cloud-is-the computer paradigm is inadeguate for supporting forthcoming time sensible applications. For example, for such applications as Augmented or Virtual Reality the response time (rt), must be less than 5 ms (e.g. the so called motion-to-photon latency), whereas the round trip time (rtt) measured to hit a cloud data center is variable from tens to hundreds ms, [1]. Many others examples like connected autonomous self driving cars require a similar requirement or even less. Although this figure may in principle be reduced by future technological improvements, there are physical limits, for example dictated by the signal propagation's speed propagation, that set an upper bound to the achievable latency .

In fact, [2] critically analyze that the general distributed computing landscape of the coming years will be based on a multilevel architecture with levels that are psychically closer to the data sources. Fog Computing is an architectural change that consists of creating an intermediate layer between the cloud and data sources, where it is possible to implement (some critical parts of) critical applications [3]. In essence, the fog nodes are distributed a few kilometers away from the end users or even in the radio access networks of the 5G (F-RAN) architecture, [4]. Pushing the calculation and storage capability of cloud computing towards the edge of the network reduces the end-to-end communication delay of one / two oder of magnitude, as well as the overhead of the main network [5], [6]. The recent IEEE-1934 standard describes a reference architecture of Fog Computing.

From the research point of view, fog computing opens up several challenges, for example, see [7] for a discussion. An open research topic in this field is the design of resource sharing protocols between a set of distributed fog nodes, which are not coordinated by any central entity. The design of these algorithms should consider that the fog nodes can have different execution speeds and that the time budget available for the distribution of the tasks is less than rt-rtt. One challenging point is that the execution time of a task can become comparable with the time required to transfer the task from the origin node that received it on another node and, therefore, should be used with judgment.

In this paper, we propose a distributed resource sharing protocol designed for time-sensitive applications running on fog computing. The protocol, dubbed $LL_G(\mathbf{T})$ allocates units of jobs, or task, which are received from a set of distributed nodes of different G types, on the estimated least loaded node in the set. The type of nodes captures the execution speed of the node. The least loaded node estimation consists of the random selection of a single node in the set, and the comparison among the current execution speeds of such a remote node and the original node that has received the task from a user. This algorithm is the very cost-effective advantageous principle of the class of protocols based on the random choice of power-of-two, cite [8]. The proposed algorithm is designed to take into consideration the peculiarities of the particle deployment environment by controlling: (i) when a task can be moved from the source fog node that receives the task to another node, via a threshold T, which may depend on from the type, and (ii) through a more precise metric of the workload of the current fog node, based on the relative execution speeds between the origin and the remote fog node that was selected for a possible delivery of the activities. The protocol is studied by exploiting an asymptotic mathematical model, valid for a large set $N \to \infty$ of fog nodes. This large-scale model, although motivated by its tractability, can adapt to realistic settings, ie those based on the 5G F-RAN, characterized by the communication of BBUs (Base Band Units) associated with small radio access points, cite [4]. Furthermore, a smaller model is likely to show similar scale performance, [9]. To realistically model a single fog node, we conducted an experimental study on an implementation of a hypothetical image recognition service provided based on the FaaS model (Function as a Service), [10], based on Docker.

The rest of this paper is organized as following. Section II discusses the related work. In Section III we report the experimental evaluation of the FaaS service and a validation its model. Section IV defines the protocol, which is analyzed in Section V. Initial numerical results are given in Section VI.

II. Related work

The studies that characterize this randomization algorithm in the context of load balancing can be divided in two bodies depending on the model of the workers, that either cannot or can loss jobs. The first set of results, e.g., [8], [11], [12], model workers as M/M/1 queues and the selection criteria is to pick the Shortest among the d sampled Queues, $SQ(d)$. The most relevant finding is that the average job execution delay decay doubly exponentially in the limit as the number of servers goes to infinity, which is a substantial improvement over a classical queue case, where the queue size decays exponentially. A finer workload measurement is used in in [13]. This extraordinary improvement has motivated the research on a different worker model, that can somehow fit better application to cloud computing, e.g., [14]. The second body of works, in fact model the worker as a finite set of K servers, e.g., an M/M/K queue, and a job that cannot be scheduled it is blocked, [15], [14], [16], [12]. The selection criteria here is to pick the Least Loaded among d queues, $LL(d)$, where the server load is the number of busy servers. These works analyze the effect of randomization on the blocking probability, which is the counterpart of average delay. However, all models assume a single centralized dispatcher, and hence threshold based control cannot be applied. The work in [17] is the most one related to our idea. In the studied protocol if a job execution request arrives at an overloaded fog node, the job is forwarded to a neighboring node with some probability. Hence, contrary to our protocol no power-of-choice is used. In [9] we have used power-of-random choices to enable cooperation among different fog providers. No thresholds are used and the protocol is limited to unitary fanout. In [18] random choices are used for a p2p load balancing protocols.

