Performance Evaluation of Some Adaptive Task Allocation Algorithms for Fog Networks

Ioanna Stypsanelli, Olivier Brun, Balakrishna Prabhu

Performance Evaluation of Some Adaptive Task Allocation Algorithms for Fog Networks

Ioanna Stypsanelli, Olivier Brun, Balakrishna J. Prabhu
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

October 20, 2020

Abstract

The Fog Computing paradigm provides a seamless bridge between the Cloud and the Edge computing architectures. Depending on the their QoS requirements, tasks can be processed in either the Edge or the Cloud or migrated from one to the other. For the benefits of this flexible architecture to be seen, task allocation algorithms should be able to adapt to the load in the Fog and in the Cloud, and send the tasks to a lightly loaded resource.

Current task allocation algorithms in Fog computing literature use a simple offloading strategy: a task is first sent to the nearest Fog node. If the nearest Fog node is saturated, it offloads the task to the Cloud. Although simple to implement, such a strategy disregards available resources in the Fog nodes that could be further away but less congested.

Using a discrete-event simulation approach which relies on the network simulation framework OMNeT++, we show that simple adaptive algorithms based on congestion estimation outperform the standard nearest node algorithm. For this, we choose four distributed routing algorithms that were proposed for other networking applications but which are also well-suited for Fog networks. These algorithms improve the resource usage as well as reduce the mean job processing times in scenarios with offloading as well as without offloading. Our usecase is that of Fog networks that use the cellular network: base stations (access nodes) forward traffic to computing nodes (Fog and Cloud nodes) in a distributed way without coordination and sharing of state-information.

We evaluate the performance of these algorithms on several scenarios that include sudden changes in the arrival rate of requests (to model peak hours) and changing the variance of request processing times (to study the robustness to request-size distributions) to understand the advantages and drawbacks of each of them.

1 Introduction

The Internet of Things (IoT) will connect various objects, buildings or facilities to the Internet generating a tremendous amount of data and will thus result in a another dramatic increase of network usage. Examples of emerging IoT applications are *connected cars*, *tactile internet*, *smart homes*, *augmented reality*.

This breed of applications will likely contain latency critical features that have stringent delay requirements that will not be satisfied by the far away Cloud. This calls for the extension of the classical centralized Cloud computing architecture towards a distributed architecture closer to user premises at the *edge* of the network. One extension proposed over the last decade is *Edge computing*.

While the Cloud is about sharing infrastructure and finding cost saving opportunities, the Edge is about distributing infrastructure closer to user in order to increase availability and improve *Quality of Service* (QoS).

They are complementary and allow a better usage of resources. The Cloud does it by sharing computing resources while the Edge computing does it by keeping computation local, making it useless to reach the deeper Internet and saving network resources.

Fog Computing, a paradigm introduced by Cisco in 2015 [1], recognizes the benefits of both the Cloud and the Edge and tries to bridge them seamlessly in a unified model where different jobs can be processed on either the Edge or the Cloud or migrated from one to the other depending on their requirements.

Connected vehicles are expected to become one of biggest usecases of the Fog Computing architecture. Vehicles in the future will be sharing data and requests amongst themselves as well as the network. Requests requiring a quick turnaround (information of nearby road safety restrictions, e.g.) are more suited for processing in a nearby Fog node, while others with more flexible constraints (infotainement applications, e.g.) can be sent to to Cloud.

To exploit the benefits of Fog Computing paradigm, one has to design the right task allocation algorithms that routes jobs to either one of the Fog nodes or the Cloud in a way that balances the load while respecting the QoS constraints. For example, during off-peak hours when the vehicular traffic is sparse, jobs can be handled exclusively by the Fog nodes. On the other hand, when traffic is dense during peak hours, the algorithm should divert some of its traffic to the Cloud so as not to overload the Fog nodes. That is, the algorithm should be able to adapt and react to changes in traffic patterns and user demands.

Current task allocation algorithms in Fog computing literature use a simple offloading strategy: a request is first routed to the nearest Fog node. If the nearest Fog node is saturated, then the request is offloaded to the Cloud.

Although very simple to implement, such a strategy disregards available resources in Fog nodes that could be further away but less congested. Can we design adaptive algorithms that make better use of Fog resources? One approach involves adapting the allocation decision based on the estimated congestion in Fog nodes and allocate tasks to nodes that are less congested. Of course, if all the nodes are congested, the request can be sent to the cloud.

1.1 Contributions

The main contribution of this article is to show that simple adaptive algorithms based on congestion estimation indeed outperform the standard nearest node algorithm. For this, we choose four distributed routing algorithms that were proposed for other networking applications but which are also well-suited for Fog networks. These algorithms improve the resource usage as well as reduce the mean job processing times in scenarios with offloading as well as without offloading. Our usecase is that of Fog networks that use the cellular network for communicating with end-user. This is the case for connected vehicles where cars use the cellular network to exchange information. The base stations (access nodes) forward traffic to computing nodes (Fog and Cloud nodes) in a distributed way without coordination and sharing of state-information.

