This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

A Power-of-Two Choices Based Algorithm for Fog Computing

Roberto Beraldi, Hussein Alnuweiri, *Senior Member IEEE,* and Abderrahmen Mtibaa

Abstract—The fog computing paradigm brings together storage, communication, and computation resources closer to users' end-devices. Therefore, fog servers are deployed at the edge of the network, offering low latency access to users. With the expansion of such fog computing services, different providers will be able to deploy multiple resources within a restricted geographical proximity.

In this paper, we investigate an incentive-based cooperation scheme across fog providers. We propose a distributed cooperative algorithm amongst fog computing providers where fully collaborative fog nodes are subject to different loads. The proposed algorithm leverages the power-of-two result and exploits a cooperation probability, namely the probability that a given provider collaborates by accepting a computation request from another provider, as a mean to achieve a fair cooperation.

We adopt an analytical approach based on exploiting a simplified performance model to demonstrate numerically that a set of optimal accepting probabilities exits when the number of server nodes goes to infinity. This result then drives the design of our distributed algorithm. Second, in our experimental approach, we perform a set of simulation analysis to verify the validity of the proposed solution when the number of servers is limited.

1 Introduction

The rapid rise of the Internet of Things (IoT) as the main connectivity medium for billions of industrial sensors, smart home appliances, and consumer wearable devices, is paving the way for novel communication and computation paradigms that can help scale such unexpected new communications [1], [2]. The resulting IoT data volume is expected to overwhelm the network bandwidth, the storage systems, the compute resources, and the analytics services. Current Cloud computing services have shown limitations in dealing with such scale of data and compute resources [3], [4].

Fog or Edge computing were proposed to address the needs to bring the computation closer to the edge of the network taking advantage of the shorter round trip time (RTT) delays to reduce operational. Fog computing paradigm aims at decreasing network congestion while increasing the Quality of Experience (QoE)–faster analysis and shorter computation delays. Fog computing takes advantages of the shorter round trip time delays to minimize the operational cost [5], [6], limit the overwhelming of the core network [7], and reduce computation delays [8]. With the ever-increasing wireless and compute capabilities, offloading computation to edge nodes, called cloudlets [9], become an attractive and cheap solution than reaching a distant and often costly cloud.

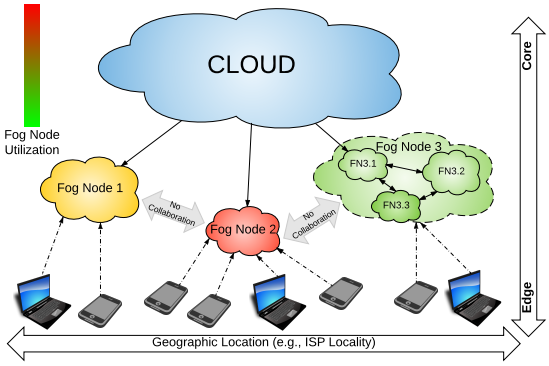


Fig. 1: A Fog computing scenario: fog nodes may have different utilization levels resulting on varying performances (green indicates low and red indicates high utilization levels).

While traditional uncoordinated compute resources have been deployed as operational schemes for such fog computing platform, we consider a scenario of coordinated scheme, where such compute resources, called fog nodes, can collaborate by dedicating part of their resources to improve the overall system performance with respect to satisfying all computational requests as shown in Fig. 1. Collaboration between competitive entities is often restricted unless a credit based incentive is ensured and applied in exchange for dedicating resources for foreign fog nodes. They simplify the coordination and the accounting, however in the context of fog computing, ensuring that all providers will be willing to participate in order to help each others is challenging and may not guarantee the system stability.

In this paper, we aim at ensuring a fair use of resources across all providers as a mean to increase the overall service utility. We leverage the power-of-two random choices result to design a simple and effective scheduling algorithm for collaborative fog computing. We consider a network consisting of N collaborative fog nodes belonging to different providers and each serving a Poisson flow of computation requests, or tasks. When a given task h reaches a fog node $u \in N$, u polls another node $v \neq u$ at random. The polled node decides to cooperate with a certain probability, which is tuned dynamically. In case of cooperation, if the polled node's service utilization U_v is lower than U_u, h is forwarded to node v. The design of the algorithm is inspired to the results of the analysis of an ideal system with infinitely

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

many nodes loaded with Poisson traffic whose mean can only assume a finite set of values. We first numerically show that in such a system the nodes can cooperate in a fair and optimal way, i.e., they can set their cooperation probabilities in such a way that on the average a node receives the same amount of CPU cycles that the node borrows to other providers. We then propose an algorithm to tune the cooperation probabilities dynamically. Our simulation results show that the tuning algorithm converges and correctly reacts to changes in the loads. In addition, a provider can get the same performance that it would obtain if the resources of the provider were increased of 30 %.

The paper is organized as follows. Section 2 reports background and the state-of-the-art of fog computing and the power of two choice models. Section 3 gives a detailed overview of our system model and the corresponding performance model. In Section 4, we propose an adaptive cooperative model where we discuss the tuning of the cooperative probabilities. Section 5 discusses the design of our proposed protocol, LOCAL. Before concluding this paper in Section 7, we provide the experimental results and discuss the performance of our proposed scheme in Section 6.

2 Background and related work

Cloud Computing is a new term for a long-held dream of computing as a utility [10], which has recently emerged as a commercial reality. Cloud computing providers build huge data centers from where they deliver computation power and other abstractions as a Service to the end users. While this approach suites many companies it is less appropriate for the new emerging needs raising from mobile devices and smart objects. Due to their limited resource availability, new support is required in terms of capacity to execute computation on-demand with little delay and jitter [11].

Fog computing, in its simplest concept, can be seen as an evolution of the current cloud model, where cloud functionalities are made available closer to the end user, i.e., to the edge of the network. This model can for example support the Internet of Things (IoT) concept, in which most of the devices used by humans on a daily basis will be connected to each other, [12]. Recently, the fog computing model has also been used in the context of Wireless Sensor Networks (WSN) as a support to a global monitoring capability for tracing moving sensors and detecting malicious ones [13]. A discussion about the flexibility of Fog Computing can be found in [14]. In addition, a broader definition of the fog concept is discussed in [15] and a survey can be found in [16].

As far as our work is concerned, we register that Fog, Edge, and various other solutions have been proposed to offload computation to powerful surrogate machines [17], [18], [19] such in CloneCloud [20] and MAUI Project [21], or smaller computational resources such as cloudlets [22] which reside closer to edge devices. Research for fog computing have been mainly focusing on building platforms and prototypes to make the case for such novel computing paradigms. Little research, to the best of our knowledge, have modeled interaction between distributed fog nodes [23], especially cooperation and incentives models for fog and edge computing [24].

In this work, we consider the power-of-two result as a mean to randomize load balancing across different fog nodes in the network [25], [26]. Load balancing is the process of scheduling jobs among a set of N physical servers. In general, each server can run up to C jobs in parallel and has a waiting area of capacity B jobs. The power of two (in general d) random choices is a term used to describe the improvement in the performance of a stochastic process characterized by a random decision that has to be taken. By allowing the inspection of d possible options, instead that just one, a substantial improvement in the performance of the process is observed, with the largest improvement seen when passing from $d = 1$ to $d = 2$. This phenomena has been analyzed in different communities and reader can refer to [25] for a broad survey.

In Shortest Queue among d (SQ(d)) scheduling discipline, the scheduler picks d out N nodes at random and schedules the job the the lowest loaded server, with ties broken at random. SQ(d) has been studied in the seminal paper [26] where each server is modeled as an M/M/1 queue, i.e., $C = 1, B = \infty$. The proposed supermarket model describes the dynamics of the system as the fraction of servers with at least k jobs and it is formulated as a set of differential equations. The key result is that the average delay seen by a job entering the system decays doubly exponentially in the limit of $N \to \infty$ and $\lambda \to 1^-$, compared to an isolated queue size, where the delay decays exponentially. An important result that guided our model is that, for $N \to \infty$, the dynamic of queues becomes asymptotic independent from each other, e.g., [27]. More precisely, for a randomized load balancing scheme in equilibrium, any fixed number of queues become independent of one another as $N \to \infty$. Authors in [28] studied a system with C finite, i.e., jobs can be discarded. Authors provide upper bounds of the discard probability for when $N, C, \lambda \to \infty$. Another study [29] used SQ(d) for a finite set N of M/M/1 queues. They provide numerical bounds on the delay and show that the delay can be quite different w.r.t the limit case.