Load balancing for the fog model has been studied in several papers. In [19] an algorithm called Multi-tenant Load Distribution Algorithm for Fog Environments (MtLDF) has been proposed to optimize load balancing in Fogs environments considering specific multi-tenancy requirements. However, the proposed load balancing scheme adopts a centralized fog management layer that receives all the state information about the fog nodes.

In [20], the tasks that the nodes are called to complete, are characterized according to their computational nature and are subsequently allocated to the appropriate host. Edge networks communicate through a brokering system with IoT systems in an asynchronous way via the Pub/Sub messaging pattern. However, a centralized workload balancer is used in the solution.

In [21] an approach is presented to periodically distributing incoming tasks in the edge computing network so that the number of tasks, which can be processed in the edge computing network, is increased, and the quality of-service (QoS) requirements of the tasks completed in the edge computing network are satisfied. The model however assumes that a set of tasks to be assigned is available, i.e., the tasks are not processed on line.

III. Modeling Function as a Service

This section reports an experimental evaluation of an implementation of a fog node that performs a hypothetical image classification service, which is delivered based on the delivery model Function as a service (FaaS), [10]. We use experimental data to validate a simple mathematical model of the fog node service, which is the basis for studying the proposed resource sharing algorithm.

The visual service is implemented using openFaaS [22] and Docker. The visual service consists of calling up a function that hosts a classification algorithm that receives the image to be classified as a parameter. Each invocation service sets up a Docker container, and the visual service allows openFaaS to create up to K concurrent containers. The underlying Linux kernel runs any container in a separated thread. When this limit is reached, further requests are dropped. The reason for this hard limit is to guarantee a minimum execution speed to requests that are accepted and hence a time bound on their response time. Since no priorities are used and threads do not communicate, this image classification service running on the fog node is modeled as an M/M/1/K-PS (Processor Sharing) queue. The assumption of the arrival Poisson process is a common convenient way used in similar works, eg. [23].

We have conducted several experiments using the Pigo face recognition library, openFasS, Docker on a 2.60 GHz, 2 cores Celeron CPU with 4MB cache and 4 GB RAM. Images were taken from [24]. The visual service runs in a VM allocated to a single core. One single face recognition task, took about 300 ms if executed alone. Image classifications are requested according to a Poisson random process by a client program executed on another VM, which was allocated on the other core of the physical machine. The visual service is exposed as a HTTP endpoint.

The stationary distribution of this queue is well known, [25]:

$$p_k = \frac{1-\rho}{1-\rho^{K+1}}\rho_k \quad k = 0, 1, \dots, K$$

where $\rho = \frac{\lambda}{\mu}$ is the *traffic intensity*. The *blocking probability* namely the probability that a new task cannot be executed is:

$$p_b = \frac{\rho^K - \rho^{K+1}}{1-\rho^{K+1}}$$

The mean number of tasks running is:

$$m = \sum_{k=1}^{K} kp_k = \frac{\rho}{1-\rho} - \frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}}$$

So that from the Little's result, the *average execution* time of a task is

$$d_e = \frac{m}{\lambda(1-p_b)}$$

Note that this model is insensitive to the service time distribution, but rather the above measures depends only on its mean, [25].

Figure 1 and 2 show the average execution delay and the blocking probability measured using the described setting with $K = 10$ threads and the one computed from the model. This delay is computed as the time elapsed from when the client starts sending the image until the classification text is received from the service, in the form of a short JSON object. The blocking probability is the fraction of images that are not accepted by the service.

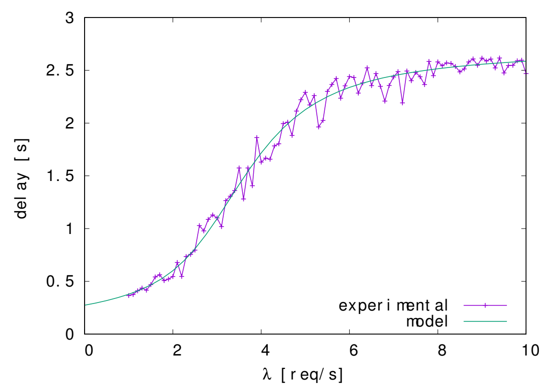

Figure 1: Average execution time vs load, experimental data and theoretical model.

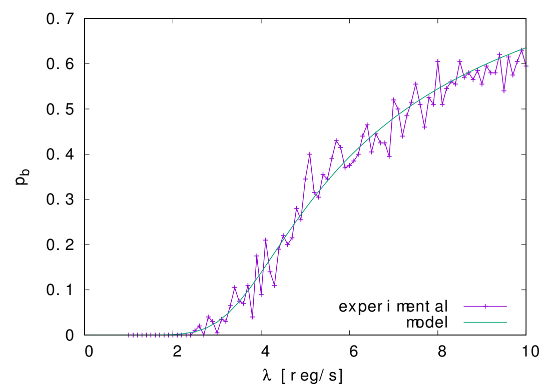

Figure 2: Blocking probability vs load, experimental data and theoretical model.