The four distributed task allocation algorithms considered require neither cooperation between base stations nor knowledge of the physical infrastructure. These algorithms are learning based and do not rely on a specific model of the infrastructure but just react to the response times they observe from the Fog nodes. We assume that each base station independently adapts its task allocation strategy, without coordination or even clock synchronisation with the other base stations.

We adopt a discrete-event simulation approach which relies on the network simulation framework OMNeT++. We evaluate the performance of these algorithms on several scenarios that include varying the arrival rate of requests (to model peak hours) and changing the variance of request processing times (to study the robustness to request-size distributions) to understand the advantages and drawbacks of each of them.

1.2 Organization

This article is organized as follows. Section 2 is devoted to related work. Section 4 presents the different task allocation strategies considered. Numerical results are presented in Section 5. Finally, some conclusions are drawn in Section 6.

2 Related literature

In [12], the authors aim to allocate tasks adaptively and efficiently in the Cloud and provides a platform named TAP (Task Allocation Platform) that uses a Linux kernel module and can compare scenarios in a realistic setting. They use a reinforcement learning algorithm based on random neural network proposed in [8] in order to allocate tasks to the Cloud.In our study we use an OMNeT++ simulation model instead an actual Linux implementation. In comparison with our work this happens in a centralized fashion since the TAP control gathers all information. The sensible strategy we use is taken from this paper and we compare it with many more strategies.

In [4] the authors formulate a cooperative offloading policy between two edge data centers for load balancing. They define a blocking state in which the requests are dropped and compare with two other schemes in order to minimize the amount of blocked requests: with an isolated policy, where no data center works with cooperation with the other, and a fully shared where any request is forwarded to any other datacenter. The cooperative scheme they propose behaves better than the two others. The problem the authors study is similar with ours as the offloading scenario we study dynamically happens when a Fog node is overloaded with tasks. However their scheme is static whereas the strength of our approach is that it is dynamic and can adapt to the unknown.

A theoretical work is [7] where authors consider the scenario of offloading with a certain probability blocked requests at the Fog to the neighboring data centers and to the Cloud. Through functional equations and Markov theory they estimate the gain achieved via cooperation between neighboring data centers.

In [3], the authors formulate the load balancing problem as a Markov Desicion Process (MDP) solved by Q-learning. Q-learning is a common reinforcement learning algorithm that can solve MDPs. The multi-armed bandits algorithms we use are simpler forms of reinforcement learning that deal with problems with only one state in contrast with MDPs that model problems with several states. This allows Q-learning models to keep knowledge over time and remember the past to better handle the future. Their reward function takes into account processing time to minimize and overload probability. A difference in our approach is that we take the constraint of giving base stations no knowledge of the system, while the authors place their Q-learning algorithm in the SDN controller which has an overview of all the system.

There are different studies on offloading which describe benefits other than latencies improvements, such as energy saving opportunities. This is particularly interesting in fog models where Fog nodes may be battery-powered devices. In [10] for instance authors formulate a multi-objective optimization problem to minimize the energy consumption, execution delay

and payment costs that finds the optimal probability to offload.

We can mention existing surveys on computation offloading such as [5] [9] which provide a review of the state-of-the-art of computation offloading in the various contexts it can be useful: energy consumption minimization, Quality of Services guarantees, and computation and storage requirements. There are also a number of tools and frameworks that help building offloading infrastructure or mobile application developers implement the required facilities. In [6] the authors present a framework that allows computations to be dynamically offloaded to the Cloud with a runtime optimizer that can transform local computation into remote calls on-demand and thus make it easy for application developers to built adaptive applications. They show energy consumption improvement in resource-intensive workloads.

3 Simulation Model

In our simulation model, message objects are used to represent jobs, but also events such as the end of service of a job on a given server. A job is specified by a source, a destination and a delay, which represents the processing time of the job. Two main elements in our simulation model are a base station and a Fog/Cloud node.

For simplicity, a Fog/Cloud node is modelled as a set of parallel single server queues (Fig. 1). A load-balancer assigns incoming jobs to one of the queues. The queues in our model represent a single-server queues with infinite capacity. We assume that the discipline of service is FCFS.

The parameters of a Fog/Cloud node are the number of servers it is composed of, the service time of a job on a server for instance we can define exponential service times or Pareto distributions. (for simplicity, we assume homogeneous servers) as well as the routing algorithm for assigning jobs to servers (e.g., *round-robin* or *join-the-shortest-queue* policies).

A base station is connected to all Fog nodes. It has a task allocation algorithms that can be one of the five algorithms described in the next section. There are other parameters such as inter-arrival time between job requests and traffic perturbations.