To the best of our knowledge, the closest work that investigates cooperation in fog computing, [30], proposes probabilistic forwarding of requests blocked at a small compute-limited data center to a bigger compute-capable server. While findings suggest that the proposed strategy can significantly improve blocking at a small data center without affecting the performance of the server, this work remains limited to cloudlet-cloud cooperation and does not investigate a fully cooperation among similar providers of fog nodes. Our paper aims at filling this gap and focuses on proposing a fair use of resources across all providers as a mean to increase the overall service utility.

3 System Model

We consider a set of N providers divided into G different groups and each managing a single fog node composed of C independent physical servers. We define, S_u^t the *state* of fog node u at time t, as the number of busy servers (i.e., on use) at time t, $S_u^t \in \{0, 1, ..C\}$. Thus, we define the workload of node u at time t, w_u^t, as the fraction of busy servers of a given node u, $w_u^t = S_u^t/C$. We assume that computation requests to fog nodes all follow a Poisson process whose mean can

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

N	Number of fog nodes
G	Number of groups $G = \|\Lambda\|$
C	Number of servers in a fog node
Λ	Set of different traffic intensities
λ_g	g-th traffic
ρ_g	Load for nodes in g, $\rho_g = \frac{\lambda_g}{C}$
ρ_{gE}	Out coming cross load from a random node in g to other nodes
ρ_{Eg}	Incoming cross load coming from other nodes to a random node in g
p_g	Cooperation probability for nodes with traffic λ_g
g	Group of nodes with same λ_g and p_g
PB_g	Blocking probability of a node in g
π_{gi}	Stationary probability that the state of a node in g is i
P_{gi}	Tail probability $\sum_{j\geq i}\pi_{gj}$
P'_{gi}	$= 1 - P_{gi}$

TABLE 1: Model terminology.

take value in the set $\Lambda = \{\lambda_1, \ldots, \lambda_G\}$, where $|\Lambda| = G \leq N$ and require an exponentially distributed service time with unitary mean. Table 1 summarizes the main notations and parameters used throughout our paper.

We call λ the traffic intensity, i.e., the average number of computation requests per unit of time of a node, and $\rho = \frac{\lambda}{C}$, i.e., the average traffic per server, its load. As the behavior of a node is only influenced by the value of its load, we assign all nodes subject to the same load to the same group g. Thus, for any node belonging to group g, has mean traffic λ_g and load ρ_g. For the sake of simplicity, we assume that the size of all groups is constant.

We are interested in deriving the probability that a new computation request (i.e., task) is not accepted by a fog node u due to idle compute resources' unavailability (i.e., S_u^t=C), namely the *blocking probability* (PB_u).

We derive the blocking probability for $N \to \infty$ and Λ finite, when providers cooperate in the following way: when a new request arrives at fog node $s \in g$, s polls another node, say $s' \in g'$, selected uniformly at random among all providers. s' decides to accept the request and cooperate with a probability $p_{g'}$ referred to as the *cooperation* probability. In case of cooperation, if $w_{s'}^t < w_s^t$ then s forwards the given request to s' while if $w_{s'}^t = w_s^t$ the task is sent with probability 0.5. The cooperation probability fog nodes of type g is p_g. We call $\mathbf{p} = (p_1, \ldots, p_G)$ the *cooperation probability* vector.

For $|\Lambda| = 1, p_1 = 1$, our model corresponds to the $SQ(2)$ scheduling discipline. For this case, in the limit of $N \to \infty$ nodes become statistically independent from each other and a unique equilibrium distribution for the state of a node exists [27]. Our analysis is based on the ansantz that is also true for $|\Lambda| > 1$ and any $\mathbf{p}$. As a consequence, in the limit of $N \to \infty$, all nodes with the same load λ_g have the same steady state probabilities.

We then focus on a tagged node s in the system, $s \in g$. Due to the independence among nodes, the dynamic of its state can be captured by a simple birth-death Markov Chain with state dependent birth rate, see Fig. 2. For this reason, we will refer to this model as Markov Chain model, MC for short.

Let π_{gj} be the steady state probability of such a node, $P_{gi} = \sum_{j\geq i}\pi_{gj}$, $P'_{gi} = 1 - P_{gi}$. The state transition rate from state i to $i-1$ is i, whereas from i to $i+1$ it is obtained

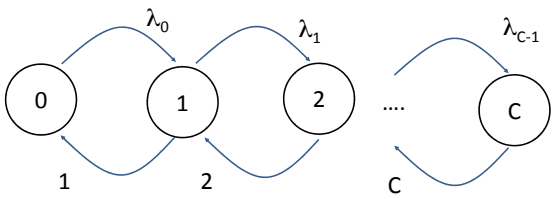


Fig. 2: Markov chain capturing the dynamic of the state of a node.

by inspecting all the possible cases that make the state of the node to increase.

3.1 Transition Probabilities

In an infinitesimal time interval $\delta t \to 0$, we identify the following exclusive compound events that make the state of s change from i to $i+1$: (C1) a new request arrives at s, or (C2) s is polled by another node s'.

The following cases may then occur:

C1-1: The polled server, s' has state $i+1$ and belongs to a group g'. Since the probability of belonging to group g' is $\frac{1}{G}$, and $P_{g'i+1}$ is the probability of s' having at least $i+1$ tasks running, hence:

$$\alpha_{1g}(i) = \frac{\lambda_g \pi_{gi}}{G} \sum_{g'=1}^{G} P_{g'i+1}$$

C1-2: s' belongs to g', its state $S_{s'}^t = S_s^t = i$ and ties are broken: (i) in favor of s (this occurs with probability $\frac{1}{2}$), or (ii) in favor of s', and s' does not cooperate (occurring with probability $\frac{1-p_{g'}}{2}$). Thus:

$$\begin{aligned}\alpha_{2g}(i) &= \frac{\lambda_g \pi_{gi}}{G} \sum_{g'=1}^{G} \Big(\frac{1}{2} + \frac{1-p_{g'}}{2}\Big)\pi_{g'i} \\ &= \frac{\lambda_g \pi_{gi}}{2G} \sum_{g'=1}^{G} (2-p_{g'})\pi_{g'i} \qquad (1)\end{aligned}$$

C1-3: s' belongs to a group g', its state $S_{s'}^t < i$, and s' does not cooperate. This event occurs with probability:

$$\alpha_{3g}(i) = \frac{\lambda_g \pi_{gi}}{G} \sum_{g'=1}^{G} P'_{g'i}(1-p_{g'})$$

The second category (C2) of events satisfying an increase of s's state from i to $i+1$, captures the cases when s is polled from another node s', subject to load $\lambda_{g'}$. Since the request flow from nodes in a group g' is $\lambda_{g'}\frac{N}{G}$ and the probability s' polls s is $\frac{1}{N}$, we have:

C2-1: $S_{s'}^t > i$, and s accepts to cooperate. This event occurs with probability:

$$\alpha_{4g}(i) = \frac{p_g \pi_{gi}}{G} \sum_{g'=1}^{G} \lambda_{g'} P_{g'i+1}$$

C2-2: $S_{s'}^t = S_s^t = i$, ties are broken in favor of s, and s cooperates. Thus:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

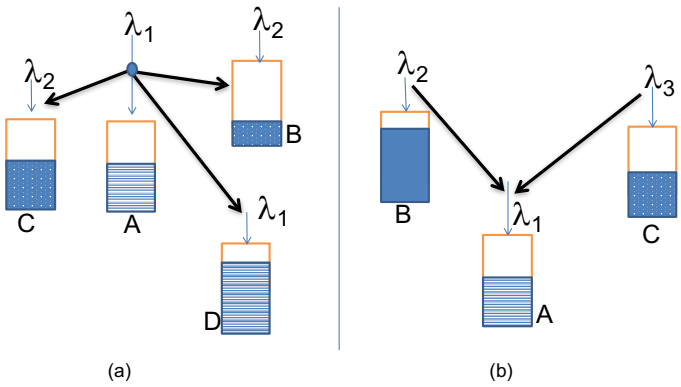


Fig. 3: Possible cases when the state of node A increases. Fog nodes are represented as rectangles, with the filled part representing its state (workload).

$$\alpha_{5g}(i) = \frac{p_g \pi_{gi}}{2G} \sum_{g'=1}^{G} \lambda_{g'} \pi_{g'i}$$

Therefore, any of the above probabilities, $\alpha_{kg}, 1 \leq k \leq 5$, represents a compound event of finding one node in a given state *and* another into another different state [1]. The transition rate of the tagged node from state i to state $i+1$ is then given by the sum of above probabilities divided by the probability to find s in state i (so that the result is a conditional probability). Hence, in order to compute the steady-state probabilities, π_{gi}, we can study a birth-death Markov process with state dependent transition rates λ_{gi} from i to $i+1$ given by:

$$\lambda_{gi} = \frac{\sum_j \alpha_{jg}(i)}{\pi_{gi}} \tag{2}$$

whereas the rates from state i to state $i-1$ are given by $\mu_{gi} = i$. For instance, in Fig. 3, (a) Node A polling three different nodes to increase its state; Node B has a lower utilization but it does not accept the computation request, node C has the same utilization but it does not accept the request, node D has a higher utilization. Nodes A, D belong to the same group (their load is the same). Nodes C, B form to another group. (b) Node A is polled by node B (whose utilization is higher) and by C (same utilization) and it accepts the task.