We can see that the deployed architecture is well captured by the model. Similar correspondence was observed for other parameters.

IV. Proposed Protocol

This section describes $LL_G(\mathbf{T})$, Least Loaded with thresholds $\mathbf{T} = (T_1 \ldots T_G)$ among G types, a resource sharing distributed algorithm for load balancing among a set of G types of nodes. The algorithm is designed for allocation of independent unit of executions, or *tasks* i.e., functions in the FaaS parliance, as described in the previous section. In particular, the protocol assumes that the nodes running the algorithm offers the same service, the image detection of our previous example. A similar scenario is envisioned for example in [23]. The *type* of a node is determined by the *nominal* execution performance of the node, and the maximum number K of admitted concurrent tasks. The execution performance of a node is defined as the service time of a task running at full speed on the node, i.e., without any other concurrent tasks. Using the previous model $c = \frac{1}{\mu}$. The key innovative parts of the protocol are designed to cope with the fact that the execution time of a task is not negligible compared to the transfer time of the task from one node to another. They are:

- Nodes of type i use a T_i threshold to adjust when trying to remotely schedule received tasks, on a less-loaded node; this regulation allows the penalty for the delay in sending the tasks to be taken into account
- The remote execution decision is taken on the basis of a weighted measure of the current relative execution speeds between the sending and receiving fog node
- Direct forwarding of tasks, consisting in sending the task directly to the possible remote node without using the control probe messaging; if required, the task is bounced back to the sending node

Figure 3 shows the flow diagram of the proposed protocol. When a node receives a task, one of the following cases can occur, depending on the nature of the task J. This task can be a:

1) A task produced by a client connected to the fog node. If the current number k of tasks running at the node is less than the threshold T associated to the node, J is executed immediately (*local task*); otherwise, the node forwards J to a randomly selected node, along with the pair (k, c)
2) A task forwarded by another fog node (*remote task*). In this case, if the current speed of the node is greater than that of the sending node, then the task is executed; otherwise, it is bounded on the sending node
3) a rejected task previously submitted to another node. In this case, if the node has not reached the limit, the node starts executing the task (*bounced task*); otherwise, the task is discarded and *blocked*

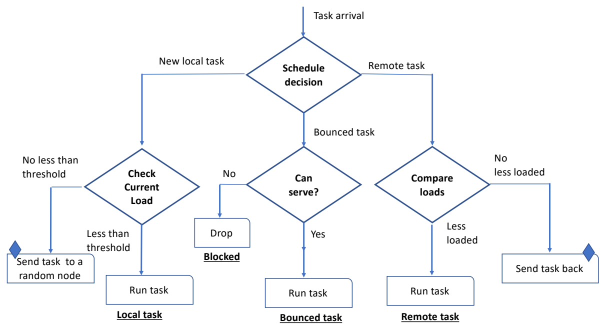

Figure 3: Schedule decisions at a fog node.

a) Speed comparison: Due to processor sharing, a task with nominal service time c executed on a node with other

$k-1$ tasks running lasts for kc. Let i and j two nodes and let c_i and c_j be the nominal service time of the same task, and k_i-1, k_j-1 the number of tasks that run on the two nodes. When:

$$c_i k_i = c_j k_j$$

the service time of the task will be the same on the two nodes, and clearly the task runs faster on i than a node j if:

$$k_i < \frac{c_j k_j}{c_i}$$

Let:

$$k_{ij} = \begin{cases} \left\lfloor k_j \frac{c_j}{c_i} \right\rfloor - 1 & \text{if } c_j k_j = c_i k_j \\ \\ \left\lfloor k_j \frac{c_j}{c_i} \right\rfloor & \text{otherwise} \end{cases}$$

The condition to determine if a task runs faster on a node i than on a node j is:

$$k_i \leq k_{ij}$$

V. Protocol Analysis

In this section we present an approximated asymptotic analysis of the proposed protocol. The definition of the main symbols used are reported in table I. To analyze the protocol we consider a large number N of nodes, i.e., $N \to \infty$, partitioned into G groups, where a group is composed of nodes of the same type. We consider the case of nodes executing the $LL_G(\mathbf{T})$ protocol with zero communication delay, type g nodes subject to a Poisson flow with rate λ_g, and nominal average service time exponentially distributed with mean $c_g = \frac{1}{\mu_i}$, Furthermore, we *assume* that this population of nodes has a unique steady state configuration. A formal proof of this statement is left as future work. By this we mean that the average fraction of nodes of a given type g and in a given state k (k being the number of tasks running) doesn't change over time. This is the point of view of the mean field analysis [27]. Let π_{gk} be the fraction of type g nodes in the state k, and $\overline{\pi}_{gk}$ the fraction of nodes with state at least k. Note that $\overline{\pi}_{g0} = 1$. Define $p_{s_{ij}}$ the probability that a node of type i selects a node of type j. This probability reflects the relative composition of the population. For example, $p_{s_{ij}} = \frac{1}{G}$ means that the number of types in the system is the same. The steady state composition of the system, given that it exits, should satisfy the following set of equations, $1 \leq g \leq G$:

