KRS: Kubernetes Resource Scheduler for resilient NFV networks

Mohamed Rahali[1], Cao-Thanh Phan[1], and Gerardo Rubino[2]

[1]Firstname.Lastname@b-com.com, IRT B<>COM, Rennes, France
[2]Gerardo.Rubino@inria.fr, INRIA Rennes – Bretagne Atlantique, France

***Abstract*—To address the diversity of use cases envisioned by the 5G technology, it is critical that the design of the future network allows maximum flexibility and cost effectiveness. This requires that network functions should be designed in a modular fashion to enable fast deployment and scalability. This expected efficiency can be achieved with the cloud native paradigm where network functions can be deployed as containers.**

Virtualization tools such as Kubernetes [1] offer multiple functionalities for the automatic management of the deployed containers hosting the network functions. These tools such as resource scheduling and replicas must be applied efficiently to improve the network functions availability and resilience. This paper focuses on resource allocation in a Kubernetes infrastructure hosting different network services. The objective of the proposed solution is to avoid resource shortage in the cluster nodes while protecting the most critical functions. A statistical approach is followed for the modeling of the problem as well as for its resolution, given the random nature of the treated information.

I. Introduction

Future 5G and 6G netorks and beyond will be based on specific 5G evolutions to address new applications such as virtual reality, augmented reality, holography, intelligent connectivity, deep sensing, etc [2]. These evolutions include the support of multiple new access networks, for instance in airplane communications, satellite communications and underwater acoustic communications. These applications are very demanding in terms of end-to-end performance to meet the requirements of industries. Thus, the beyond 5G architecture should evolve to enable fast and efficient management of the network infrastructure [3].

One of the key enablers for this evolution is the support of the heterogeneous cloud environment and the ability to orchestrate resources among multiple technical domains like RAN, edge cloud, central/core cloud, private and public cloud. All the capability coming from the hardware acceleration should be exposed to the services to enable more flexible function placement in order to address high performance. Another key enabler will be the generalization of the service-based architecture (SBA) [4], which has been introduced in 5G networks mainly for the core network. This evolution will move to the edge and the RAN to enable the convergence of RAN and core networks. This allows a more flexible deployment of the network function and enhances their life cycle management.

The adoption of a cloud native [5] approach for the implementation and deployment of virtual network functions helps to achieve the required flexibility and autonomy. The use of containers enables to reduce the overhead and enhances the availability of the network services [6]. Future networks must be able to support multiple applications with different quality of service requirements. Eventually, the VNFs that compose these services will share the same infrastructure and will enter in competition for resources if the demand is too high. This situation may be more frequent in environments with limited resources such as the Edge or the RAN [7]. However, finding the right configuration in terms of resource allocation for the deployed functions while taking into account the temporal variability of the demands remains a major challenge to reduce resource shortage risk and protect the most important VNFs.

Resource management in cloud environments has been well studied in the literature [8]. However, most of the work already proposed study the virtual machines (VMs) as the basic and smallest entity to deploy and manage [9]. To the best of our knowledge, only a few papers study the particularity of containers and native cloud environments, especially the associated emerging technologies like Docker [10] and Kubernetes [1]. It is important to note that there are many similarities between the classic cloud environment and the cloud native one, especially in the resource management and allocation problems. However, the new technologies offer more features to improve the robustness of the deployed services. The Kubernetes technology represents an essential tool for the management and orchestration of cloud infrastructures. This technology has shown great industrial maturity and is becoming the key player in native cloud environments. It offers several features for automatic management, scalability and availability of services.

The scope of this paper is to explore some of the Kubernetes features for resource scheduling to enhance the availability and the resilience of NFV networks. We focus on specific ones dealing with resource scheduling to enhance the failure risk for the deployed network services. Then, we propose a statistical method to compute the corresponding parameters for each CNF taking into consideration the future demands and the importance of the hosted network function.

The remainder of the paper is organized as follows. Section II overviews the state-of-the-art and the main technologies for cloud resource allocation and management. Section III

describes the context of our contribution and the problem model formulation. Section IV presents our solution that we call the KRS module. Section V provides a description of the performance evaluation method and the obtained results. Finally, Section VI concludes the paper and outlines some perspectives for future developments.

II. Background and related work

In [11], the authors study the Kubernetes performances and propose a Petri Net model for pods and containers management. This model can help in the design of the applications and how the pods and containers should be deployed and structured. Authors in [12] propose a network-aware approach for Fog Computing service scheduling. The main objective of the proposed algorithm is reducing the network latency by optimizing the containers' placement. Meanwhile, the proposed solution do not consider the memory and CPU usage and their corresponding features offered by Kubernetes. In [13], the authors study the Kubernetes scheduling module and propose an improved one by combining two optimization methods: the Ant colony algorithm and the swarm optimization algorithm. The main difference of the proposed solution compared to the exiting scheduler is that they consider the infrastructure resource cost and not only the load-balancing aspects. In [14], the authors propose the KEIDS Kubernetes controller for Edge environments and IoT applications. The particularity of their solution is that it takes into account the energy consumption optimization while considering other aspects such as the interference between the containers. The problem is formulated and solved as a multi-objective ILP. Paper [15] also focuses on resource planning in cloud native environments with the objective of minimizing energy consumption. The task is formulated as a multi objective optimization problem with two main goals: energy consumption and computing time. Paper [16] studies the CPU usage interference between co-located containers. The results show that the performance degradation can reach 50% with intensive demands. The authors highlight the need for an efficient method of resource scheduling inside the cluster, to avoid the interference without specifying a specific solution. The scope of our paper is precisely to investigate the features offered by Kubernetes for this purpose.

