# CLOSE: A Costless Service Offloading Strategy for Distributed Edge Cloud

Farah Slim, Fabrice Guillemin, Yassine Hadjadj-Aoul

# CLOSE: A Costless Service Offloading Strategy for Distributed Edge Cloud

Farah Slim
Orange Labs (France)
Email: farah1.slim@orange.com

Fabrice Guillemin
Orange Labs (France)
Email: fabrice.guillemin@orange.com

Yassine Hadjadj-Aoul
University of Rennes 1 (France)
Email: yhadjadj@irisa.fr

***Abstract*—New bandwidth-intensive and time-constrained services in 5G networks combined with network function virtualization is pushing network operators to deploy distributed cloud infrastructures at the edge of the network. Allocating resources in capacity-limited infrastructures raises new challenges, which have not really been so far considered in the cloud literature. In this context, we investigate placement and offloading strategies of constrained services. We set design principles of future distributed edge clouds in order to meet application requirements. We precisely introduce a costless distributed resource allocation algorithm, named *CLOSE*, which considers local information only. We compare via simulations the performance of *CLOSE* against those obtained by using mechanisms proposed in the literature, notably the Tricircle project within OpenStack. It turns out that the proposed distributed algorithm yields better performance while requiring less overhead.**

## I. Introduction

During the last decade, carrier grade virtualization technologies have been very successful in offering on-line services via on-demand computing and storage capacities in the cloud (cf. EC2 by Amazon, Azur by Microsoft, etc.). While cloud resources were so far used to run applications owned by end users (residential or business customers), various initiatives in the design of 5G networks, including ETSI Network Function Virtualization (NFV) [1] and ECOMP [2], explore the virtualization of network functions. This is a groundbreaking evolution that will impact not only the architecture of networks but also the business models of network operators, who will become providers of IT infrastructures.

NFV raises many issues in terms of robustness, resilience, security, etc. but also with regard to resource allocation. Indeed, some functions with no strict real-time constraints can clearly be hosted in big centralized cloud platforms (e.g., authentication, address allocation, etc.). Some others such as Radio Access Network (RAN) functions, firewalls, deep packet inspection, etc. have to be executed close to end users. This imposes to deploy data centers at the edge of the network in order to meet real-time requirements [3]. If we assume that an edge data center is installed in each Point of Presence (PoP) of an IP network, then a few hundreds of edge data centers are necessary to equip a national network in an European country.

It is very likely that edge data centers will be installed with limited capacities and regularly upgraded as it is the case today for transmission links in IP backbone networks. Implementing services, which were so far hosted in centralized cloud platforms with an assumption of infinite resources, rises new challenging issues. In fact, the upgrade of capacities becomes an issue in terms of operational expenditure, when there are hundreds of data centers disseminated at the edge of a network. Moreover, supporting network functions introduces additional dimensions to the resource allocation problem, namely storage and computing to be combined with bandwidth.

In the cloud literature, request blocking is most of the time overlooked because of the infinite capacity assumption mentioned above or thanks to overbooking. In data centers, resources are often overbooked in the sense that more requests are accepted than the data center can actually handle at a given time. To ensure access to resources a queue is set up to schedule jobs while respecting some fairness criteria.

Overbooking can, however, be performed only within certain limits. If too many requests are accepted by a data center, then too many jobs will compete for accessing resources, and the job queue will overflow. Even in the case of a pure egalitarian fair sharing scheme of resources, a job may receive a so small fraction of resources that the job cannot be executed or the quality perceived by the end user is extremely bad because of response times above acceptable thresholds. We consider in this paper that requests are blocked, when overbooking has reached these limits.

We address resource allocation from a game theory perspective, based on two classes of players. This is a preliminary formulation of the problem. We assume that players use a commodity infrastructure (the IT infrastructure of the network). We do not assume any interaction between players. We only consider the utility (or reward) function of the network.

The organization of this paper is as follows: In Section II, we review some resource allocation algorithms in cloud platforms. In Section III, we state our basic design principle and we introduce our distributed algorithm for resource allocation. We shall, particularly, focus on the service offloading strategy and formulate the problem by means of game theory. We subsequently compare its performance against those obtained by using other algorithms, notably the Tricricle approach of OpenStack, in Section IV. Some concluding remarks are presented in Section V.