$$(\lambda_g + \sum_j \lambda_j p_{s_{jg}} \overline{\pi}_{j max\{k_{ij}+1, T_j\}}) \pi_{gk} = \mu_g \pi_{gk+1}$$
$$k < T_g$$
$$(\lambda_g \sum_j p_{s_{jg}} \overline{\pi}_{jk+1} + \sum_j \lambda_j p_{s_{jg}} \overline{\pi}_{j max\{k_{ij}+1, T_j\}}) \pi_{gk} =$$
$$\mu_g \pi_{gk+1}$$
$$k \geq T_g$$
$$\sum_k \pi_{gk} = 1$$

Each of the above equation is a balance equation requiring that the fraction of type g nodes that increases the state from k to $k+1$ must be equal to the fraction of nodes that decreases their state from $k+1$ to k, so that the fraction of nodes in the state k doesn't change. An equivalent and more convenient interpretation, is that for any g the equations describe the dynamic of the state of a tagged node of type g. This set of equations then describes the dynamic of G nodes, one for each type. The solution of the above system is found numerically.

Symbol	Definition
G	number of type of nodes
c	nominal execution speed
π_{gk}	probability that the state of a type g node is k
$\overline{\pi}_{gk}$	Tail distribution, $\sum_{i=k}^{K} \pi_{gi}$
$p_{s_{ij}}$	probability that a type i node selets a type j node

Table I: Definition of the main symbols

A. Metrics

Once the solution of the system composition is in our hands, one can compute the probabilities associated to how a received task is processed. This computation is approximated by assuming that the state of the two nodes (local and remote) doesn't change during task forwarding (the communication delay among these two nodes is zero).

A task received by a type g node may undergo to one of the following accidents, see also Figure 3.

- *executed locally*, i.e., from the received node without any attempt to schedule task execution elsewhere; as this occurs if the state of the node is less than the threshold T_g, the probability of this event is:

$$p_{l_g} = 1 - \overline{\pi}_{gT_g}$$

- *forwarded* to a type i remote node and executed by that node; this occurs when the current state of the node is $k \geq T_g$ and on the remote node the task runs faster. The probability of this event is:

$$p_{f_{gi}} = p_{s_{gi}} \sum_{k=T_g}^{K_g} \pi_{gk} (1 - \overline{\pi}_{ik_{ig}})$$

- *bounced back* by the selected node and executed locally; this occurs when the current state of the node that received the task is $T_g \leq k < K_g$ and on the remote node the task runs no faster than on the local node. The probability of this event is:

$$p_{bb_{gi}} = p_{s_{gi}} \sum_{k=T_g}^{K_g-1} \pi_{gk} \overline{\pi}_{ik_{ig}}$$

- *blocked*. This occurs when both the local node and the remote one have already the maximum number of tasks running. The probability of this event is:

$$p_{b_g} = \pi_{gK_g} \sum_{i=1}^{G} p_{s_{ig}} \overline{\pi}_{gk_i}$$

B. Delay analysis

We now analyze the average total execution delay of a task. This delay is the time elapsed from when a task is received by a fog node, until the task is received from the client. Again, we follow an approximation of the computation of the delay components, but we gain in terms of a much simpler framework. The key idea is to focus on a single *tagged* fog node and applying the Little's result to two subparts of the node once they are recognized as queuing systems. The result allows to compute the average delay in a queuing system, given the input rate in the system and the average number of tasks in the system - see Figure 4. Recall that π_{gk} is also the probability that the state of the tagged type g node is k.

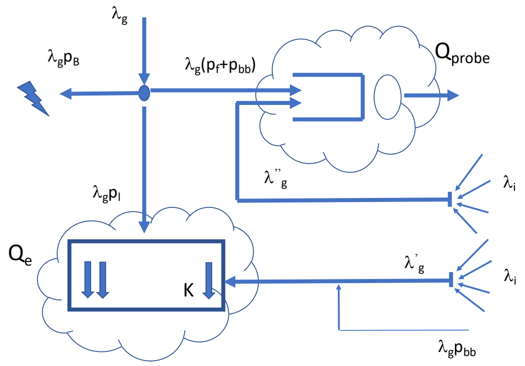

Figure 4: Delay model.