To compute the set of the steady state probabilities of all the nodes, we use standard flow balance equations:

$$\pi_{gi+1} = \frac{\lambda_{gi}}{i+1} \pi_{gi} \quad \forall g$$

with the normalized condition $\sum_i \pi_{gi} = 1$.

Thus, starting from g arbitrary normalized state distributions, $[\pi_{\mathbf{1}}^{\mathbf{0}}, \pi_{\mathbf{2}}^{\mathbf{0}}, \ldots, \pi_{\mathbf{g}}^{\mathbf{0}}]$, each representing the state of a node in each group ($\pi_{\mathbf{j}}^{\mathbf{0}} = [\pi_{j0}^{0}, \pi_{j1}^{0}, \ldots, \pi_{jC}^{0}]$), a set of transition rates is computed according to Eq. 2. These rates are then used to compute a new set of distributions $[\pi_{\mathbf{1}}^{\mathbf{1}}, \pi_{\mathbf{2}}^{\mathbf{1}}, \ldots, \pi_{\mathbf{g}}^{\mathbf{1}}]$, and so on, until a fixed point is reached for every node, i.e, $\pi_{\mathbf{i}}^{\mathbf{k}} = \pi_{\mathbf{i}}^{\mathbf{k+1}} = \pi_{\mathbf{i}}^{\infty}, \forall i$. Once the steady st ate probabilities for all nodes are computed, the probability of a given request arriving to node $s \in g$ is not served, namely the blocking probability, can be derived as following:

$$PB_g = \frac{\pi_{gC}}{G} \sum_{g'=1}^{G} \pi_{g'C} + (1 - p_{g'}) P'_{g'C}$$

Therefore, a given request is blocked at time t, if the requested node s is congested, i.e., its state $S_s^t = C$ and the polled node, say $s' \in g'$, is also congested and/or does not cooperate.

1. Conditions C1-2 and C2-2 could in principle be removed. However we keep them in order to adhere to the original result in [26].

$\mathcal{N}$	$C=5$	$C=10$	$C=20$	$C=35$
2	0.18	0.11	0.072	0.047
5	0.15	0.094	0.047	0.021
10	0.13	0.077	0.032	0.011
50	0.12	0.060	0.016	0.0019
100	0.12	0.056	0.014	8.710^{-4}
∞	0.12	0.055	0.012	$3.5\ 10^{-5}$
Erlang	0.24	0.16	0.11	0.072

TABLE 2: Blocking probability for different $\mathcal{N}$ and C, $\rho = 0.9$. The values for $\mathcal{N}$ finite is obtained via simulations.

3.2 Discrepancy

As the MC model provides only limiting results, we measure the discrepancy between a finite system of nodes and the ideal system with infinitely many nodes. To this end, we derive the blocking probability for the case of one group. Table 2 reports the value of the blocking probability obtained from the model and from simulations for a single group of server with load $\rho = \frac{\lambda}{C} = 0.9$ and different values of C. We can see the model provides valid results even for $\mathcal{N}$ small, but only up to a given value of C; after that value, it provides a lower bound on blocking probability. We guess that it due to the independence among nodes that does not hold for C high. In addition, the table reports the blocking probability when a node works in isolation, which is given by the Erlang B Formula

$$PB(\lambda, C) = \frac{\lambda^C}{C! \sum_{k=0}^{C} \frac{\lambda^k}{k!}}$$

4 Tuning the Cooperation Probability

In this section we discuss implementing a tit-for-tat mechanism as an incentive approach that aims at ensuring fair cooperation among the different fog nodes or providers [31]. The idea behind the proposed algorithm is that a given node s sets its local cooperation probability such that s forwards to other providers as many requests as it receives. In other words, if s receives more requests than what it can sent, s reduces its cooperation probability. On the other hand, s increases its cooperation probability if it forwards more requests than what it receives.

When the average number of tasks sent is the same of the received one, we say that a local equilibrium point is reached. Our goal is to satisfy such condition for all nodes, $\forall s \in \mathcal{N}$, which we will refer to as the global equilibrium point. We, now, formally assert these conditions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

Let λ_{gE} be the mean of computation requests sent by a given node $s \in g$ and accepted by other nodes, λ_{Eg} the rate of received and accepted requests by node s forwarded by other nodes. The local equilibrium condition corresponds to:

$$\lambda_{gE} - \lambda_{Eg} = 0$$

In addition, the global *equilibrium* is reached if:

$$\sum_{g=1}^{G} |\lambda_{gE} - \lambda_{Eg}| = 0$$

Clearly, before designing any practical algorithm, it is important to assess first if such equilibrium points exist. Therefore, we calculate such rates and provide numerical evidence that these equilibrium points indeed exist. In addition, we show that it there exits a single equilibrium point that minimizes the blocking probabilities at each fog node.

The out-coming traffic is a Poisson process with mean:

$$\lambda_{gE} = \frac{\lambda_g}{G}\Big(\sum_{i=1}^{C}\sum_{g'=1}^{G} \pi_{gi} P'_{g'i} p_{g'} + \sum_{i=1}^{C}\sum_{g'=1}^{G} \pi_{gi}\pi_{g'i}\frac{p_{g'}}{2}\Big)$$

The probability of a given request reaching node s and forwarded to another node s' is equal to the probability of: (1) $S_s^t = i$, $S_{s'}^t < i$, and s' accepts the request, or (2) $S_s^t = S_{s'}^t = i$, ties are broken in favor of s', and s accepts the request.

Similarly, the incoming traffic is a Poisson flow with mean:

$$\lambda_{Eg} = \frac{1}{G}\Big(\sum_{i=1}^{C}\sum_{g'=1}^{G} \lambda_{g'} \pi_{gi} P_{g'i+1} p_g + \sum_{i=1}^{C}\sum_{g'=1}^{G} \lambda_{g'}\pi_{gi}\pi_{g'i}\frac{p_g}{2}\Big)$$

In fact, a request arrives to s from another node s' if: (1') the state of s' is $S_{s'}^t > i$, $S_s^t = i$, and s accepts the request, or (2') $S_{s'}^t = i$ and s' accepts to collaborate.

In the rest of the paper, we call $\rho_{gE} = \frac{\lambda_{gE}}{C}$ the out-coming load and $\rho_{Eg} = \frac{\lambda_{Eg}}{C}$ the incoming loads and refer to them as *cross-loads*. In addition, we define $\Delta\rho_g$ the net incoming cross-load for group g as:

$$\Delta\rho_g = \rho_{Eg} - \rho_{gE} \tag{3}$$

4.1 Equilibrium and Optimal Points

In our preliminarily analysis we use the MC model to elaborate on the equilibrium points. We start by considering $G = 2$. The two groups g_1 and g_2 are of equal size, with loads $\rho_1 = 0.9$ and $\rho_2 = 0.8$, respectively. We consider a fixed number of servers per node and vary the cooperation probability p_2 when $p_1 = 1$ or $p_1 = 0.9$. Fig. 4 shows the incoming and out-coming loads for the tagged node in the two groups.

Two pairs of cooperation probabilities provide equilibrium (occurring at the vertical line at the figures), roughly at $p_1 = 1, p_2 = 0.4$ and $p_1 = 0.9, p_2 = 0.35$. The corresponding blocking probability is therefore $PB_1 = 0.030, PB_2 = 0.010$ and $PB_1 = 0.039, PB_2 = 0.013$ for the top and bottom figures respectively. Note that we have performed other empirical experiments using different, however we decide to omit the results as they are very similar to the one shared in Fig. 4.