## II. Related work on resource allocation

In the recent few years, many research works have addressed the problem of resource allocation in distributed cloud envi-

ronments [4]. However, most contributions have considered configurations in which the placement decision runs in a centralized platform. In such a context, a popular approach is to adopt an optimization formulation: Given a demand for resources in terms of storage and computing, the problem is to find the optimal request placement. This leads in general to Mixed Integer Linear Programming (MILP) problems.

In [5], an optimization problem is proposed for the placement of VNFs across a distributed cloud. It can be observed from that paper that MILP takes less than one second to run for an infrastructure comprising 5 data centers, while it needs several tens of minutes for only 20 data centers. Hence, this approach will hardly scale with the size of a distributed data center system

To achieve optimal resource placement, an exact formulation that aims at finding the best placement of resources by maximizing the revenue and minimizing the corresponding cost is proposed in [6]. The authors have also noted that the ILP formulation suffers from scalability problems. They then proposed an alternative approach via dynamic resource placement by representing the resource allocation problem by a directed graph and by using a minimum cost maximum flow algorithm for resource placement. To compute the minimum cost maximum flow in the graph, the Edmonds-Karp algorithm is used. This approach does not consider, however, the latency constraints of requests.

Since optimization approaches can be very time-consuming and may suffer from scaling issues, several works propose alternative approaches. In [7], an algorithm for network-aware allocation of virtual machines in distributed cloud systems is studied. By representing the distributed cloud system as a complete graph, where vertices represent data centers, weights represent the number of available virtual machines or data center capacities, edges represent links between data centers and labels represent the number of hops or distance, the proposed algorithm selects first the relevant data centers to serve a user request and then the physical machines to run the virtual machines. Even if this selection aims at minimizing the maximum distance among virtual machines running the request and therefore the bandwidth usage, this algorithm applies only if the total amount of network traffic between virtual machine is known.

In [8], a cloud management middle-ware is proposed in order to reduce web application response time by migrating virtual machines closer to end users. In [9] a high locality scheduling for an edge cloud environment that reduces the networking costs is presented. In [10] a resource allocation algorithm for distributed cloud system is proposed with the primary objective of minimizing the overall operating cost, which is a trade-off between energy cost and WAN cost, when energy price is not the same across different geographical locations.

Last but not least, the Openstack community has created the Tricircle project to cope with distributed cloud architectures (refer to https://wiki.openstack.org/wiki/Tricircle). The main objective of the project is: (1) to allow cloud capacity expansion, by adding new instances; (2) to improve reliability and availability through supporting a geo-distributed cloud architecture; (3) to reduce bandwidth usage by allocating resources close to end users on each site. With regard to resource allocation, the data center with the maximum amount of available resources from a user's availability zone (sub-list of data centers) is selected to accommodate a request. The selection of the most appropriate physical machines to place the request, within a data center, is made locally by the Nova and Cinder schedulers managing the data center. Since one availability zone could include several groups of data centers, if one data center reaches the limit of the resource utilization, the request will be rerouted to another data center but in the same zone.

It should be emphasized that most of the above-mentioned works consider a global knowledge about resource utilization. Additionally, some of the contributions cannot be applied in realistic use cases with a dynamical arrival of user requests. Finding a strategy, which is based only on local knowledge with no signaling overhead, represents the main focus of the present research work.

## III. CLOSE: A Services' Offloading Algorithm for Distributed Edge Cloud

### A. Preliminary considerations

Network function placement is a complex problem depending on several parameters. This problem is all the more difficult when one considers the dynamics of the allocation of resources. In this context, it is natural to consider geographical aspects related to the origin of requests, especially for constrained functions (i.e., latency, overhead, bandwidth, security ...). Indeed, some functions such as firewall, deep packet inspection, etc. have to be placed close to end users (e.g., to prevent users from sending confidential data through the Internet). Some others such as authentication, IP address allocation, etc. with looser time constraints can be placed in a distant data center.

Generally speaking, requests may be made of components (i.e., sub-functions) having various requirements in terms of latency. Some of them may have stringent requirements, for instance a response time of the order of a few milliseconds, while some others may be more tolerant with regard to delay. The placement of sub-functions could be in principle distributed over several data centers as long as the global response time requirements are met. In this paper, we assume, however, that a function with strict latency requirements, possibly composed of several sub-functions, shall be instantiated on the same edge data center or be rejected. This leads us to claim our first design statement.