1) Execution delay: This component of the delay corresponds to subsystem Q_e of Figure 4, that models the execution engine. A fog node of type g executes tasks that are received from its own clients (at rate λ_g) either because when they are received the state of the node is less than T_g (local task) or because they are rejected from the selected node (bounced task), see Figure 3. This flow has then arrival rate $\lambda_g(p_{l_g}+p_{bb_g})$. The node also executes tasks on behalf of other remote nodes. The arrival rate of these tasks is:

$$\lambda'_g = \sum_i \lambda_i \sum_{k=T_i}^{K_i} \pi_{gk} \sum_{i=1}^{G} p_{s_{gi}}(1-\overline{\pi}_{ik_{gi}})$$

which corresponds to a node i with $k < T_g$ tasks running which selects our tagged node. By exploiting the Little's result, the execution delay of a task running on a type g node is then:

$$d_{e_g} = \frac{\sum_k k\pi_{gk}}{\lambda_g(p_{l_g}+p_{bb_g})+\lambda'_g}$$

2) Task forwarding delay: Consider now the flow corresponding to task forwarding, i.e., subsystem Q_{probe} in Figure 4. For taking the communication delay into account, we model the transmission functionalities as an $M/D/1$ queue, where the service time is the time required to send a task to a remote node, assumed constant and denoted by α. For the sake of simplicity, the delay associated to sending the result of task execution back to the node is not considered, although it could easily added to our model. The reason is that the size of the result is very small, e.g., a short JSON object. Furthermore, the flows entering this system are assumed to be Poisson flows. The tasks that a type g node sends are the sum of those received directly by the node and sent to other nodes (whose fraction is $p_{f_g}+p_{bb_g}$ of those received), plus those that the node receives from other nodes but that are bounced back. It is easy to see that:

$$\lambda''_g = \sum_{i=1}^{G} \lambda_i p_{s_{ig}} p_{bb_{ig}}$$

From the well kwon M/D/1 queue's waiting time we can compute the average time d_{t_g} required by the tagged node to forward a task:

$$d_{t_g} = \frac{1}{2}\frac{1}{\frac{1}{\alpha}-[\lambda_g(p_{bb_g}+p_{f_g})+\lambda''_g]}$$

3) Total delay: The overall delay of a task received by a type g node, d_g, is obtained as the weighted sum of the execution delays as follows:

$$\begin{aligned} d_g =& (p_{lg}+p_{bb_g})d_{e_g} + \sum_{i=1}^{G} p_{f_{gi}} d_{e_i} + \\ & (p_{f_g}+p_{bb_g})d_{t_g} + \sum_{i=1}^{G} p_{s_{gi}} p_{bb_i} d_{t_i} \end{aligned}$$

VI. Results

In this section we report some numerical result obtained from our previous model.

A. Effect of randomization

As a first step, we consider only one type of nodes, negligible communication delay and no threshold, $G=1, \alpha=0, T=1$. This setting is a sanity check that allows to study the effect of randomization on the performance of an ideal service among nodes that run $LL_1(1)$.

Figure 5 shows the blocking probability as a function of K. For this case, the solution of the model only depends on traffic intensity, $\rho=\frac{\lambda}{\mu}$. We can see that for $\rho > 1$, the the blocking probability decreases with K but eventually remains a positive quantity. The reason is simply because a node receives a net flow $\lambda > \mu$ and the queue length grows unboundedly. Consider now $\rho < 1$. For a given ρ, there is a value K^* after which the blocking probability falls sharply (in our experiments it is evident that $K^*=5, 11$). This is a distinguish property of the underlying randomization algorithm, that has been remarked in [14], [28]. In practical terms, this result may help in designing a FaaS service. If the traffic intensity is known, there is no reason for example to pre-warm docker instances beyond K^*, as this will not allow to accomodate new tasks in an appreciable way.

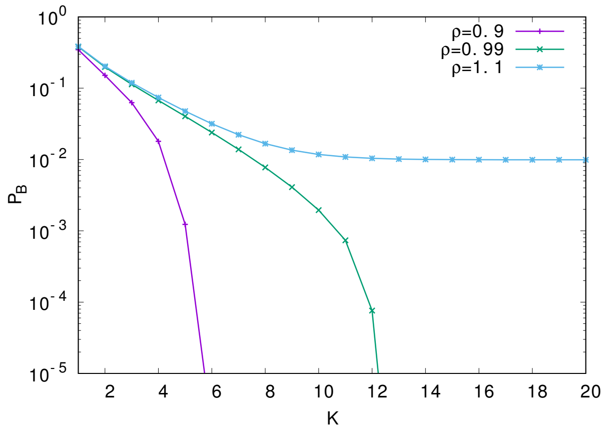

Figure 5: Blocking probability service performance as a function of K for nodes running $LL_1(1)$.