III. General context and problem formulation

A. Kubernetes architecture

Cloud native and container technologies have significantly evolved, offering great opportunities for network operators to enhance their infrastructure management. The adoption of the cloud native approach in the implementation of VNFs enables building highly resilient and autonomous networks. Indeed, moving to cloud native network functions (CNFs) for NFV networks might make it easier to overcome multiple limitations of VNFs by transforming most network functions into scalable containers.

In this context, the Kubernetes ecosystem becomes an essential tool for the deployment and management of NFV infrastructures. The interest of this technology is mainly explained by its ability to manage dynamic, complex, and distributed infrastructures with great flexibility and agility.

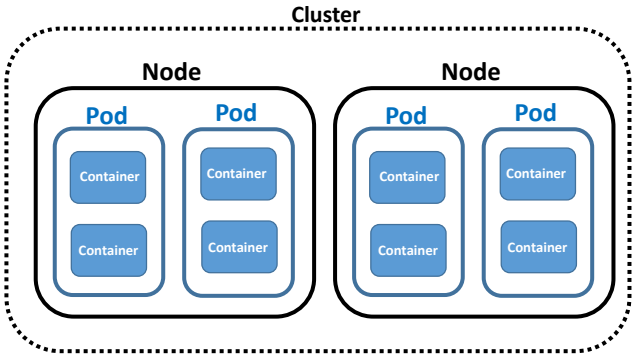

Fig. 1: Kubernetes cluster components

Fig. 1 illustrates the main components of the Kubernetes architecture which is based on multiple abstraction levels. The first one is the cluster which gathers multiple physical or virtual machines representing the available resources in terms of memory and CPU. Each cluster has a master node responsible for the management and scheduling of these resources. The cluster is composed of multiple nodes, that can be either physical or virtual. These nodes host the pods, the most basic entity that can be created and controlled by Kubernetes. Each pod runs a single service instance (a VNF instance in our case), and it can be composed of one or several containers.

B. Resource allocation and scheduling in Kubernetes

Kubernetes offers multiple tools for resource management shared by the deployed VNF instances. The network infrastructure can host heterogeneous services, allowing to take full advantage of the available resources and reduce consumption costs. On the other hand, sharing the infrastructure between heterogeneous services can reveal resource allocation problems, especially in case of a resource shortage. The allocation of cloud resources must be highly efficient. They must be neither under-utilized, as this means a loss of revenue, nor over-utilized, as this can lead to service failures or SLA degradation causing penalties.

Kubernetes offers some features for pods/containers resources scheduling which are mainly **requests** and **limits**. For each container, we can configure the requested memory or CPU which correspond to the guaranteed resources that will be reserved for it. Thus, the sum of the requested resources for the containers inside a node must be less than the total node capacity. Limits specify the maximum limit of resources that can be used by a container.

These parameters are very important for the CNF lifecycle management as they are taken into consideration in the priority and location of pods deployment by the scheduler and also in the allocation of resources inside a node, especially when it is overloaded. Let's take an example of a node that hosts pods and their memory consumption reaches its limit. The Kubernetes orchestrator has to select a pod to stop it. The

pods that do not have a fixed request value will be the first to be stopped. Then, the orchestrator selects those that consume the most compared to the value of the requests.

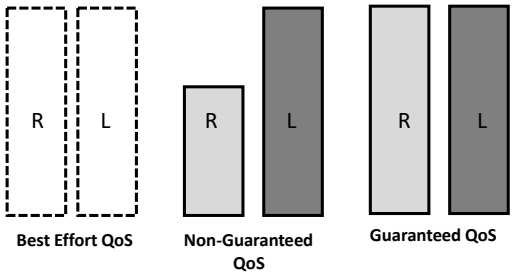

Fig. 2: Different CNF deployment configurations.

For instance, in Fig. 2 we represent three pods deployed on a single node. The first one is deployed with "best-effort" QoS, the requests and limits are not specified. The second is deployed with a "non-guaranteed" QoS, where the requests value is less than the limits. The third one has a "guaranteed" QoS since the requests value is equal to the limits. In this situation, if there is a lack of resources, the pods will be stopped in the given order, the first one, then the second one, and finally the third one, in order to liberate the required resources.

The objective of this work is to give a method to schedule the resources for the pods hosting the VNF instances, to limit the failures and the caused damage in the case of lack of resources. Our strategy aims to avoid the resource shortage in the first place. However, if it is unavoidable, the goal is to protect the most critical functions.