4 Adaptative Tak Allocation Algorithms

In this section, we present the task allocation algorithms considered in this paper. In all these algorithms, an access node $i = 1, \ldots, K$ maintains a probabilistic choice vector $\mathbf{p}_i$, and allocates a job to Cloud/Fog node $j = 1, \ldots, N$ with probability $p_{i,j}$. Most of the algorithms update the choice vector $\mathbf{p}_i$ whenever a new job reply is received, usually based on an estimation G_j of the mean response time of Fog node j. We first explain how the mean response time of a Fog node can be estimated from job replies in Section

Figure 1: Fog node processing in parallel queues

4.1. We then briefly describe how each algorithm computes the probabilistic choice vector $\mathbf{p}_i$ in Section 4.2.

4.1 Estimation of the mean response time of a Cloud/Fog node

In most algorithms, the task allocation agent maintains a weighted average of the response times of Fog node j. Let $G_j(k)$ be the estimated value of the response time of Fog node $j = 1, \ldots, N$ after k job replies have been received from it. This estimated value is updated each time a new job reply is received as follows

$$G_j(k) = (1-\alpha)\, G_j(k-1) + \alpha d_j(k),\ k = 1, 2, \ldots, \tag{1}$$

where $d_j(k)$ is the response time of the k^{th} job sent to node j. The parameter $0 \leq \alpha \leq 1$ is used to vary the weight given to the most recent measurement as compared to past values. In our experiment, we have used the value $\alpha = 0.1$. Initial values $G_j(0)$ are set to 0 to encourage exploration. Note that after k measurements, we have

$$G_j(k) = (1-\alpha)^k G_j(0) + \alpha \sum_{n=1}^{k} (1-\alpha)^{k-n} d_j(n),$$

so that the weight of a measure decays exponentially according to the exponent on $(1-\alpha)$.

Instead of choosing a constant step-size α, it is of course possible to vary the step-size as the number of measures grows. For stationary problems, it is known that the conditions $\sum_{k=1}^{\infty} \alpha_k = \infty$ and $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$ are sufficient to guarantee convergence to the (true) mean values. In particular, the choice $\alpha_k = 1/k$ is often used and gives $G_j(k) = \frac{\sum_{n=1}^{k} d_j(n)}{k}$, that is, the estimated value of Fog node j is just the average value of response times for jobs processed at this node. In our work, we shall however stick to a constant step-size α because it is known to be more convenient for non-stationary problems [11].

4.2 Task Allocation Probabilities

We now describe how the different algorithms compute the probabilistic choice vector $\mathbf{p}_i$ used at access node i when a new job request arrives. It is always assumed that the most recent estimates G_j of the mean response times are used. The algorithms work as follows:

- **Static Task Allocation**: in this task allocation strategy, which is also known as Bernoulli routing, the probabilistic choice vector $\mathbf{p}_i$ is initially given and it is not updated when a new job reply is received. A special case that we shall consider in the following is the case where $p_{i,j} = 1$ if $j = j^*$ and 0 otherwise, where j^* is the geographically closest micro-datacenter to the access node i.

- **ϵ-greedy Allocation**: the ϵ-greedy algorithm is a method used for solving multi-armed bandits problem (see Chapter 2 of [11]). The basic idea of this method is to behave greedily most of the time, that is to choose the fog node j with the lowest estimated response time G_j most of the times, but every once in a while (with small probability ϵ) to select instead another fog node uniformly at random. More precisely, let j^* be the Fog node with the lowest estimated response time, that is, $j^* \in \text{argmin}_n G_n$. Then, the next job is allocated to fog node j^* with probability $1-\epsilon$, and to another fog node $j \neq j^*$ with probability $\frac{\epsilon}{N-1}$, where N is the total number of Fog nodes.

- **Softmax Allocation**: with the Softmax algorithm [11], the next job is allocated to Fog node j with probability

$$p_{i,j} = \frac{e^{-G_j/\tau}}{e^{\sum_{n=1}^{N} -G_n/\tau}},$$

  where τ is a positive parameter called the temperature. High temperatures cause all choices to be (nearly) equiprobable, while low temperatures favour Fog nodes with low estimated response times (in the limit $\tau \to 0$, the Fog node with the lowest estimated response time

is selected with probability 1). In our experiments, we have used the value $\tau = 30$ ms, which is the value of the same order of magnitude as the total response time of a job.

- **Sensible Routing**: in the sensible routing algorithm, which was proposed in [12], the probability $p_{i,j}$ to allocate a job to Fog node j is computed as follows

$$p_{i,j} = \frac{1/G_j}{\sum_{n=1}^{N} 1/G_n},$$

where again G_n represents the most recent estimate of the mean response time of node $j = 1, \ldots, N$.

- **EXP3 Allocation**: the *adversarial bandit* is a version of the multi-armed bandit problem introduced by Auer and Cesa-Bianchi in 1998 in which almost nothing is assumed about the mechanism that generates the rewards. In this problem, it is simply assumed that, at each iteration, an agent chooses an arm and an adversary simultaneously chooses the cost structure for each arm. The goal is still to compete with the best action in hindsight. The EXP3 algorithm was proposed in 2001 by Auer, Cesa-Bianchi, Freund, and Schapire [2] and is known to have an expected regret bound of $\sqrt{2Tn\log(n)}$ when costs take their values in $[0, 1]$. In contrast to previous algorithms, EXP3 does not use directly the estimates G_j, $j = 1, \ldots, N$, but it instead assigns a weight $w_j(j)$ to each node j. Initially set to 1, the weights are updated as follows when the k^{th} job reply is received from node j

$$w_j(k) = w_j(k-1) \times exp\left(-\alpha \frac{d_j(k)}{N p_{i,j}}\right),$$

where $d_j(k)$ represents the response time of the k^{th} job sent to node j, as in Section 4.1. When a new job request arrives, the choice vector $\mathbf{p}_i$ is computed as follows

$$p_{i,j} = (1-\gamma)\frac{w_j}{\sum_{n=1}^{N} w_n} + \gamma\frac{1}{N}, \quad j = 1, \ldots, N,$$

where again it is assumed that the most recent values of the weights are used.

