Noisy neighbor detection and avoidance for network slicing in 5G

Hanane Biallach, Marouen Mechtri, Chaima Ghribi

Noisy neighbor detection and avoidance for network slicing in 5G

Hanane Biallach, Marouen Mechtri, Chaima Ghribi

Noisy neighbor detection and avoidance for network slicing in 5G

Hanane Biallach, Marouen Mechtri and Chaima Ghribi
Orange Labs, Paris, France
{hanane.biallach, marouane.mechteri, chaima.ghribi}@orange.com

***Abstract*—This paper addresses the problem of noisy neighbor in the context of network slicing in 5G networks. Noisy neighbor is considered as an important issue since VNFs or VMs running on the same physical machine compete for resources, such as CPU, memory or bandwidth, which can degrade their performance. In this paper, we present a new approach based on supervised learning and optimization that detects and avoids bad neighboring VNFs through intelligent VNFs placement and migration.**

***Index Terms*—NFV, Machine learning, Noisy Neighbor, Integer Linear Program, Network Slicing**

I. Architecture for noisy neighbor detection and avoidance in NFV environment

Figure1 depicts our proposed framework to detect and solve the noisy neighbor problem. The architecture is composed of two main modules: **Noisy neighbor detection module:** monitors the VNFs using Prometheus [1] to detect noise by applying Machine Learning methods. **Placement and migration module:** embeds our Integer linear Program (ILP) model for VNF placement and migration and interacts with the Cloud manager (OpenStack controller) to instantiate or migrate VNFs.

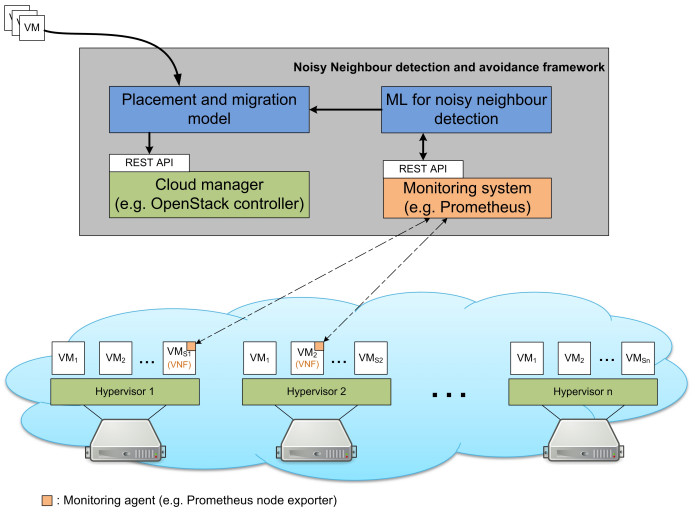

Fig. 1: Noisy neighbour detection and avoidance framework

A. ML process for noisy neighbor detection

To detect the noise, we need firstly enough data. As we monitored the virtualized infrastructure, the following metrics were collected: CPU utilization, Usage of Memory, Inbound network traffic and Outbound network traffic. After **collecting** data through Prometheus API, data **cleaning** is needed, it is the process of standardizing the data to make it ready for analysis. Once data are cleaned, we labeled each instance to one of these following statuses: 1) Noise: present the situation where the collocated VMs, running in the same server, compete for the available resources which lead to a degradation of performance. 2) Normal: present the absence of noisy neighbor phenomenon. 3) Overload: define the situation where the VNF is stressed by applications running on it. 4) overprovisioning: describes the situation where a VM is allocated in an overloaded server (a server that runs out of resources). The overprovisioning issue is another definition of noisy neighbor which have been observed during our experimentations. It's an important observation for the related work since it's the first time that overprovisioning is cited as a noisy neighbor problem.

As Machine learning methods, we applied K-Nearest Neighbor, Desicion Tree and Random Forest. After that, the corresponding action can be taken according to the VNF status; a scale-up in case of overload, and a migration to the non-noisy server in case of noise/overprovisioning using our optimisation model.

B. Joined VNF placement and migration model

For our VNF placement and migration problem which is NP-Hard [2], we considered a virtualized data center that is composed of n servers, m VMs to be placed and V_j VMs to be migrated from server j that is in a noisy state. Our objective is to perform placement and migration jointly in order to minimize the noise in the infrastructure.

$$min(\sum_{i=1}^{m}\sum_{j=1}^{n} q_j x_{ij}) \tag{1}$$

Equations 2 and 3 indicate that in a given server l; the sum of the resource requirements (CPU and memory respectively) of the VMs placed in server l and the VMs migrated to server l, subtracting the resource requirements of VMs in server l which migrated to another server l' must be less than the server l's requirements capacity.

$$\begin{aligned}&\sum_{i=1}^{m} cpup_i x_{il} + \sum_{\substack{j=1\\ j\neq l}}^{n}\sum_{k=1}^{V_j} cpum_k z_{jlk} - \\ &\sum_{\substack{l'=1\\ l'\neq l}}^{n}\sum_{k'=1}^{V_l} cpum'_{k'} z_{ll'k'} \leq CPU_l, \forall l \in [1,n]\end{aligned} \tag{2}$$

$$\begin{aligned}&\sum_{i=1}^{m} memp_i x_{il} + \sum_{\substack{j=1\\ j\neq l}}^{n}\sum_{k=1}^{V_j} memm_k z_{jlk} - \\ &\sum_{\substack{l'=1\\ l'\neq l}}^{n}\sum_{k'=1}^{V_l} memm'_{k'} z_{ll'k'} \leq MEM_l, \forall l \in [1,n]\end{aligned} \tag{3}$$

Equation 4 ensures that each VM is placed in one server.