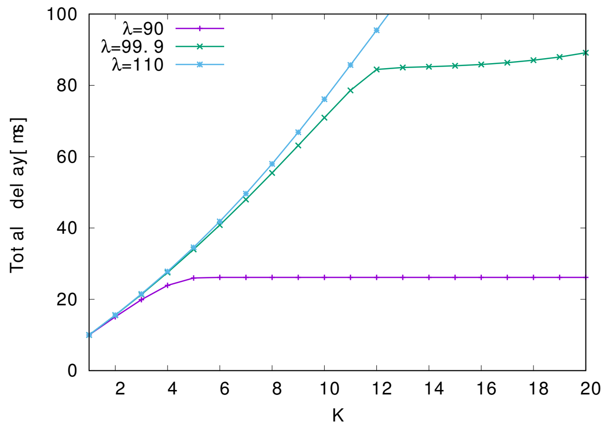

Figure 6: Delay service performance as a function of K for different loads for nodes running $LL_1(1)$. The nominal service time is $10ms$.

Figure 6 shows the total delay as a function of K. In this results we have set the nominal service time c to $10ms$. Note that as $\alpha = 0$, this delay corresponds to d_e. The shape of the delay is determined by the blocking probability. For $\rho < 1$, the delay increases up to K^* and then it remains a constant. The reason is that once K^* is reached, practically all tasks are served [1].

Figure 7 shows a comparison among the delays of an isolated fog node implementing a FaaS service and an hypothetical set of fog nodes that share their computation resources according to $LL_1(1)$. We see here a typical consequence of the proposed protocol. As the protocol allows to reduce the blocking probability, a fog node under $LL_1(1)$ serves more tasks. Hence, the execution speed seen by a task decreases and the average delay of a task is higher than if the protocol were not used. Overall, the proposed protocol allows to increase the number of served task at the cost of an increased delay. For example, for $\lambda = 99.9$, a node blocks 7.6% of the incoming tasks, whereas under $LL_1(1)$ only a negligible fraction of tasks are blocked, less than 10^{-4}, see also Figure 5.

[1]We provide a practical interpretation of the results, avoiding other more sophisticated analysis in the limit of $\rho \to 1$, e.g.. see [14].

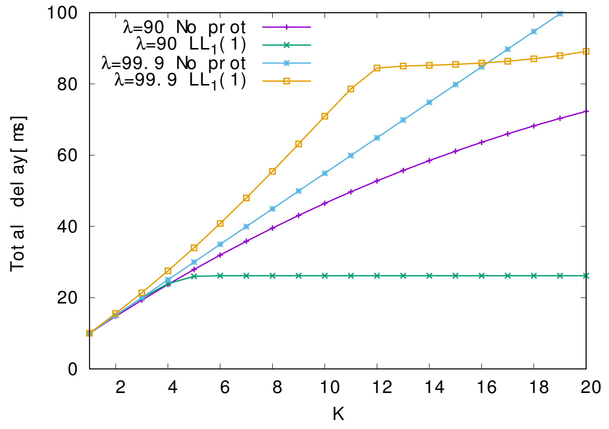

Figure 7: Comparison among the delay service performance as a function of K for two different load conditions. The nominal service time is $10ms$.

For $\rho > 1$ each task that is able to enter the system, is served at a speed $\frac{c}{K}$ (where $c = 10ms$). The delay is in fact proportional to K.

B. *Effect of the threshold on randomization*

The second set of experiments aims at studying how the threshold T mitigates the beneficial effect of randomization. A randomization protocol has its root in the capability of scheduling (transferring in our model) a task on (to) a least loaded node as the the key mechanism to achieve load balancing. However, task forwarding adds a delay to the response time, so it is worth to reduce task forwarding as much as possible without affecting the effectiveness of the underlying randomization effect. Again, we first consider the case of $\alpha = 0$. Figure 8 reports a breakdown of where an $LL_1(1)$ protocol allocate the tasks for $\lambda = 90$. The plot shows the fraction of tasks that are directly executed (because the threshold is not reached), the fraction that are executed remotely and those that are bounced back from the selected node. Figure 10 shows a similar breakdown for $LL_1(2)$, i.e., when the threshold is increased of just one unit.

We can see that these percentages are *extremely* sensible to the threshold as it is moved from the minimum value $T = 1$ to just $T = 2$.

C. *The role of the communication delay*

Randomization algorithms are usually analyzed considering immediate task transfer time. In a fog computing setting where the task corresponds to a simple function and the delay requirements can be very stringent, the execution time of a task is expected to be small, e.g., order of ms. The task transfer time can then be comparable with the execution time, so it is interesting to check the role of this delay on the final delay performance.