In the next section we present various experiments we performed using the above task allocation algorithms.

5 Performance Evaluation

The objective of this section is to compare those adaptive learning-based task allocation schemes.

We focus on two main scenarios. One scenario where base stations are connected to two Fog nodes and the main Cloud and can allocate their jobs without coordination. The second is similar but this time the Fog nodes are connected to the Cloud and can use a dispatching strategy to offload their tasks. We compare the two scenarios and the gains observed in response times by using offloading.

The organization of the experiments is the following. In the first scenario we perform experiments with constant link delays and with variable link delays. We as well compare the response times obtained for exponentially-distributed service times against those obtained with Pareto service time distributions. We then perform robustness tests by executing each scenario 5000 times and taking the average, minimal and maximal values on all of these scenarios. In the case of offloading, we first compare four different dispatching policies: power-of-two-choices, join the shortest queue first, random policy and round robin and then we perform similar tests.

5.1 First Scenario : Task Allocation without offloading

5.1.1 A. Exponentially-distributed Service Times - Constant Link delays

We first consider the scenario illustrated by Figure 2. In this scenario, there are two Fog nodes (corresponding to `fog[0]` and `fog[1]`), one cloud datacenter (corresponding to `fog[2]`) and two access nodes. Note that communication delays are symmetric for `client[0]` and `client[1]`. The access node `client[0]` is closer to `fog[0]`, and similarly `client[1]` is closer to `fog[1]`.

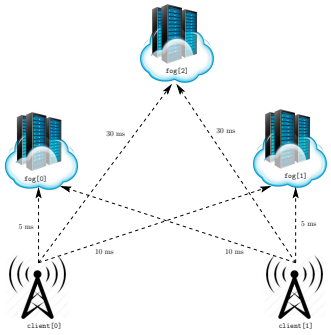


Figure 2: A simple scenario with two Fog nodes and one Cloud.

Strategy	`fog[0]`	`fog[1]`	`fog[2]`	Total
Static	24.7	43.5	0.0	68.2
ϵ-greedy	4.3	4.6	1.8	10.7
Sensible	4.5	4.6	0.9	10.0
Exp3	6.4	7.9	1.0	15.3
Softmax	5.0	5.1	0.9	11.0

Table 1: Mean number of jobs under the different task allocation strategies.

The network is simulated for 100 seconds. Each Fog node has only 5 parallel servers[1], and the service times are exponentially distributed with mean 15 ms for both nodes. In contrast, the Cloud (that is, `fog[2]`) has 100 parallel servers, and the service times in the Cloud are exponentially distributed with mean 10 ms. Jobs requests are generated according to Poisson processes, with mean 7.75 ms for the first access node and with mean 6.06 for the second one. The intensity of the traffic generated by the first access node is multiplied by 2.5 between $t_0 = 15$ s and $t_2 = 40$ s. The inter-arrival times of job requests at the second access nodes are divided by a factor 2.0 between times $t_1 = 30$ s and $t_3 = 60$ s. Note that, if we choose to always process an incoming job at the closest Fog node, it means that the utilization rate of node `fog[0]` (resp. `fog[1]`), which is initially 0.387 (resp. 0.495), becomes 0.968 (resp. 0.99) between times t_0 and t_2 (resp. t_1 and t_3). In other words, under this routing strategy, the system is stable but operates in heavy load between times t_0 and t_3. Values are averaged over $5,000$ parallel simulation runs.

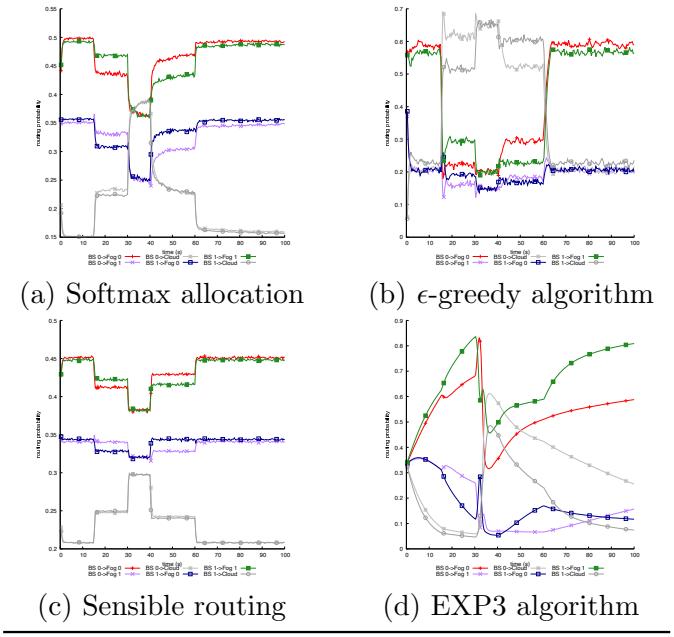
The results obtained for this scenario are shown in Tables 1 and 2.