*Design principle 1:* Instantiate a function with latency constraints on the same edge data center instead of spreading it on many data centers.

Generally speaking there is a design choice between optimization and dimensioning. This latter task consists of assessing the amount of resources needed to accommodate a resource demand with a prescribed very small rejection rate.

Optimization may lead to better acceptance rate of requests but also to send traffic to a sub-function hosted by an edge data center and then back for further treatment. To avoid this traffic "tromboning" effect, we state that a complete function shall be hosted by an edge data center or rejected.

*Design principle 2:* Avoid traffic "tromboning" in the network due to function splitting.

To achieve a request acceptance rate objective, edge data centers have to be dimensioned according to the demand in terms of resources (compute, storage, bandwidth). We thus transform an optimization problem into a dimensioning problem.

*Design principle 3:* Dimension edge data centers instead of optimizing function placement.

Dimensioning edge data centers differ from the traditional transmission link dimensioning problem in telecommunications networks. On the one hand, we have to take into account more parameters beyond the sole bandwidth resource. A multidimensional Erlang formula can, nevertheless, be expected [11]. On the other hand, a request can be displaced as long as response time requirements are met. This introduces some flexibility in the acceptance of a request. An Erlang loss formula taking into account potential migration of requests among a possible set of servers, capable of meeting request requirements, is still an open problem.

### B. Algorithm principles

To allocate resources in a distributed data center system by taking into account the location of requests, we clearly have two possibilities. In the first one, there is a centralized entity which has a view of the resources available in the various data centers and depending upon the location a request is issued from, this central dispatcher can select those data centers, which can accommodate the request while respecting the constraint on the maximum displacement of the request; once this set of data centers is known, the dispatcher can pick up a data center at random or one among those with the maximum amount of available resources or with the better score [12]. If all data centers able to respect the displacement constraint are occupied, then the request is simply rejected. It is worth noting that this approach is in line with the current Tricircle approach of OpenStack, which in addition uses the concept of area. This approach is refereed to as the centralized approach. As discussed in the Introduction, we assume in this paper that edge data centers are operated up to a certain overbooking limit of resources. If an edge data center is too loaded, a request is blocked.

For the second possibility, a user request can be intercepted by the first data center on the data path. The Distributed resource allocation algorithm that we propose is as follows:

1) When a request arrives in the system, the request is intercepted by the first data center along the data path.
2) If the request cannot be accommodated by this edge data center (i.e., when the function `IsAvailable` returns **False**), then, it is forwarded to one of its neighbors, which may respect the time constraints of the service. A Time To Live (TTL) field can, also, be considered to limit the displacement of the request.
3) To forward the request, the edge data center takes into account the number of redirections from its neighbors and the time constraints[1]. Specifically, an edge data center maintains a counter, which records the moving average number of redirections (deflected requests) from its neighboring edge data centers. The edge data center with the smaller number of deflected requests is chosen. The request is forwarded with the label of the deflecting data center in order to avoid loops.
4) The redirected request is examined by the edge data center, the request is forwarded to, and the TTL value is decreased accordingly. If the request can still not be accommodated and if the TTL field is non null and time constraint can be met, then the previous step is repeated otherwise the request is discarded using the `Discard` function.

The pseudo-code of the proposed mechanism is illustrated in Algorithm 1 by using the notation summarized in Table I.

TABLE I
NOTATION USED IN THE DISTRIBUTED ALGORITHM.

| Variable | Description |
|---|---|
| $N$ | Number of requests |
| $DC_i$ | Data center $i$ |
| $l_{i,j}$ | Latency between $DC_i and DC_j$ |
| $d_{i,j}$ | Number of deflected requests from $DC_i$ to $DC_j$ |
| *Rq* | A request including: the amount of requested resources, the maximal hops in terms of *TTL* and latency $l_{max}$, and the cumulative latency |

The major difference between the proposed algorithm and the centralized one is the amount of information to be exchanged between the various edge data centers and the central dispatcher. For the centralized algorithm, each time a request is accepted by or leaves an edge data center, then this data center has to send an update of available resources to the central dispatcher so that this latter maintains accurate information about the occupancy of the system. In big systems, with several hundreds of distributed data centers, this may represent a significant overhead. For the distributed algorithm, such information has not to be exchanged between edge data centers since only local information is used. The counterpart is that the forwarding of a request is performed with less information (and hence less accuracy).