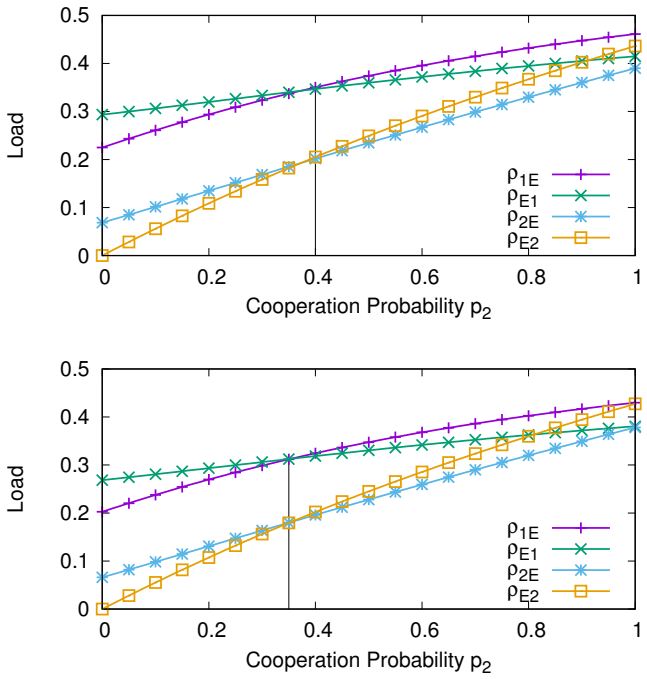


Fig. 4: Example of cross loads measurement while varying the cooperation probability p_2 for two groups. We set $p_1 = 1$ (top), and $p_1 = 0.9$ (bottom).

These numerical experiments reveal few interesting properties. The first property is that, at the equilibrium point, nodes in g_1 have higher cooperation probability compared to nodes in g_2. This derives from the observation that the nodes in g_1 see their servers busy more often, and this will *produce* a higher out-coming load compared to nodes in g_2. As a consequence, in order to reach the equilibrium point, nodes in g_1 accept requests more often than nodes in g_2, i.e., $p_1 > p_2$.

The second property is related to the impact of the cooperation probabilities among each others at the equilibrium. If nodes in g_1 decrease their cooperation probability, then the requests coming from g_2 are accepted less often. Therefore, to preserve the equilibrium, the cooperation probability of nodes in g_2 must be reduced. In other words, for $G = 2$ and $\rho_1 > \rho_2$, any pair of cooperation probabilities (p_1, p_2) that ensures the equilibrium is such that $p_1 > p_2$; in addition, p_2 is directly proportional to p_1.

The last observation is that the highest possible pair of cooperation probabilities that ensure the equilibrium implies the maximum possible amount of fair resource sharing among nodes, with the net effect of minimizing both blocking probabilities (resource sharing is a well known principle to increase resource utilization in stochastic systems). We can then conclude that for $G = 2, \rho_1 > \rho_2$ it exits an equilibrium point $(1, p_2)$ providing the lowest blocking probability while assuring fair cooperation.

A similar behavior was observed for $G = 3$. Fig. 5 shows $\Delta\rho = \sum_i |\Delta\rho_i|$ when $\rho_1 = 0.95, \rho_2 = 0.9, \rho_3 = 0.85$. We compute the cross load as a function of p_2, p_3 while setting $p_1 = 1$. The cooperation probabilities vary in step of 10^{-2}. The lowest value registered was $\Delta\rho = 1.2 \times 10^{-3}$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

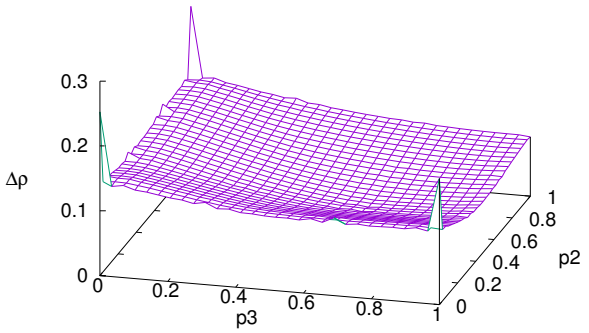


Fig. 5: Measuring $\Delta\rho$ while varying the acceptance probabilities p_2, p_3 and setting $p_1 = 1$.

corresponding to $p_2 = 0.67, p_3 = 0.43$. If one zooms in square $[0.660, 0.680] \times [0.420, 0.440]$, at step 10^{-3}, the pair $p_2 = 0.666, p_3 = 0.426$ provides $\Delta\rho = 1.1 \times 10^{-4}$. We repeat the same magnification and we find $\Delta\rho = 8.66 \times 10^{-6}$ for $p_2 = 0.6662, p_3 = 0.4256$. As a conclusion, we argue that iterating such procedure, eventually will lead to $\Delta\rho = 0$. We have repeated the same experiment setting $p_1 = 0.9$ and found that, similar to the previous case, the blocking probability increased, whereas p_2, p_3 decreased.

Based on these numerical evidence, we argue that the aforementioned properties are valid for any G, which leads us to, without a formal proof, the following property:

Claim: For a system consisting of: (i) infinite servers divided into G groups of equal sizes and a vector of loads $\rho = (\rho_1, \ldots, \rho_G)$, there exists infinitely many equilibrium points, i.e., vector of cooperation probabilities $\mathbf{p} = (p_1, \ldots, p_G)$ that provides $\Delta\rho = 0$, where $p_i > p_{i+1}$. However, there exists a "unique" probability vector $\mathbf{p}^* = (1, p_2 \ldots, p_G)$ that minimizes all the blocking probabilities PB_g, which we call *the optimal cooperation probability vector*.

To find the optimal cooperation vector for any G, we cast the problem as an optimization problem, where the objective function is to minimize $\Delta\rho(p) = \sum_i |\Delta\rho_i(p)|$, where each $\Delta\rho_i(p)$ is computed by Eq. 3. We adopt a heuristic approach as shown in Alg. 1. The algorithm is inspired by the optimization technique known as downhill simplex method developed by Nelder and Mead [32].

This method iteratively approximates a local optimum of a problem with n variables when the objective function varies smoothly, by applying at each iteration several variation rules to the set of current possible solutions. These rules include the possibility to 'go back' (reflection) and to reduce (shrink) the step according to solution's variation. Although the original algorithm could be applied, we use a much simpler version, where only one solution is carried out and the two above variations are applied.

Our algorithm (shown in Alg. 1) starts from $\mathbf{p} = (1, 1, \ldots, 1)$. As according to our claim, the optimal solution has $p_1 = 1, p_i > p_{i+1}$ with each p_i be the maximum value that ensures an equilibrium, the algorithm keeps $p_1 = 1$ and tries to sequentially decrease each p_i from $p_i = 1$ to

1. where ρ_i is the load of any server in a given group g and $\rho_i > \rho_{i+1}$

Algorithm 1 Finding the optimum vector p

Input: $\rho = (\rho_1, \ldots \rho_G), G > 0, \epsilon > 0, \delta p \in (0, 1)$
Output: $p = (p_1, \ldots p_G)$

1: **for** i=1..G **do** ▷ initialization
2: $\delta p_i \leftarrow \delta p$
3: $p_i \leftarrow 1$
4: $dir_i \leftarrow 0$ ▷ direction (0=init, -1=backward, 1=forward)
5: $\Delta\rho_i \leftarrow \Delta\rho_i(\rho, p)$ ▷ See Eq. 3
6: **end for**
7: $\Delta\rho \leftarrow \infty$
8: **while** $\Delta\rho > \epsilon$ **do**
9: **for** i=2..G **do**
10: **if** $\Delta\rho_i < 0$ **then**
11: $p_i \leftarrow max(0, p_i - \delta p_i)$
12: **if** $dir_i == 0$ **then**
13: $dir_i \leftarrow -1$
14: **end if**
15: **if** $dir_i == 1$ **then** ▷ change direction
16: $dir_i \leftarrow -1$
17: $\delta p_i \leftarrow \delta p_i / 2$
18: **end if**
19: **end if**
20: **if** $\Delta\rho_i > 0$ **then**
21: $p_i \leftarrow min(1, p_i + \delta p_i)$
22: **if** $dir_i == 0$ **then**
23: $dir_i \leftarrow +1$
24: **end if**
25: **if** $dir_i == -1$ **then** ▷ change direction
26: $dir_i \leftarrow 1$
27: $\delta p_i \leftarrow \delta p_i / 2$
28: **end if**
29: **end if**
30: **end for**
31: $\Delta\rho \leftarrow 0$
32: **for** i=1..G **do**
33: $\Delta\rho_i \leftarrow \Delta\rho_i(\rho, p)$ ▷ See Eq. 3
34: $\Delta\rho \leftarrow \Delta\rho + |\Delta\rho_i|$
35: **end for**
36: **end while**
37: **return** p

the first (highest) p_i^* that ensure the equilibrium. Hence, the equilibrium condition found also corresponds to the optimal cooperation probability vector.