Table 1 gives the time-average number of jobs in each datacenter under each task allocation strategy. It is clear that, even though they are very simple and assume almost no knowledge of the infrastructure, the adaptive task allocation algorithms all lead to a significant reduction of the number of jobs in the system.

We now turn our attention to Figure 3, which shows the job routing probabilities as functions of time under the different task allocation strategies. We remark that the adaptation of routing probabilities is very slow for the EXP3 algorithm as compared to other dynamic strategies. We also note that under peak traffic conditions (between times t_1 and t_2), the ϵ-greedy strategy send much more jobs to the Cloud (almost 65% for both access nodes) than the sensible routing and softmax strategies, which sends only 30% and 40% of their jobs to the Cloud between times t_1 and t_2, respectively.

[1]The number of servers in Fog nodes and in the Cloud are of course not realistic. These values have been chosen so as to reduce the simulation times.

The two latter strategies lead to similar routing probabilities.



(a) Softmax allocation (b) ϵ-greedy algorithm

(c) Sensible routing (d) EXP3 algorithm

Figure 3: Task allocation strategies.

Table 2 gives the time-average values of the response time for every source-destination pair under the different task allocation strategies. It is clear from this table that dynamic task allocation strategies lead to a significant reduction of response times. We also note that the sensible task allocation strategy outperforms the other strategies.

	`client[0]`			`client[1]`		
Strategy	`fog[0]`	`fog[1]`	`fog[2]`	`fog[0]`	`fog[1]`	`fog[2]`
Static	99.5	–	–	–	156.0	–
ϵ-greedy	46.1	56.1	70.3	56.1	48.0	70.3
Sensible	38.9	49.2	70.1	48.9	39.2	70.1
Exp3	45.2	59.2	70.1	54.7	49.0	70.1
Softmax	42.0	52.7	70.2	52.3	42.4	70.2

Table 2: Response times (ms) for each origin-destination pair under the different task allocation strategies.

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
Static	100.2	-	-	-	156.1	-
ϵ-greedy	45.9	55.6	70.0	55.8	47.8	70.2
Sensible	38.9	49.1	70.1	48.8	39.2	70.2
Exp3	45.4	58.3	70.0	48.8	39.2	70.2
Softmax	41.9	52.5	70.1	52.3	42.4	70.2

Table 3: Response times (ms) for each origin-destination pair under the different task allocation strategies under variable link delays.

5.1.2 B. Exponentially distributed Service Times - Variable Link Delays

In order to evaluate the impact of the variability of link delays on the performance, we now use a uniform distribution to model them. More precisely, we replace the constant communication delay of 5 ms by a random delay with the same mean but uniformly drawn in the interval $[3\ ms, 7\ ms]$ (the standard deviation is $\sigma = 1.15$). Similarly we replace fixed delays of 10 ms (resp 30 ms) by uniform random delays in the interval $[4\ ms, 16\ ms]$ (resp. $[20\ ms, 40\ ms]$), which yields a standard deviation $\sigma = 3.46$ (resp. $\sigma = 5.77$).

Table 3 presents the response times under the different task allocation algorithms when link delays are variable. These values have to be compared to the mean response times obtained for fixed link delays, which are given in Table 2. Sweeping through all values in both tables, we see that the variation of the mean response times never exceeds 1.2%. We thus conclude that the adaptive algorithms considered in our simulations are relatively robust to a moderate variability in the communication delays.

5.1.3 C. Pareto-distributed Service Times - constant link delays

In order to evaluate the impact of the variability of job sizes on the task allocation algorithms, we now assume that the processing times of the job follow a Pareto distribution, instead of an exponential distribution. In other words, we assume that the probability that the processing time X of a job be greater than some number x is given by

$$\Pr(X > x) = \begin{cases} \left(\frac{x_{\mathrm{m}}}{x}\right)^{\alpha} & x \geq x_{\mathrm{m}}, \\ 1 & x < x_{\mathrm{m}}, \end{cases}$$

where x_{m} is the (necessarily positive) minimum possible value of the processing time, and α is a positive parameter. If $\alpha > 2$, the Pareto distribu-

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
Static	103.1	-	-	-	161.1	-
ϵ-greedy	50.5	59.8	70.7	59.6	52.9	70.7
Sensible	44.2	54.6	70.1	53.9	45.0	70.1
Exp3.	48.6	60.1	70.2	57.6	51.2	70.2
Softmax	52.3	61.2	70.5	65.1	55.2	70.5

Table 4: Response times (ms) for each origin-destination pair under the different task allocation strategies for the first Pareto distribution of job sizes.

tion has a finite mean and variance which are given by $\alpha x_{\mathrm{m}}/(\alpha-1)$ and $\alpha x_{\mathrm{m}}^2/\left[(\alpha-1)^2(\alpha-2)\right]$. We consider two different Pareto distributions:

- **First Pareto distribution** - We choose $\alpha = 2.25$ and compute the minimum value x_{m} so as to keep the same mean values for the job processing times as in Section 5.1.1 (that is, 15 ms in Fog nodes and 10 ms in the Cloud). For the jobs executed in the Fog nodes (resp. in the Cloud), the standard deviation of the processing time is now 20 ms (resp. 13.3 ms) instead of 15 ms (resp. 10 ms) with the exponential distribution.