There is clearly a trade-off between the accuracy of the placement of requests and the cost to maintain accurate information. This is a classical issue in telecommunications networks and has been so far solved by using monitoring tools and regular upgrade of network capacities. In the present case, we use an additional degree of freedom by allowing the displacement of requests up to a certain limit; this is impossible with bandwidth in classical networks, even for

[1]The current data center selects a sub-list of data centers respecting the request criterions using the `GetNeighborsIdx`, which returns the set of possible data centers.

**Algorithm 1** CLOSE algorithm for services' offloading

```
1: procedure FORWARD(DC_cur,DC_ori,Rq)
2:     if IsAvailable(DC_cur, Rq) then
3:         Allocate(DC_cur, Rq)
4:     else
5:         Rq.latency ← Rq.latency + l_cur,ori
6:         Rq.TTL ← Rq.TTL − 1
7:         J ← GetNeighborsIdx(DC_cur, Rq) \ {DC_ori}
8:         if J ≠ {} then
9:             dst ← arg min_{j∈J} {d_j,cur}
10:            FORWARD(DC_dst,DC_cur,Rq)
11:        else
12:            Discard(Rq)
13:        end if
14:    end if
15: end procedure
16:
17: while True do
18:    Rq ← getRequest()
19:    DC_cur ← getClosestDC(Rq)
20:    FORWARD(DC_cur,DC_cur,Rq)
21: end while
```

elastic traffic, which can severely suffer from congestion (e.g., flows with very small bit rates leading to very poor quality of experience). In the following, we shall see that displacement allows local congestion to be absorbed by the system.

## IV. Performance Evaluation of the CLOSE Algorithm

To assess the performance of our proposal, we have different strategies. In a first one, we do not consider any offloading. In other words, edge data centers does not collaborate, which means that a request is rejected if there is not enough resources to accommodate it. We refer to this algorithm as "Isolation" since an overloaded edge data center cannot take benefit of the possible available capacity in the global system.

A second strategy relies on a central dispatcher which is aware of the occupation of all edge data centers and selects the one with the most available capacity and respecting the constraints of the service (without considering latency), even if the latter data center is not the closest to the origin of the request. This algorithm is referred to as "Full Sharing".

We have, also, considered the algorithm used by Openstack, where the centralized dispatcher has to maintain an updated view of the infrastructure topology as well as the resource utilization level. Respecting the maximum displacement constraint of a request, the dispatcher has to select those data centers, which are eligible to accommodate it. Those candidate data centers are mapped onto a geographically area from which the request can be serviced. This area can be mapped to an "Availability Zone" according to the Openstack terminology. Technically, each Openstack project implements it differently – with regard to resource allocation – in a manner to enable logical subdivision of resources.

Furthermore, we have considered requests with two types of latency requirements. The first class has stringent latency requirements (namely, a response time less than 4 ms) while the other class is more delay tolerant (10 ms). Data centers take local decisions and we evaluate the global blocking rate.

This problem can be viewed as a game with two classes of players, who compete for resources at the various data centers. Contrary to classical gaming problems, the players do not apply specific strategies to maximize their reward functions. They try to place to place their requests and leave the system in the case of blocking. In the present case, only the network applies a resource allocation strategy to maximize its own reward function (acceptance rate), which amounts to minimizing the overall blocking rate. The introduction of strategies by users (for instance, by relaxing latency requirements to increase acceptance rate) is for further study. In this case, the network appears as an additional player.

### A. Simulation settings

To study the performance of a distributed cloud system, we have considered a realistic network of 21 data centers with different capacities located at the edge of an Autonomous System, as illustrated in Figure 1. We have considered the structure of the network of Orange, in which the distance between small data centers, located at Main Central Offices (MCO), is 100 km from Core Central Offices (CCOs), equipped with bigger data centers. CCOs are connected to a big centralized data center at a distance of 300 km[2]. Latencies are computed by using the speed of light in fiber.