More specifically, the algorithm iteratively computes the cross-loads $\Delta\rho_i$ for each group under a collaboration vector $\mathbf{p}$, and modifies the cooperation probabilities until $\Delta\rho \approx 0$ (line 8). We will empirically and quantitatively test the convergence of the algorithm in Fig. 6.

At each iteration, the cooperation probability of each node of group i can either be increasing (dir=1) or decreasing (dir=-1) of a variable amount called the step size δp_i. Initially the direction is set to $dir = 0$. If $\Delta\rho_i < 0$, the incoming load is higher than the out-coming load; therefore, the node reduces the cooperation probability p_i to $p_i - \delta p_i$ (line 11). In addition, if in the previous step, p_i was increased (line 15), the sign of the variation is inverted (from $dir = 1$ to $dir = -1$) and the step is halved (line 17). In this way, the cooperation probability goes 'back' of half δp_i, i.e., $p_i = p_i - \frac{\delta p_i}{2}$, and from now on it will change at such halved step. If the cooperation probability was already decreasing, i.e., $dir = -1$, no changes are required. A similar behavior occurs if $\Delta\rho_i > 0$.

Fig. 6 shows a typical behavior of Alg. 1. We plot the cooperation probability as a function of the iterations performed by the Alg. 1 using the following parameters: $C = 20, \delta p = 0.1, \epsilon = 10^{-10}$ when: $\rho_1 = 0.95, , \rho_2 = 0.90, \rho_3 = 0.85, \rho_4 = 0.80$

As shown in Fig. 6, all the cooperation probabilities start from 1, and then converge to smaller values. The discontinuity in the value of the probability occur when the direction

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

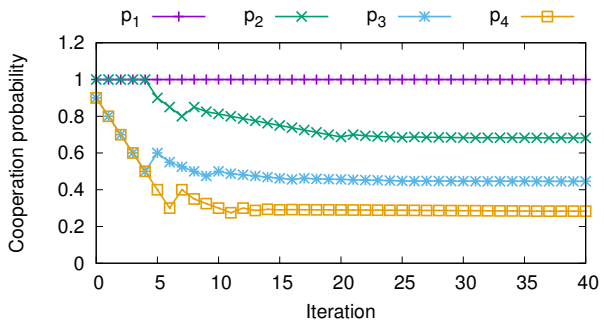


Fig. 6: Computation of the optimal cooperation probabilities vs iterations.

dir in the algorithm changes. The cooperation probability of g_1 remains stable to 1, indicating that the cooperation probabilities correspond to the optimal equilibrium point.

ρ_1	$\delta p = 0.5$	$\delta p = 0.2$	$\delta p = 0.1$	$\delta p = 0.05$	$\delta p = 0.01$
0.95	70	54	63	71	151
0.85	66	63	74	84	291
0.75	67	76	91	101	893

TABLE 3: Number of iterations for different δp and ρ

Table 3 shows the number of iterations required to converge, obtained for $G = 4, C = 20, \epsilon = 10^{-10}$ under loads $(0.75, 0.85, 0.95)$ as a function of step size δp. When the step size is high ($\delta p = 0.5$), the cooperation probabilities at each iteration vary of a big amount, so that there is the risk for $\Delta\rho_i$ to change often its sign; the variations in the cooperation probability are then bounced back and forth more frequently. However, for very small step size ($\delta p = 0.01$), the variation is very slow which requires more iterations for algorithm convergence. A good compromise seems to be $\delta p = 0.2$.

5 A Distributed Algorithm: Design and Analysis

Despite verifying the existence of an optimal cooperation between infinitely many servers, subject to different loads, we investigate, in this section, the existence of such optimal cooperation for a finite number of servers. Then, we grasp the results and the ideas from the analysis to design an distributed algorithm for fog cooperative computing.

The main challenge for using a finite number of nodes is to dynamically tune the nodes' cooperation probabilities. As fog computing scenarios are distributed by nature, a centralized solution is undesired. Indeed, a central node can be designed to receive the loads statistics from all nodes and run an algorithms, e.g., the greedy algorithm presented in the previous section, to compute the optimal cooperation probabilities. This solution is not suitable in fog computing for several reasons. First, a centralized architecture always represents a single point-of-failure. Second, considering one distant central node that serves thousands/millions of nodes adds significant delay [22]. Another solution will be electing a cluster-head like node from existing collaborating nodes arises questions such as trust, overhead, and delay. Finally, periodic gathering of queue states at a central node (proactive approach) often relies on stale information due to communication delays and data gathering granularity, while in distributed architecture nodes will always get the most up-to-date state of their queue when needed (reactive approach).

Algorithm 2 LOCAL running at node s

```
1: c, c' ← 0
2: p ← 1
3: On receiving compReq:
4: if compReq.sender == OtherNode() then
5:     Serve(compReq)
6:     return
7: end if
8: s' ← Rnd(N)
9: probeMeggage.busy ← Busy()
10: probeMeggage.sender ← s
11: ok = send(probeMessage, s')
12: if ok then
13:     send(compReq, s')
14:     c−
15: else
16:     Serve(compReq)
17: end if
18: On receiving probeMessage:
19: rb=probeMessage.busy
20: lb=Busy()
21: ok = (rb > lb & Rnd() < p)||(rb == lb & Rnd() < 0.5p)
22: if ok then
23:     send(ok, probeMessage.sender)
24:     c++
25: else
26:     send(nok, probeMessage.sender)
27: end if
28: On receiving timeout
29: setNextTimeout(ΔT)
30: c ← αc + (1 − α)c'
31: if c > 0 then
32:     p ← max(0, p − δp)
33: else
34:     p ← min(1, p + δp)
35: end if
36: c' ← c
37: c ← 0
```

We propose, LOCAL, a distributed and LOcal Cooperation ALgorithm that leverages only node's assessments of cross loading to dynamically tune the node's cooperation probability, see Alg. 2.

In LOCAL, a given node s receives a new computation requests sent by another node s' only if s was previously probed by s' and it accepted to cooperate. Therefore s accepts received requests as per line 5 in Alg. 2.

Otherwise, the request comes from an provider's user and s probes another node at random, s', by sending its current number of busy servers and waits until it receives the reply (lines 8-11). In case of cooperation, s forwards the computation request to s' and decreases its local counter c. If s' does not cooperate, the request is served locally.

When s receives a probe message (line 18) it extracts rb (number of *r*emote *b*usy servers) from the message and checks lb its current number of *l*ocal *b*usy servers. It applies the following cooperation rule: If $lb < rb$ then it cooperates with probability p. If $lb = rb$ it cooperates with probability $0.5p$. In case of cooperation a positive message is sent back and the local counter is increased (lines 23-24). In the other case it doesn't cooperate and a negative message is sent (line 26).

The cooperation probabilities are updated every ΔT units of time (lines 28-37).

When a given time slot ΔT expires, an exponential weighted moving average (EWMA) with parameter α is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

computed based on c and the counter of the previous time slot, stored in c' (line 30). If the EWMA is greater than 0

then the number of computation requests accepted by the node was higher than the one it got accepted. As a consequence, the node decrease its cooperation probability (line 32). Otherwise the cooperation probability is increased (line 34). Finally, the current counter is stored in c' and a new time slot begins with $c = 0$ (lines 36-37).

6 Performance Results

In this section, we perform a set of simulation-based experiments to assess the efficiency and effectiveness of our proposed tuning mechanism and its computed blocking probabilities. We used a custom simulator written in Python. In all our experiments we set $\mu = 1$ unit of time. We also fixed $\alpha = 0.9$ which we choose empirically after testing different values ($[0.5, 0.9]$ interval). $\alpha = 0.9$ provides the lowest variance of the cooperation probabilities.