- **Second Pareto distribution** - We choose $\alpha = 2.05$ and, as before, we compute the minimum value x_{m} so as to keep the same mean values for the job processing times. For the jobs executed in the Fog nodes (resp. in the Cloud), the standard deviation of the processing time is now 46.8 ms (resp. 31.2 ms) instead of 15 ms (resp. 10 ms) with the exponential distribution.

We note that the variability of job sizes is greater with the first Pareto distribution than with an exponential distribution, and that its is even greater with the second Pareto distribution.

The average response times obtained are reported in Table 4. The best results are obtained with the ϵ-greedy and sensible allocations. We note that, the EXP3 allocation scheme provides better average results than the softmax allocation.

If we compare to the mean response times reported in Table 2, we see that all values increase when passing from exponentially-distributed service times to Pareto-distributed service times. This is something expected since it is well known that a greater variability in job sizes lead to larger response times. We note however the impact on response times is not the same for

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
Static	127.1	-	-	-	192.4	-
ϵ-greedy	59.3	66.0	71.4	67.9	59.9	71.5
Sensible	57.9	69.5	70.2	67.4	60.0	70.2
Exp3.	59.1	64.2	70.3	66.0	57.9	70.4
Softmax	54.1	70.9	70.7	73.4	72.3	70.7

Table 5: Response times (ms) for each origin-destination pair under the different task allocation strategies for the second Pareto distribution of job sizes.

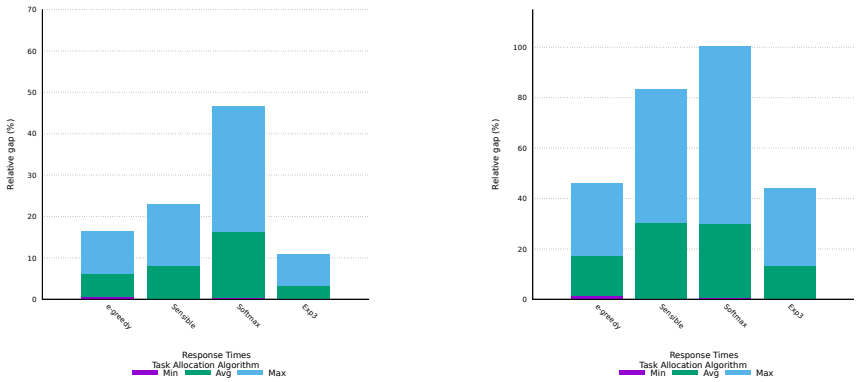
all task allocation algorithms. In order to assess the robustness of the algorithms with respect to the variability of job sizes, we consider the following metric for each access node i and each Fog node j

$$v_{i,j}^{A} = \frac{R_A^{Pareto}(i,j) - R_A^{Exp}(i,j)}{R_A^{Exp}(i,j)},$$

where $R_A^{Pareto}(i,j)$ (resp. $R_A^{Exp}(i,j)$) represents the mean response time of job requests sent by access node i to Fog node j under task allocation algorithm A and for Pareto-distributed (resp. Exponentially-distributed) service times. Using the values in Tables 2 and 4, we can compute $v_{i,j}^{A}$ for each source-destination pair (i,j) and each algorithm A, and compare the increase in response times obtained under the different algorithms. This comparison is done in Figure 4a where the minimal, maximal and average increases relative to the exponential distribution are displayed for each algorithm. Interestingly, the EXP3 allocation scheme seems to be more robust to the variability of job sizes. Indeed, we see that with EXP3 the mean response times increase by at most 7.5% when we pass from an exponential distribution to the first Pareto distribution. The increase in response times is as high as 30.2% (resp. 14.8%) for the Softmax allocation (resp. sensible allocation). Although slightly less robust, the ϵ-greedy allocation is also quite robust to the variability of job sizes since the increase in the mean response times is at most 10.2%. While the minimal variations observed are non-significant, we see that the average relative increases is smallest in EXP3 with 3.2%, confirming its good performance. ϵ-greedy remains quite robust when looking at that metric with an increase of 5.7%, followed by sensible routing at 8.6%. However, Softmax allocation does not handle the variation well with an average increase of 16%.