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i \in [1, m] \tag{4}$$

Equation 5 shows that a VM can only be migrated to one server

$$\sum_{\substack{l=1 \\ l \neq j}}^{n} z_{jlk} = 1, \forall j \in [1, n], \forall k \in [1, V_j] \tag{5}$$

Equation 6 ensures that VM_k is migrated to a server where there is no noise; this equation prevents migration to noisy servers.

$$(q_l + 1 - q_j) z_{jlk} = 0, \forall j \in [1, n], \forall l \in [1, n], \forall k \in [1, V_j], j \neq l \tag{6}$$

To sum up, the machine learning algorithms were used to identify the reason behind the performance degradation, whether it is due to overload or noisy neighbor problem. For the case of noisy problem, our ILP is invoked to propose a solution for migrating the impacted VNFs to non-noisy servers.

II. Performance Evaluation

In this section we present the experimental evaluation of our approach, for both machine learning and ILP models.

A. Machine Learning

To evaluate the machine learning algorithms, we used Scikit-learn python's library. The dataset contains 48112 instances and 4 features with categorical and equitable outputs. We chose to use K-Nearest Neighbor (KNN) since we only have four features and dataset with medium size. As illustrated in table I, KNN performs well with an accuracy equal to 99.8%. However, in case of real virtualized infrastructure with bigger volume of data, KNN risks to be slow in terms of execution time. In our experiment, the Decision Tree (DT) model generates a lot of false positives that degrade the model performance. One of the reasons creating this problem could be the overfitting which is much encountered problem when dealing with decision trees. To solve this problem, we used Random Forest (RF) classifier that is known by its robustness to overfitting and especially its efficiency in case of big sized data [3]. As illustrated in table I, the Random Forest corrected the problem of false positives and it proves its high performances in terms of accuracy, precision and recall with a low classification errors.

TABLE I: Evaluation metrics for each Model

	DT	RF	KNN
Accuracy	77.9	99.9	99.8
Classification Error	22.1	0.02	0.15
Recall	77.9	99.9	99.8
Specificity	65.9	99.9	99.8
False positive rate	22.3	0.02	0.2
precision	77.7	99.9	99.7

B. ILP

We adopted an existing simulator used in [2] to evaluate our ILP. When a request arrives, we run our model using IBM CPLEX [4] Optimizer engine to solve the proposed problem. The metrics used in our evaluation are **Rejection rate** which is the percentage of requests that have not been accepted and the **Convergence time** which represents the time needed to find a solution for one request.

The rejection rate measured in our experimental results increases according to different variations of lambda (the arrival rate) and noise rate (the percentage given to estimate the number of noisy servers in the infrastructure). The system reject exponentially as much as we increase the noise rate (table III). 98% requests are rejected while just 19% requests are rejected when we variate lambda parameter (as shown in table II) which means that our ILP is sensitive more to noise.

TABLE II: Rejection rate when varying the arrival rate λ

λ	5	10	15	20
Rejection rate (%)	8	13	18	19

TABLE III: Rejection rate when varying the number of noisy server

Percentage of noisy servers	0.2	0.4	0.6	0.8
Rejection rate (%)	4	10	52	98

To further extend the evaluation of scalability, we evaluate the convergence time. The table IV and table V show clearly that the convergence time depends strongly on the substrate graph sizes comparing with request graphs. The model spends more time to find solution in case of large infrastructure due to the np-hardness of the problem.

TABLE IV: Execution time when varying request sizes

Request size	5	20	50	100
Execution time (s)	17.067	24.997	27.056	28.448

TABLE V: Execution time when varying substrate graph sizes

Substrate graph size	100	500	1000
Execution time (s)	0.185	16.768	262.078

III. Conclusion

This paper deals with the noisy neighbor problem for network slicing NFV infrastructures. Results show an accuracy of 90% for noise detection and reveal some strengths of our ILP model. To the best of our knowledge, our approach is the first that address the noisy neighbor issue while identifying and solving the problem jointly. As future work, we are improving the proposed ILP to achieve better convergence time.

References

[1] Prometheus: "Monitoring system & time series database", https://prometheus.io/ [Accessed on: April 09, 2019].

[2] C. Ghribi, M. Hadji, and D. Zeghlache, "Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, May 2013, pp. 671–678.

[3] M. Zakariah, "Classification of large datasets using random forest algorithm in various applications: Survey," *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 4, pp. 189–198, 09 2014.

[4] IBM ILOG CPLEX Optimization Studio. [Online]. Available: https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/