6.1 Effectiveness of Probability Tuning

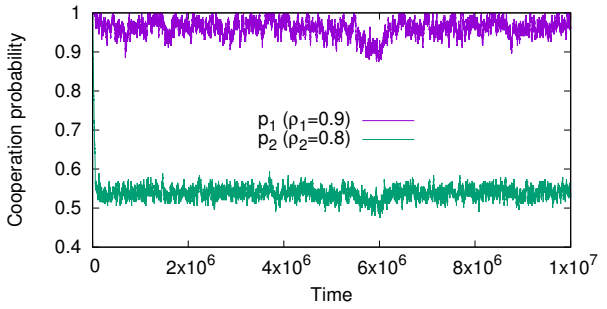
As a key element in LOCAL is the estimation of the cooperation probability, our first goal is to assess the suitability of such mechanism both under constant loads and under variable loads. We performed the following three experiments:

Exp1.1: The goal of this experiment was to verify that LOCAL reaches and maintains an equilibrium condition under fixed loads. We choose the following parameters of this experiment: $G = 2, \rho_1 = 0.9, \rho_2 = 0.8, N_1 = N_2 = 10, C = 10, \Delta T = 500, \delta p = 0.005$. [2]

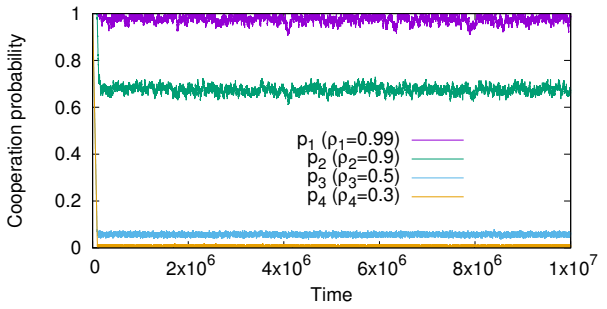
Fig. 7a plots the cooperation probabilities p_1 and p_2 of two probed nodes belonging to the two groups. Initially $p_1 = p_2 = 1$. We can see that p_2, with load ρ_2, decreases and converges around an average stable value, whereas p_1 remains close to $p_1 = 1$. This is in accordance with our analysis given in Sect. 4.1, i.e., $p_2 < p_1$ because $\rho_2 < \rho_1$.

Exp1.2: We now consider $G = 4$ groups of nodes composed of 10 server each subject to very different loads; specifically $\rho_1 = 0.99,\ \rho_2 = 0.95,\ \rho_3 = 0.50,\ and\ \rho_4 = 0.3$, respectively. As before, we set $C = 10$ and we consider one probe node in each of the four groups. The results, reported in Fig. 7b, show that despite the large difference between loads, the cooperation probabilities converge to a set of values with $p_i > p_{i+1}$, as predicted by the performance model. In addition, the first cooperation probability is close to 1. As a matter of comparison, the heuristic algorithm used to compute the optimal probability vector under the same load, for $\epsilon = 10^{-10}$ provided the following values after only 65 iterations: $p_1 = 1,\ p_2 = 0.68,\ p_3 = 0.054$, and $p_4 = 0.006$, which are very close to the probabilities found for finite system size, that were equal to $p_1 = 0.99,\ p_2 = 0.67,\ p_3 = 0.055$, and $p_4 = 0.005$ on average.

Exp1.3: In this last experiment, we assess the adaptiveness of LOCAL to a change in the load. We consider the same system detailed in Exp1.2, i.e., $G = 4$ groups, 10 nodes per group, $C = 10$. This time, we fixed the loads in the time interval between 0 to 5×10^6 to $\rho_1 = 0.99, \rho_2 = 0.90, \rho_3 = 0.80, \rho_4 = 0.6$; then we change $\rho_1 = 0.70$ afterwards, i.e., when $t > 5 \times 10^6$. Fig. 8 plots the cooperation probabilities at 4 probed nodes over time as we vary new load profile at timescale 5×10^6. When ρ_1 falls to 0.7, the highest load becomes $\rho_2 = 0.90$; and, in fact, the corresponding acceptance probability increases to almost 1. The reduction of the first group's load incurs fewer forwarded requests to the third and to the last group of nodes. As a consequence their accepting probabilities are increased and stabilize around a new higher value. We show that, as ρ_2 registers the highest value, p_2 rises to 1, whereas p_1 falls down to approximatively 0.2, crossing p_2 and p_3. The time required to reach the new set of probabilities is $\approx 8 \times 10^4$.



(a) Exp 1.1



(b) Exp 1.2

Fig. 7: Cooperation probability over time using (a) $G = 2$ groups of nodes, and (b) $G = 4$ group of nodes.

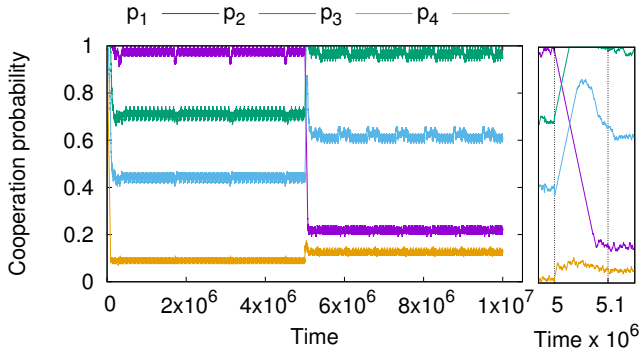


Fig. 8: Cooperation probability over time for Exp1.3; Zoomed-in version displayed in right.

2. As opposed to variable δp used in Alg. 1, LOCAL uses constant steps, which we choose to set to small values in order have a low variance of the blocking probabilities. A possible algorithm based on variable δp is proposed later in this section.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

6.2 Algorithm Responsiveness

We define the response time of LOCAL, R, as the minimum amount of time required to reach the new set of cooperation probabilities that provides a new equilibrium. From Fig. 8, we find $R =\approx 8 \times 10^4$ time units. In order to see how the absolute values of the loads affect R we repeat a similar experiment with initial loads scaled of 0.1, i.e., $\rho_1 = 0.89, \rho_2 = 0.85, \rho_3 = 0.80, \rho_4 = 0.65$ and ρ_1 changing from 0.89 to 0.7. The stable cooperation probability before and after the change was $p_1 \approx 1, p_2 \approx 0.8, p_3 \approx 0.6, p_4 \approx 0.2$ and $p_1 \approx 0.38, p_2 \approx 1, p_3 \approx 0.73, p_4 \approx 0.28$ respectively. In this case, we registered a value $R \approx 6.2 \times 10^4$ time units. After that, we have empirically tested multiple load variations and found that R varies by orders of magnitude smaller or larger due mainly to the steps δp.

In general, after a load change, the responsiveness R is related to the difference gap between the old (p) and new (p') stable cooperation probabilities over all the nodes. Recall that the cooperation probability at every node changes every ΔT time units with steps δp. *If* p_i is the stable cooperation probability of a node belonging to group i prior to a load change, p_i' the new stable probability after the load change, and δp is the probability step size, *then* the probability reaches p_i' no earlier than $\Delta T \frac{p_i - p_i'}{\delta p}$ time units.

Thus, a lower bound to the responsiveness of the algorithm is determined by nodes whose variation in the cooperation probabilities is maximum, i.e.,

$$R \geq \Delta T \frac{\max\limits_{i \in G}(p_i' - p_i)}{\delta p}$$

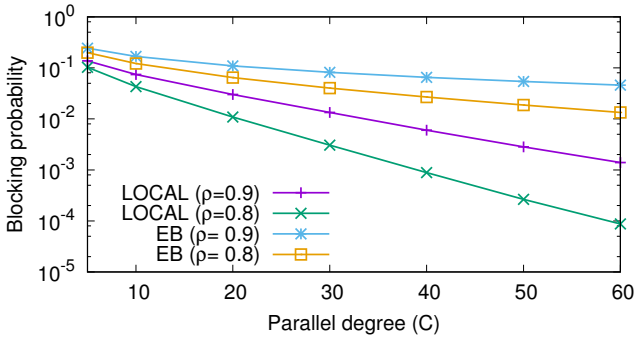
In general, to reduce R, one can (1) decrease ΔT, or (2) increase δp. However, a small value of ΔT implies a less accurate estimate in the cross load with an increase in variance, say σ, of the cooperation probability and, consequentially, of the blocking probability seen by a end user. Similarly, a high value of δp means a higher magnitude in the oscillation of the cooperation probabilities once the stable point is reached that result in a higher σ.

The value δp is then a trade-off between R and σ. To reduce both values the step δp should be variable in a way that δp is high when the load is changing and smaller when the load is constant. To mitigate such a drawback, a simple dynamic tuning adaptation protocol is described in Sec. 6.4.

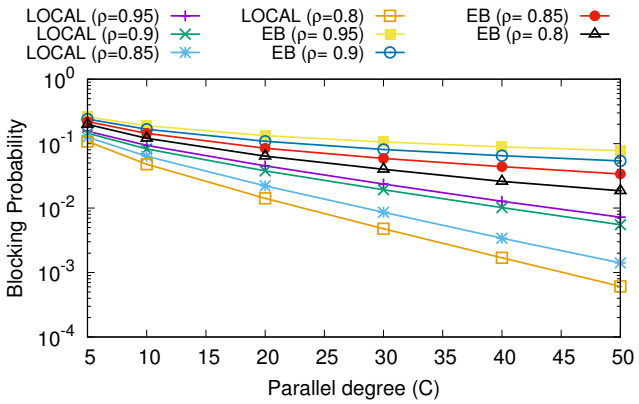
6.3 Blocking Probability Measurements

We investigate the impact of fair cooperation on the overall system performance by measuring the blocking probability as a function of C, number of servers per node as well as the loads. We fix the number of servers $C = 30$ and vary the loads. We consider two sample simulation scenarios both last for 10^5 time units and statistics are collected after a warm-up period of 3×10^4 time units that ensure stable cooperative probabilities. Each simulation is repeated 10 times with a different random seed.