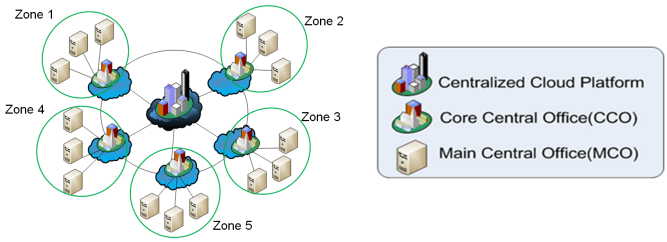


Fig. 1. Network topology.

Request arrivals are assumed originating from one of the considered regions according to a Poisson process, which has proven realistic for a number of real traffic arrival processes [13]. For the sake of clarity, only one type of resource is considered in this paper, typically the CPU. Similar results can be obtained using multiple resources as we demonstrated in our previous work [11]. As mentioned above, we have considered two profiles of requests. Profile 1 is assumed to have strong requirements in terms of latency (4ms), while Profile 2 is more delay tolerant (10 ms).

[2]For the sake of confidentiality, the DC locations and real distances used in the simulations are not given in the paper.

The global load of the system (data centers) is defined as:

$$\rho \stackrel{def}{=} \sum_{i=1}^{N} \frac{\rho_i}{\mathbb{C}_i}$$

where $\rho_j \stackrel{def}{=} \lambda_j/\mu_j$ is the load of Data Center $j$ (for short, DC$j$) and $\lambda_j$ and $1/\mu_j$ represent the arrival rate and the exponentially distributed holding time of resources at DC$j$, respectively. Data centers (DCs) are unevenly loaded; we only consider the global load $\rho$ of the system given that some DCs are overloaded while others are underloaded.

In order to compare the various allocation schemes, we introduce the average blocking rate defined as the fraction of requests, which are eventually rejected by the system:

$$\overline{\beta} = \frac{1}{\Lambda} \sum_{j=1}^{N} \lambda_j \beta_j$$

where $\Lambda = \sum_{j=1}^{N} \lambda_j$ is the global arrival rate and $\beta_j$ is the blocking rate of requests originally arriving at DC$j$. More precisely, $\beta_j$ is the fraction of requests which are originally arriving at DC$j$ but eventually not accepted by the system.

### B. Simulation Results

The average blocking rates of the system under the various allocation strategies are given in Figure 2. This figure displays the blocking probability versus traffic intensity under three different regimes: underload ($\rho < 1$), critical ($\rho = 1$) and overload ($\rho > 1$). Simulation results are averaged to obtain confidence intervals with a 95% confidence level.

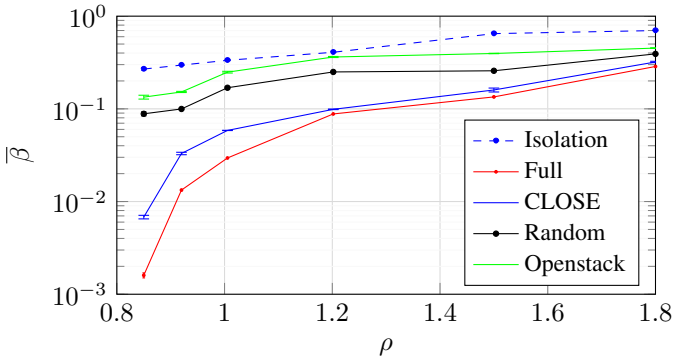


Fig. 2. Blocking rates under different load conditions.

Results show that collaboration between data centers significantly reduces blocking of requests. We verify that the *Isolation* scheme yields the worst performance in terms of rejection. Blocking is obviously minimal with the Full Sharing algorithm when the global dispatcher has a full view of the occupancy of the system, but this strategy does not take into account the latency constraints.

It is worth noting that the proposed policy *CLOSE*, which counts the average number of deflections of each neighboring data center, significantly reduces blocking when compared against the *Isolation* allocation and yields a performance comparable to that obtained with the *Full* Sharing strategy. We clearly see that deflecting requests can significantly reduce blocking and share load on data centers. Moreover, the proposed scheme with no exchange of information between data centers performs well and even better than the Tricircle approach of OpenStack, where load is shared only within a geographical area.

To further evaluate the performance of the *CLOSE* algorithm, let us emphasize another key difference when compared to the *Full* Sharing strategy. The displacement of requests for the Full Sharing algorithm may be larger than that for the distributed one, since the centralized algorithm only takes into account resources and not displacement constraints.