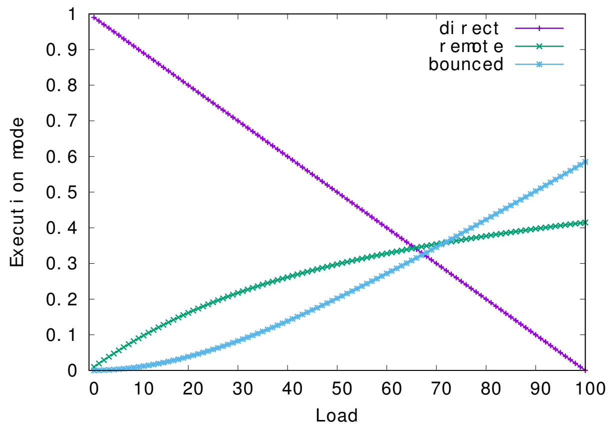

Figure 8: Execution mode of a task $LL_1(1)$.

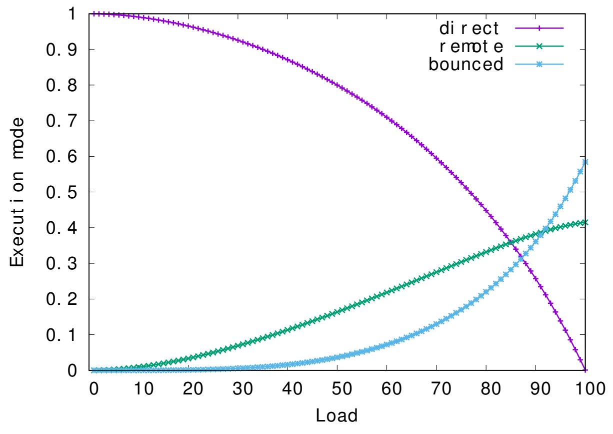

Figure 9: Execution mode of a task $LL_1(2)$.

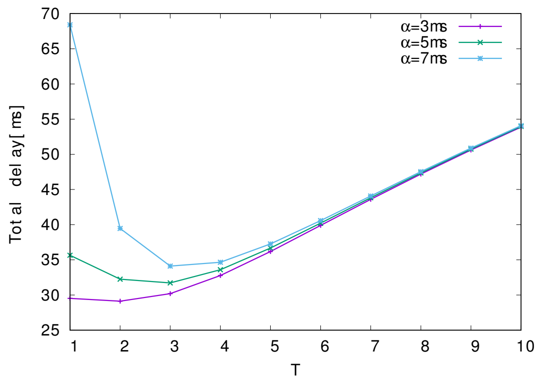

Figure 10: Total delay as a function of the threshold, $\lambda = 90, K = 10, c = 10ms$.

Figure 10 shows the total delay as a function of the threshold, for $\lambda = 90, K = 10, c = 10ms$. As the α is increased, the delay becomes very sensible to the threshold which is a consequence of the change in the task breakdown execution location.

Figure 11 and Figure 12 show the total delay for the whole spectrum of loads for two values α. When $\alpha = 3ms$, $LL_1(1)$ provides the best performance, whereas as α is increased, the best setting depends on the load and it is either $LL_1(2)$ or $LL_1(3)$.

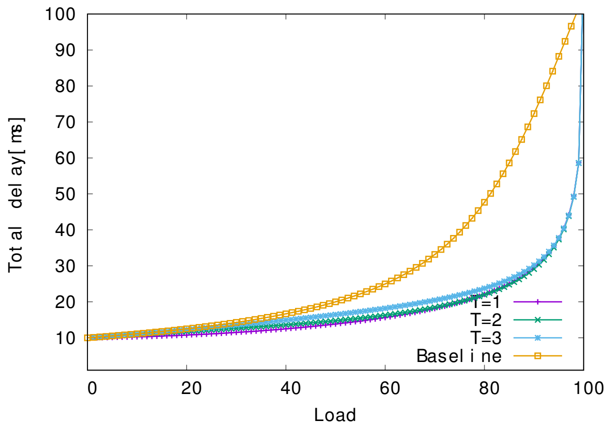

Figure 11: Average execution time vs load, $\alpha = 3ms$.

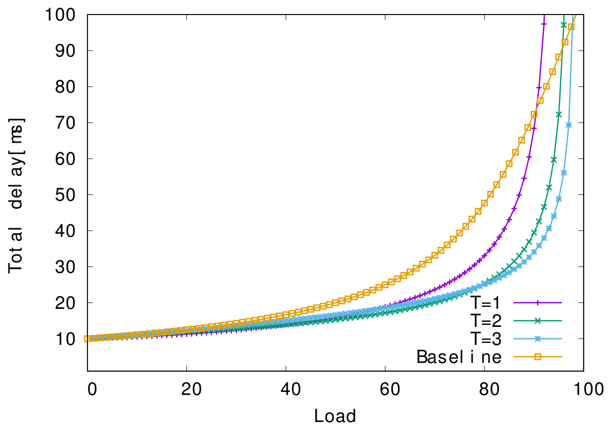

Figure 12: Average execution time vs load, $\alpha = 7ms$.