Exp2.1: We experiment a system of 10 nodes loaded with $\rho_1 = 0.9$ and 10 nodes with $\rho_2 = 0.8$. Fig. 9 (a) compares the blocking probabilities of LOCAL and one obtained from the Erlang B formula (EB), corresponding to nodes working in isolation. We show that LOCAL reduces the blocking probability more than a non cooperative scheme, EB. For example, for the first group and $C = 50$ the blocking probability decreases from 5.4×10^{-2} to $\approx 2.8 \times 10^{-3}$ for the first group and from 1.8×10^{-2} to $\approx 2.6 \times 10^{-4}$ for the second one. In the case of non cooperation, a node works in isolation in order to reduce the blocking probability of the same amount C must be increased from $C = 50$ to $C = 65$ in the first case and to $C = 64$ in the second case, i.e., of about 30%.



(a) Exp 2.1



(b) Exp 2.2

Fig. 9: Blocking probabilities in function of the number of psychical servers.

Exp2.2: We now consider a system consisting of 10 nodes with $\rho_1 = 0.95$, 10 nodes with $\rho_2 = 0.9$, 10 nodes with $\rho_3 = 0.85$ and 10 nodes loaded with$\rho_4 = 0.8$. The plot in Fig. 9 (b) shows similar results compared to Exp2.1. For example, for $C = 50$, the blocking probability in the second group decreases from 5.4×10^{-2} to $\approx 5.5 \times 10^{-3}$ and from 1.8×10^{-2} to $\approx 6.1 \times^{-4}$ for the last group .

Exp2.3: In this experiment, we consider two groups of 10 nodes each with $C = 30$ servers each. The first group's load is constant and set to $\rho_1 = 0.9$, whereas ρ_2 varies from 0.4 to 0.9. The goal of this experiment is to investigate how the difference between loads of cooperating nodes affect the blocking probabilities.

Fig. 10 plots the blocking probability for LOCAL and the one obtained from the Erlang-B formula. We show that LOCAL outperforms Erlang-B, for all considered loads, by 10 to 50 $\times$. This gain ratio while slightly reduced when $\rho_2 \rightarrow 1$, the absolute performance difference increases; note that the y-axis is in log-scale. We also note that when $\rho_2 = \rho_1 = 0.9$, the two groups will have similar load,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2828809, IEEE Transactions on Cloud Computing

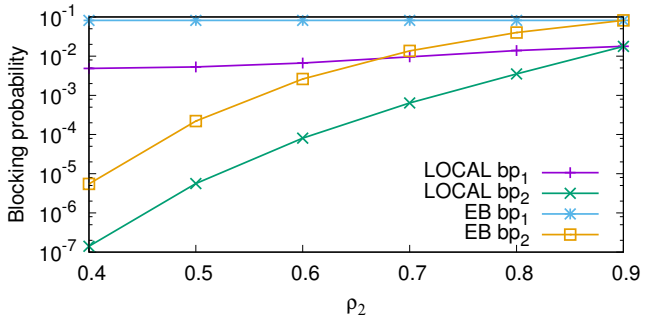


Fig. 10: Blocking probabilities as a function of the load (ρ_2); $\rho_1 = 0.9$.

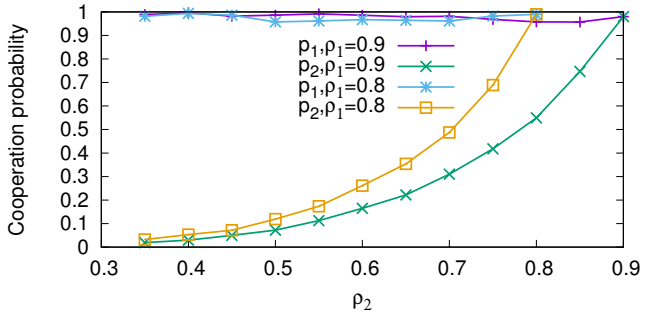


Fig. 11: Cooperation probabilities as a function of the load (ρ_2); $\rho_1 = \{0.8, 0.9\}$.

therefore LOCAL assigns the same blocking probabilities of 0.01 which is much smaller than EB's blocking probability of roughly 0.1.

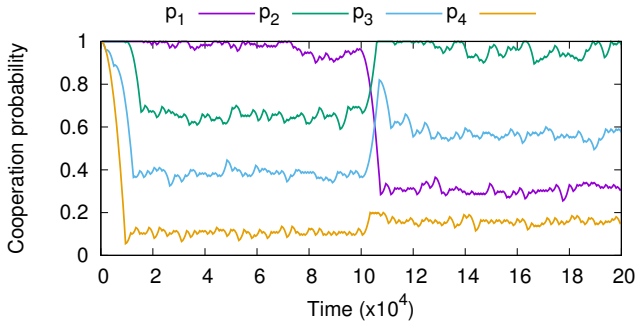
Exp2.4: In this last experiment, we consider similar setup than Exp2.3 while using $\rho_1 = 0.9$ and 0.8. We measure, in Fig. 11, the cooperation probabilities while varying ρ_2 from 0.4 to 0.9. The cooperation probability p_2 increases with ρ_2 because when ρ_2 increases nodes in g_2 sees a larger number of their request to be accepted (if fact, nodes in g_1 always accept requests). And thus, to maintain the equilibrium their cooperation probabilities increase. Also, the absolute value of p_2 is lower for $\rho_1 = 0.9$ because the requests towards nodes in g_2 increases with ρ_1 and thus, the keep the equilibrium, nodes in g_2 have to increase their cooperation probability w.r.t to $\rho_1 = 0.8$.

6.4 DyLOCAL: LOCAL Enhancement With Dynamic Tuning

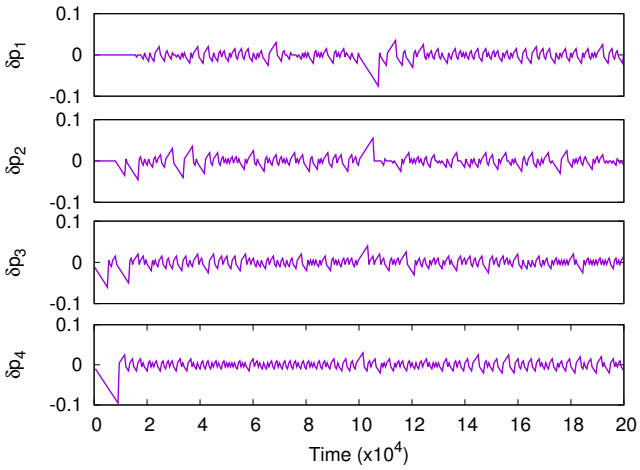
We discuss, in this section, possible enhancement of LOCAL by leveraging dynamic tuning of δp in order to reduce the overall response time R. We refer to this protocol enhancement as DyLOCAL.

The key idea of DyLOCAL is motivated by the observation that when the load changes, the current cooperation probability, i.e., p varies with several *consecutive* increases (or decreases). Thus we propose that, when the previous condition is detected, we increase the step size linearly over time until the new cooperation probability value is reached.

More precisely, let us assume for the sake of simplicity, that under the new load a cooperation probability increases from p to p'. DyLOCAL operates as follows: the 1^{st} probability increase is equal to δp, the 2^{sd} consecutive (with



(a) Variation of the cooperation probability over time.



(b) Variation of the steps δp over time.

Fig. 12: Results of Exp3, where the load changes at time $t = 10^5$.

no decrease registered between two consecutive increases) is equal to $2\delta p$, and for the i^{st} consecutive increase, the probability change is of $i \cdot \delta p$. When p' is reached (i,e, the condition of consecutive increases may change), the step size is reset to δp and DyLOCAL re-operates again as above. Note that DyLOCAL is a very simple algorithm, optimizing it is beyond the goal of this paper and will be investigated in our future work.

To grasp the potential improvement of DyLOCAL over LOCAL, assume that $p = 1$ and $p' \approx 0$. Using LOCAL, $R \approx \Delta T \frac{1}{\delta p'}$, whereas with DyLOCAL the responsiveness $R \approx \Delta T \sqrt{\frac{2}{\delta p}}$. This last value is obtained by solving:

$$\delta p \sum_{i=1}^{k} i = \delta p \frac{k(k+1)}{2} \approx \delta p \frac{k^2}{2} = 1$$

For instance, when $\delta p = 0.005$, the responsiveness of LOCAL is $R = 200\Delta T$, while DyLOCAL's is $R = 20\Delta T$, i.e., one order of magnitude shorter response time.