To illustrate this latter point, we have studied the Latency Distribution of accepted requests for both algorithms. In Figure 3, we have plotted the latency Cumulative distribution for an overloaded System where the system load is $\rho = 1.2592$. We see that *CLOSE* leads to the lowest latency distribution since most of requests are serviced from the edge data center closest to the end user. Hence, the proposed algorithm performs better with regard to displacement than the *Full* Sharing algorithm while offering comparable blocking.

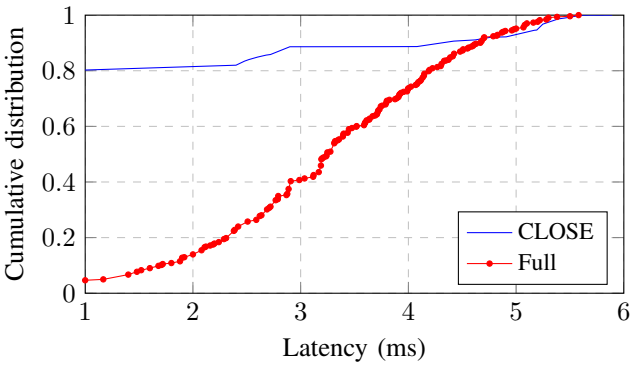


Fig. 3. CDFs of Latency: CLOSE vs Full Sharing.

For the same experiment, we have also plotted in Figure4 the latency distribution function for both profiles introduced in the previous section to show how the *CLOSE* algorithm respects request constraints in terms of latency.

Another key metric for evaluating the performance of the proposed algorithm is the load balancing index calculated on the basis of the Jain's fairness index. Figure 5 illustrates the fairness index for load balancing under the different strategies. Except the *Isolation* scenario, which leads to a totally unfair system, results are slightly different and comparable to that obtained with the *Full* sharing strategy considered as the most fair strategy.

In Figure 6, we have evaluated how the squared coefficient of variation ($CV^2$) of the loads of data centers, which reflects the degree of homogeneity of the loads between the DCs, impacts the blocking probability. The bigger is the coefficient, the more heterogeneous is the load between DCs. As the Openstack strategy is based on load sharing within the same

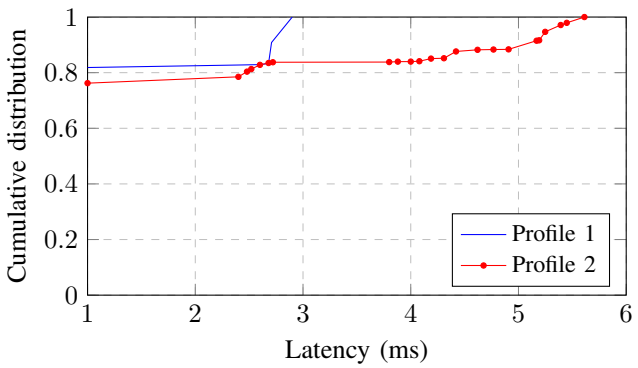


Fig. 4. Per-profile CDFs of Latency for the *CLOSE* algorithm.

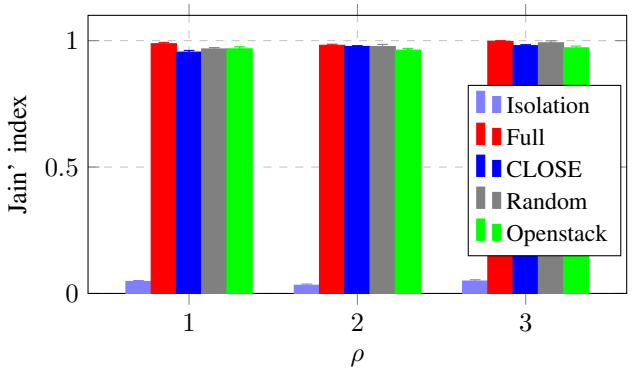


Fig. 5. Jain's index for resource utilization level.

zone (area), the performance of this strategy strongly depends on how the system is loaded, as it can be seen in Figure 6. In other words, if the zones are homogeneously loaded, this strategy ensures good load balancing in each zone by choosing the less loaded edge data center, which leads to a fair system and yields to good performance. In contrast with this latter, the *CLOSE* algorithm, which is able in some cases to distribute loads far from the origin request, yields to performance comparable to that obtained with *Full* Sharing.