Similarly Table 5 and Figure 4b present the results obtained with the second Pareto distribution, comparing them with the exponential distribu-

tion. The response times from the exponential distribution to the second Pareto distribution increase the least in ϵ-greedy and EXP3 allocation with $28,6\%$ and $30,7\%$ increase. We note much larger gaps in Sensible routing and Softmax with 53% (resp. 70%). The average increase is best in EXP3 with 13.1%, then with ϵ-greedy with 16%. Surprisingly, while the maximal relative increase was significantly worse in Softmax, Sensible routing and Softmax allocation both obtain comparable average increases with respectively 30.3% and 29.3%.



(a) First Pareto distribution

(b) Second Pareto distribution

Figure 4: Relative performance degradation between the first/second Pareto distribution and exponential distribution for service times across task allocation algorithms.

5.2 Second Scenario: Task Allocation with offloading

The second scenario is similar to the first one, except that the Fog nodes are now connected to the Cloud (Figure 5). The communication delay between the Fog nodes and the Cloud is constant and equal to 28 ms. Note that this value was chosen so as to satisfy the triangle inequality. In this scenario, Fog nodes can offload jobs to the Cloud. All other parameters are similar to those used in Scenario 1.

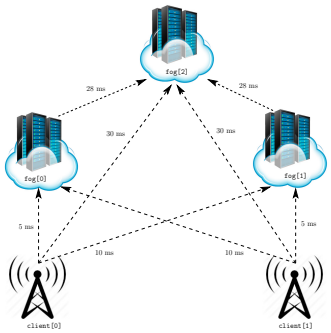


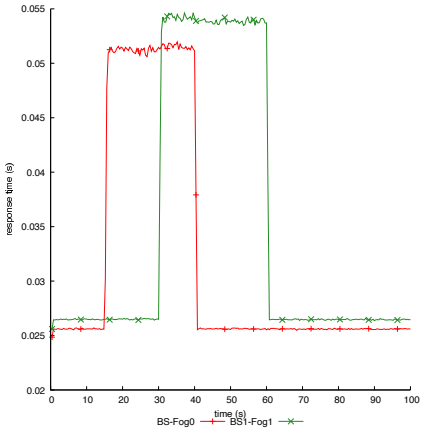
Figure 5: A scenario in which Fog nodes can offload jobs to the Cloud.

The offloading mechanism that we consider assumes that the execution time of a job in the Cloud is constant and equal to the processing time of the job in the Cloud (i.e., it assumes that a job request sent to the Cloud always find a free server). Under this assumption, a job request offloaded to the Cloud c by Fog node j will have a response time equal to $\ell_{j,c} + \frac{1}{\mu_c} + \ell_{c,j}$, which yields 66 ms with the values used in this scenario. The mechanism is then as follows. Upon reception of a job, the dispatcher of the Fog node j first selects the application server n for executing this job. It then queries the number of jobs q_n at this server, so as to estimate the execution time of the job using the formula $(q_n + 1) \times \frac{1}{\mu_j} = (q_n + 1) \times 15$ ms. If this execution time is greater than the response time obtained by offloading the job to the Cloud, then the job is offloaded to the Cloud.

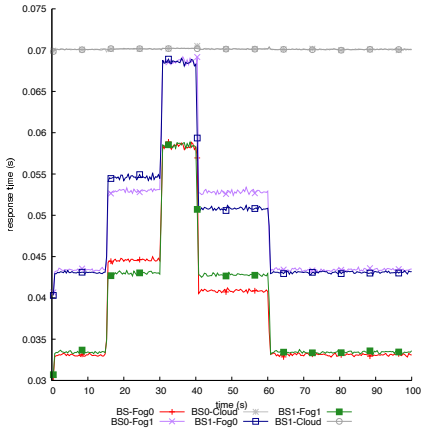
We note that this dispatching mechanism requires the dispatcher to keep track of the number of jobs executing at each server. We also note that this mechanism depends on the job dispatching scheme used in the Fog nodes. If the Fog nodes use the "Join the Shortest Queue" scheme, then the execution time estimated by the dispatcher corresponds to the minimum execution time that can be achieved. However, for another dispatching scheme such as a random allocation, the estimated execution time will be greater. We emphasize that the information used in this offloading mechanism is completely different from the one used by the adaptive routing algorithms discussed so far, even though the information used by the offloading mechanism is either local (number of jobs executing at each server) or static (response time from the Cloud). In the following we shall assume that all Fog/Cloud nodes use the *Join-The-Shortest-Queue* policy to select the server to which a task is allocated.

Figure 6 compares the performance obtained under different strategies. The first one uses only job offloading to the Cloud by Fog nodes and a static allocation strategy routing tasks to the closest node (in this case always Fog node 0 for the first base station and Fog node 1 for the second base station). The second one uses only an adaptive task allocation strategy (which is

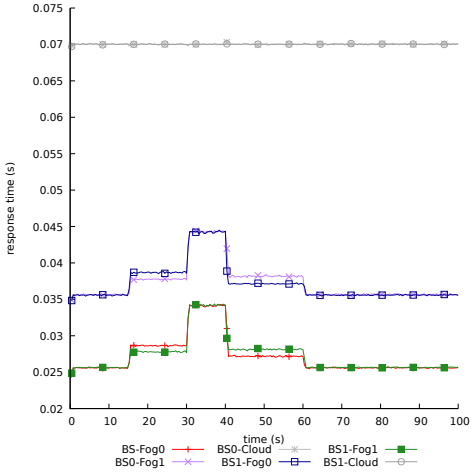
sensible routing in this case), but once affected to a Fog node, jobs cannot be offloaded to the Cloud. Finally, the third one corresponds to the case where both mechanisms are combined.