Exp3: We experiment DyLOCAL using the same scenario in Exp.1.3 with the only difference in the load, which changes at time $t = 10^5$. Fig. 12a shows the cooperation probabilities p_i over time for a representative node of each of the $G = 4$ groups ($i = 1..4$). DyLOCAL registers a shorter responsiveness of $R \approx 8 \times 10^3$ time units, which is indeed one order of magnitude faster than the one observed for LOCALwhose response time was $R \approx 8 \times 10^4$.

Fig. 12b plots the step size δp_i used to tune p_i over time. The steps increased linearly when the load changes (either at the beginning of the simulation or at $t = 10^5$). Once a stable probability is reached, the step variations becomes smaller as in LOCAL. The figure shows for example, that when the simulation starts, at $t = 0$, only p_3 and p_4 decreases, with steps that increase linearly (i.e., p_2, p_3 change quadratically). A clear variation in the step sizes is also visible at $t = 10^4$, i.e., when ρ_1 decreases from 0.99 to 0.7. Here p_1 decreases quadratically, while all the other probabilities increase quadratically.

7 Conclusion and Future Work

In this paper, we have addressed the problem of cooperation among different providers in the context of fog computing. We have addressed cooperation fairness so that a win-win condition can be met. The proposed distributed protocol is based on the power of two model which sows efficient fair collaboration that improves the overall system utilization. Indeed, each provider can improve its service level, in a fair way and at no cost.

We have first analyzed the problem with a simple model and then designed a distributed algorithm, LOCAL, whose performance has been investigated through simulations. Simulation results have shown that LOCAL registers a blocking probability outperforming other state-of-the-art solutions by two order of magnitudes or more.

This is a first step towards proposing a distributed platform for full cooperative fog computing. In the future, we plan to study an optimized (or near optimal) cooperation probability tuning algorithms.

References

[1] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[2] D. Uckelmann, M. Harrison, and F. Michahelles, "An architectural approach towards the future internet of things," in *Architecting the internet of things*. Springer, 2011, pp. 1–24.

[3] A. Mtibaa, A. Fahim, K. A. Harras, and M. H. Ammar, "Towards resource sharing in mobile device clouds: Power balancing across mobile devices," in *Proceedings of the Second ACM SIGCOMM Workshop MCC '13*, ser. MCC '13, 2013, pp. 51–56.

[4] A. Mtibaa, K. A. Harras, K. Habak, M. Ammar, and E. W. Zegura, "Towards mobile opportunistic computing," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 1111–1114.

[5] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of internet of things," *IEEE Transactions on Cloud Computing*, 2015.

[6] M. Firdhous, O. Ghazali, and S. Hassan, "Fog computing: Will it be the future of cloud computing?" in *Third International Conference on Informatics & Applications, Kuala Terengganu, Malaysia*, 2014, pp. 8–15.

[7] D. Kliazovich, S. T. Arzo, F. Granelli, P. Bouvry, and S. U. Khan, "e-stab: Energy-efficient scheduling for cloud computing applications with traffic load balancing," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 7–13.

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

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

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Tech. Rep., 2009.

[11] Y. Zhang, K. Guo, J. Ren, Y. Zhou, J. Wang, and J. Chen, "Transparent computing: A promising network computing paradigm," *Computing in Science and Engg.*, vol. 19, no. 1, pp. 7–20, Jan. 2017. [Online]. Available: https://doi.org/10.1109/MCSE.2017.17

[12] P. N. F. Bonomi, R. Milito and J. Zhu, "Fog computing: A platform for internet of things and analytics," *Big Data and Internet of Things: A Roadmap for Smart Environments Springer International Publishing*, pp. 169–186, 2014.

[13] Y. J. Q. Yaseen, F. Albalas and M. AlAyyoub, "Leveraging fog computing and software defined systems for selective forwarding attacks detection in mobile wireless sensor networks," *Journal of Transactions on Emerging Telecommunications Technologies, Wiley Online Library*, vol. special issue, March 2017.

[14] D. N. B. Varghese, N. Wang and R. Buyya, "Feasibility of fog computing," *arXiv preprint arXiv:1701.05451*, 2017.

[15] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *SIGCOMM Comput. Commun. Rev. 44, 5 (October 2014)*, pp. 27–32, 2014.

[16] C. L. Shanhe Yi and Q. Li, "A survey of fog computing: Concepts applications and issues," *In Proceedings of the 2015 Workshop on Mobile Big Data (Mobidata '15) ACM*, pp. 37–42, 2015.

[17] J. Flinn, "Cyber foraging: Bridging mobile and cloud computing," *Synthesis Lectures on Mobile and Pervasive Computing*, vol. 7, no. 2, pp. 1–103, 2012.

[18] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, 2002.

[19] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *ACM MobiSys*, 2007.

[20] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966473

[21] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *MobiSys'10*, 2010, pp. 49–62.

[22] 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.

[23] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, ser. NCM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–51. [Online]. Available: http://dx.doi.org/10.1109/NCM.2009.218

[24] F. R. Dogar and P. Steenkiste, "Segment based internetworking to accommodate diversity at the edge," Computer Science Department, Carnegie Mellon University, Tech. Rep. CMU-CS-10-104, 2010.

[25] 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.

[26] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[27] M. Bramson, Y. Lu, and B. Prabhakar, "Asymptotic independence of queues under randomized load balancing," *Queueing Syst. Theory Appl.*, vol. 71, no. 3, pp. 247–292, Jul. 2012.

[28] 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.

[29] A. S. Godtschalk and F. Ciucu, "Randomized load balancing in finite regimes," in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. IEEE, 2016, pp. 580–589.

[30] C. Fricker, F. Guillemin, P. Robert, and G. Thompson, "Analysis of an offloading scheme for data centers in the framework of fog computing," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 1, no. 4, p. 16, 2016.

[31] M. A. Nowak and K. Sigmund, "Tit for tat in heterogenous populations," *Nature*, vol. 355, no. 6357, p. 250, 1992.
[32] R. M. John A. Nelder, "A simplex method for function minimization," *in Computer Journal*, vol. 7, pp. 308–313, 1965.

Roberto Beraldi received his PhD in computer science from University of Calabria, Italy in 1996. Until 2002 he worked at Italian National Institute of Statistics. Since then he has been with the Department of Computer, Control, and Management Engineering at Sapienza University of Rome, Italy. He has published more than 80 papers in journals and main conferences related to distributed systems and wireless networks. His current research interest includes mobile and cloud computing, wireless networks, and distributed systems.

Abderrahmen Mtibaa is currently a visiting assistant professor at the Computer Science Department in New Mexico State University (NMSU). He received the Ph.D degree in computer science from the University Pierre & Marie Currie (Paris VI) in 2010. In 2012, he was a postdoctoral research associate at Carnegie Mellon University, where he worked on IoT and networking system research. From 2012 to 2016 he was a research scientist at Texas A&M university in Qatar. He is an author in more than 70 journal and conference papers in various areas of computer communication and social networking. His current research interests include Information-Centric Networking, Networked Systems, Social Computing, Personal Data, Privacy, IoT, mobile computing, pervasive systems, mobile security, mobile opportunistic networks/DTN, wireless and Ad-hoc networks, mobility models, protocol design, routing/forwarding, network communities and social networking.

Hussein Alnuweiri (S'81–M'83–SM'17) received the Ph.D. degree in electrical and computer engineering from the University of Southern California, Los Angeles, in 1989. Currently, he is a Professor in the Department of Electrical and Computer Engineering at Texas A&M University, Doha, Qatar. From 1991 to 2007, he was a Professor with the Department of Electrical and Computer Engineering, University of British Columbia. From 1996 to 1998, he represented the University of British Columbia, Vancouver, BC, Canada, at the ATM Forum. From 2000 to 2006, he served as a Canadian delegate to the ISO/IEC JTC1/SC29 Standards Committee (MPEG-4 Multimedia Delivery), where he worked within the MPEG-4 standardization JTC1-SC29WG11 group to develop the first client-server MPEG4 video streaming reference software. Dr. Alnuweiri has a long record of industrial collaborations with several major companies worldwide. He is also an inventor, and holds four U.S. patents. He has authored or co-authored over 200 refereed journal and conference papers in various areas of computer and communications research. In particular, his research interests include mobile Internet technologies, cyber security and cyber systems, mobile cloud computing, wireless communications, routing and information dissemination algorithms in mobile networking, and quality-of-service provisioning and resource allocation in wireless networks.