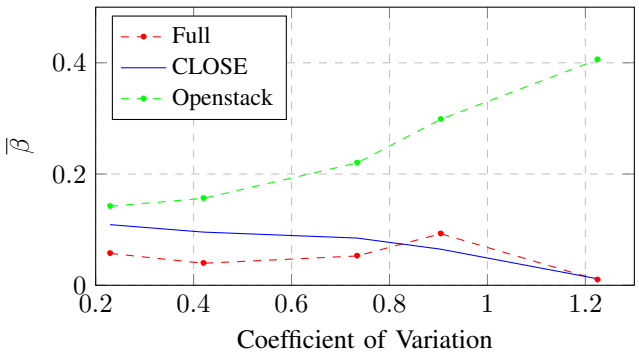


Fig. 6. Algorithm Behavior under different load conditions.

## V. Conclusion

We have devised in this paper basic design principles for handling requests with displacement constraints in a large system of distributed edge data centers. Our claim is that traffic "tromboning" and request splitting (especially for applications or virtualized network functions with stringent response time requirements) should be avoided. Moreover, adequately dimensioning edge data centers is preferable to optimizing request placement.

In this framework, we have proposed an algorithm for placing requests in a large distributed cloud platform with minimal exchange of information. Contrary to the OpenStack approach, notably the Tricircle project, this algorithm is based on local information only. In spite of minimal information, this algorithm can mitigate overload at some data centers by using a simple redirection principle by exploiting deflection information between data centers. This is a very promising result as the proposed algorithm allows a network operator to easily manage a large distributed infrastructure.

The algorithm introduced in this paper could be improved by introducing thresholds in data centers in order to early displace delay-tolerant requests and thus to improve the global acceptance rate. This point is addressed in [14] in the framework of the Open Network Automation Platform (ONAP).

## References


[1] "Introductory white paper, ETSI NFV," ETSI White Paper.

[2] "ECOMP (enhanced control, orchestration, management & policy) architecture white paper," AT&T Inc.

[3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5g," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

[4] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, S. U. Khan, and A. Zomaya, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," *Computing*, vol. 98, no. 7, pp. 751–774, 2016.

[5] Z. Abbasi, M. Shirazipour, and A. Takacs, "An optimization case in support of next generation NFV deployment," in *7th USENIX Workshop on Hot Topics in Cloud Computing*, 2015.

[6] M. Hadji and D. Zeghlache, "Minimum cost maximum flow algorithm for dynamic resource allocation in clouds," in *Cloud Computing (CLOUD), 2012 IEEE 5th Int. Conf. on*. IEEE, 2012, pp. 876–882.

[7] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *Infocom, 2012 proc. IEEE*, 2012, pp. 963–971.

[8] B. Malet and P. Pietzuch, "Resource allocation across multiple cloud data centres," in *Proc. of the 8th Int. Workshop on Middleware for Grids, Clouds and e-Science*. ACM, 2010, p. 5.

[9] A. Jonathan, A. Chandra, and J. B. Weissman, "Awan: Locality-aware resource manager for geo-distributed data-intensive applications," in *2016 IEEE International Conference on Cloud Engineering, IC2E 2016, Berlin, Germany, April 4-8, 2016*, 2016, pp. 32–41.

[10] K. Chen, Y. Xu, K. Xi, and H. J. Chao, "Intelligent virtual machine placement for cost efficiency in geo-distributed cloud systems," in *2013 IEEE Int. Conf. on Communications (ICC)*, 2013, pp. 3498–3503.

[11] F. Slim, F. Guillemin, and Y. H. Aoul, "On virtual network functions' placement in future distributed edge cloud," in *6th IEEE International Conference on Cloud Networking, CloudNet 2017, Prague, Czech Republic, September 25-27, 2017*, 2017, pp. 12–15.

[12] "Openstack Home," docs.openstack.org/developer/nova/filterscheduler.html/.

[13] R. G. Clegg, C. Di Cairano-Gilfedder, and S. Zhou, "A critical look at power law modelling of the internet," *Computer Communications*, vol. 33, no. 3, pp. 259–268, 2010.

[14] F. Slim, F. Guillemin, A. Gravey, and Y. H. Aoul, "Towards a dynamic adaptive placement of virtual network functions under onap," in *O4SDI Workshop at SDN-NF 2017 Conference, Berlin, November 2017*, 2017.