(a) Offloading only.



(b) Sensible routing only.



(c) Offloading and sensible routing

Figure 6: Sensible routing vs Offloading only vs Sensible routing and Offloading for exponentially distributed service times

As we can observe the offloading mechanism enables a significant reduction of response times. This mechanism outperforms adaptive task allocation strategies in this scenario. We note however that the use of an adaptive task allocation algorithm in combination with an offloading strategy drops response times.

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
ϵ-greedy	33.7	44.7	70.0	44.7	34.7	70.0
Sensible	27.2	37.3	70.0	37.2	27.3	70.0
Exp3.	29.2	41.7	69.9	39.2	31.7	70.0
Softmax	27.9	38.2	70.0	38.0	28.1	70.0

Table 6: Response times (ms) obtained in the second scenario by combining job offloading and adaptive task allocation in the case of exponentially-distributed service times.

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
ϵ-greedy	34.6	45.4	70.1	45.6	35.7	70.1
Sensible	27.0	37.1	70.0	37.0	27.1	70.0
Exp3.	28.8	41.1	69.9	38.8	31.0	69.9
Softmax	28.2	38.4	70.0	38.2	28.4	70.0

Table 7: Response times (ms) obtained in the second scenario by combining job offloading and adaptive task allocation for the first Pareto distribution of service times.

5.2.1 Pareto distributed Service Times

Once again, we aim to evaluate the impact of the variability of job sizes on the task allocation algorithms. We compare the results obtained with the offloading configuration using an exponential service distribution and the first Pareto distribution described in subsection 5.1.3.

We look at these results further in Table 7 showing the average response times from our simulation using the first Pareto distribution. Comparing it with Table 6 which uses the exponentially distribution for service times, we notice surprisingly that EXP3 and sensible routing perform slightly better in this distribution with an average decrease of response times of 1% (resp. 0.4%). The Softmax and ϵ-greedy algorithms, on the other hand, perform worse with average response times seeing a maximum increase of 1.1% (and 2.9%). We suppose that the relative stability of these algorithms under job size variability is compensated by the offloading strategy in the Cloud which always can handle the jobs even when the Fog node queues are full due to the increase in the service times.

6 Conclusion

Our simulation results show that very simple task allocation algorithms yield remarkable improvements in terms of average response times with respect to a static allocation strategy. It should be emphasized that all these algorithms are very simple to implement as they require neither cooperation between base stations nor knowledge of the physical infrastructure. Our results also show that, although significant gains on response times are obtained when Fog nodes are allowed to offload their tasks to the Cloud, a far more efficient solution is obtained by combining an offloading mechanism and an adaptive task allocation strategy. The main advantage of the latter solution is that it considers all Fog nodes as a single pool of resources to which job requests can be allocated. Finally, we have noticed that *Sensible Routing* provides the best response times, but that EXP3 is more robust to the variability of job processing times.

Acknowledgment

We gratefully acknowledge the funding received from Continental Digital Services France.

References

[1] CISCO Fog Computing White paper. `https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf`.

[2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The non-stochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[3] Jung-Yeon Baek, G. Kaddoum, S. Garg, K. Kaur, and V. Gravel. Managing fog networks using reinforcement learning based load balancing algorithm. *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–7, 2019.

[4] R. Beraldi, A. Mtibaa, and H. Alnuweiri. Cooperative load balancing scheme for edge computing resources. *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 94–100, 2017.

[5] Xiaolan Cheng, Xiaoxia Zhou, Congfeng Jiang, and Jian Wan. Toward computation offloading in edge computing: A survey. 2019.

[6] Eduardo Cuervo, Aruna Balasubramanian, Dae ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. pages 49–62. ACM, 2010.

[7] C. Fricker, F. Guillemin, P. Robert, and Guilherme Thompson. Analysis of an offloading scheme for data centers in the framework of fog computing. *ArXiv*, abs/1507.05746, 2016.

[8] E. Gelenbe. Random neural networks with negative and positive signals and product form solution. *Neural Computation*, 1:502–510, 1989.

[9] L. Lin, Xiaofei Liao, H. Jin, and Peng Li. Computation offloading towards edge computing. 2019.

[10] Liqing Liu, Zheng Chang, Xijuan Guo, S. Mao, and T. Ristaniemi. Multiobjective optimization for computation offloading in fog computing. *IEEE Internet of Things Journal*, 5:283–294, 2018.

[11] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction (Second Edition)*. MIT Press, 2018.

[12] L. Wang and E. Gelenbe. Adaptive dispatching of tasks in the cloud. *IEEE Transactions on Cloud Computing*, 6(1):33–45, 2018.