D. Coping with different execution speeds.

Let now consider the case of two type of nodes with different execution speeds. We consider $\alpha = 3ms$ and $T_1 = T_2 = 1$, i.e., the nodes run an $LL_2(\mathbf{1})$ protocol. Both type of nodes have $K = 10$. However, a type 1 node executes a task at the nominal speed of $10ms$, whereas type 2 nodes have nominal speed of $15ms$. To assure system stability we set $\rho_1 = \rho_2 = 0.99$, so that $\lambda_1 = 99.9, \lambda_2 = 66.6$.

Let γ be the 'fraction' of type-1 nodes, i.e., a type-1 node is selected with probability $p_{s_{11}} = p_{s_{21}} = \gamma$. Figure 13 shows the total delay for both nodes as a function of γ.

A type-2 node, i.e., a slower node, is expected to have an improvement when forwarding one of its task to a faster type-1 node, so increasing γ is expected to decrease the overall delay d_2. Figure 13 shows that this is not the case. The reason is that the traffic of type-1 is much higher than the one directed to type-2 nodes. Hence, a small presence of type-1 nodes means the possibility for type-2 nodes to receive a much higher task to execute, hence resulting in a higher node occupancy. The net effect shown in the Figure 13 is that d_2 increases with γ. After a given maximum, d_2 decreases. This can be explained with the fact that now a type-1 node has a lower changes of selecting a type-2 node.

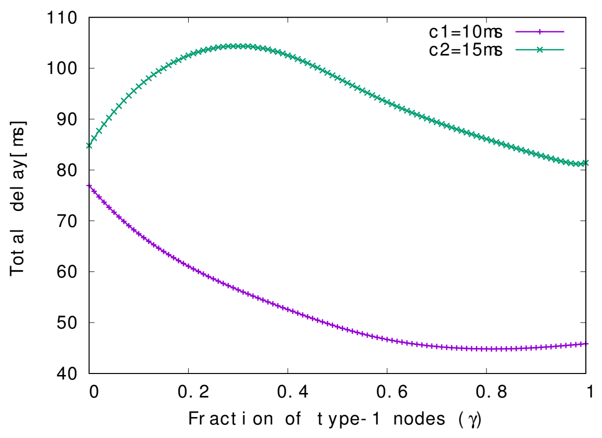

Figure 13: Service performance as a function of type-1 node percentage.

The study of higher G is left as a future work.

VII. Conclusions

This paper proposes a distributed resource sharing protocol designed for time-sensitive applications running on fog computing characterized by different execution speeds and not negligible communication delay. The protocol allows to determine when a task can be moved from the origin fog node that receives the task to another node where it can be executed faster, via a threshold T. Numerical results show that the protocol can remarkably reduce the average response time w.r.t. a no resource sharing nodes. As a future work, we plan to study the protocol under higher G and using a more precice metric to measure the convenience for remote task execution that takes K and the nominal service time into account.

References

[1] T. Kämäräinen, M. Siekkinen, A. Ylä-Jääski, W. Zhang, and P. Hui, "A measurement study on achieving imperceptible latency in mobile cloud gaming," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, ser. MMSys'17, 2017.

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

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

[4] G. A. W. Group, "View on 5g architecture," 2018.

[5] A. Mtibaa, K. A. Harras, and A. Fahim, "Towards computational offloading in mobile device clouds," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1. IEEE, 2013, pp. 331–338.

[6] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

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

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

[9] R. Beraldi, H. Alnuweiri, and A. Mtibaa, "A power-of-two choices based algorithm for fog computing," *IEEE Transactions on Cloud Computing (Early access)*, 2018.

[10] e. a. E. Jonas, J. Schleier-Smith, "Cloud programming simplified: A berkeley view on serverless computing," in *arXiv:1902.03383*. arXvi, 2019.

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

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

[13] B. V. H. Tim Hellemans, "On the power-of-d-choices with least loaded server selection," *arXiv:1802.05420*, 2018.

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

[15] S. R. E. Turner, "Resource pooling in stochastic networks," in *PhD dissertation*. University of Cambridge, 1996.

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

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

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

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

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

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

[22] "https://www.openfaas.com/."

[23] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," 2019.

[24] "http://www.face-rec.org/databases/."

[25] F. Kelly, "Reversibility and stochastic networks." Wiley, 1979.

[26] T. Vasantam, A. Mukhopadhyay, and R. Mazumdar, "Insensitivity of the mean-field limit of loss systems under power-of-d routing," *CoRR*, vol. abs/1708.09328, 2017.

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

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