C. Problem formulation and notation

TABLE I: **List of main used variables**

variable	description
$\mathcal{N}$	set of nodes composing the cluster; $\mathcal{N}_i$: the i^{th} node
$\mathcal{V}_i$	list of pods inside $\mathcal{N}^i$
$\mathcal{V}_{i,j}$	pod j on node $\mathcal{N}^i$
$\mathcal{M}_i$	memory capacity of $\mathcal{N}^i$
$\mathcal{C}_i$	CPU capacity of $\mathcal{N}^i$
$\mathcal{M}^r_{i,j}$	request memory for pod $\mathcal{V}_{i,j}$
$\mathcal{M}^l_{i,j}$	limit memory for pod $\mathcal{V}_{i,j}$
$\mathcal{M}^d_{i,j}$	future demand memory for pod $\mathcal{V}_{i,j}$
$\mathcal{C}^r_{i,j}$	request CPU for pod $\mathcal{V}_{i,j}$
$\mathcal{C}^l_{i,j}$	limit CPU for pod $\mathcal{V}_{i,j}$
$\mathcal{C}^d_{i,j}$	future demand CPU for pod $\mathcal{V}_{i,j}$
$b_{i,j}$	variable representing the stopped pods, 1 if it is stopped and 0 otherwise
$\mathcal{P}_{i,j}$	penalty for stopping pod $\mathcal{V}_{i,j}$

Tab. I summarizes the notation adopted in this paper. We consider a Kubernetes cluster hosting multiple VNF instances. $\mathcal{N}$ denotes the set of nodes composing it and $\mathcal{N}_i$ refers to the ith node. Each node has a fixed memory and CPU capacities denoted by $\mathcal{M}_i$ and $\mathcal{C}_i$ respectively. The set of pods deployed on node $\mathcal{N}_i$ is denoted by $\mathcal{V}_i$. For each pod j on a node $\mathcal{N}_i$, the values of request and limits are denoted by $\mathcal{M}^r_{i,j}$ and $\mathcal{M}^l_{i,j}$ respectively for the memory, $\mathcal{C}^r_{i,j}$ and $\mathcal{C}^l_{i,j}$ for the CPU. The resources that will be consumed by each pod vary over time and depend on the VNF that hosts them. Therefore, the future resource demand for each pod is considered as a random variable denoted by $\mathcal{M}^d_{i,j}$ and $\mathcal{C}^d_{i,j}$ for the memory and CPU respectively. The VNF instances have to support the traffic load, otherwise they will be stopped and this can introduce penalties according to their importance. We denote by $\mathcal{P}_{i,j}$ the penalty introduced when stopping CNF $\mathcal{V}_{i,j}$. The objective of this work is to propose an algorithm that identifies the appropriate parameters of the requests and limits for each CNF, in order to reduce the expected penalty in a given period, where the profile of the future resource demand is known. This information is presented in the form of the probabilistic distribution of random variables $\mathcal{M}^d$ and $\mathcal{C}^d$ which can be obtained from the monitoring system.

When the demanded resources exceed the capacity of the node, the orchestrator selects the pod to stop according to the principle explained in Section III-B. Variable $b_{i,j}$ represents the state of the CNF. The selected ones should respect the condition described by (1):

$$\mathcal{M}^d_{i,j} - \mathcal{M}^r_{i,j} \leq \mathcal{M}^d_{i,j'} - \mathcal{M}^r_{i,j'}, \tag{1}$$

for $i = 1, 2, \ldots, N$ and for all j, j' such that $b_{i,j} = 1$ and $b_{i,j'} = 0$. The objective is to find the appropriate parameters $\mathcal{M}^r$, $\mathcal{M}^l$, $\mathcal{C}^r$ and $\mathcal{C}^l$ that minimize the expected penalties:

$$\text{minimize } \mathbb{E}\Big(\sum_{i=1}^{V}\sum_{j=1}^{V_i} b_{i,j}\mathcal{P}_{i,j}\Big)$$

subject to:

$$\sum_{j=1}^{V_i} \max(\mathcal{M}^r_{i,j}, \mathcal{M}^d_{i,j}) b_{i,j} \leq \mathcal{M}_i, \quad i = 1, \ldots, N,$$

$$\sum_{j=1}^{V_i} \max(\mathcal{C}^r_{i,j}, \mathcal{C}^d_{i,j}) b_{i,j} \leq \mathcal{C}_i, \qquad i = 1, \ldots, N.$$

IV. KRS MODULE PRESENTATION

This section presents the KRS algorithm for VNF resource scheduling. The randomness of the input data for the expected needed resource justifies the adoption of statistical approaches to model and solve this problem. The unknown variables are the requests and limits parameters for CPU and memory denoted by $\mathcal{C}^r$, $\mathcal{C}^l$, $\mathcal{M}^r$ and $\mathcal{M}^l$. The limits values, $\mathcal{C}^l$ and $\mathcal{M}^l$, can be easily determined for each CNF from its resource demand probabilistic distribution by choosing the maximum value. Meanwhile, finding the most suitable requests values is more challenging as they are the most important ones for resource allocation.

Our resolution method is a stochastic population-based algorithm. It is based on the random generation of multiple configurations of parameters $(\mathcal{C}^r, \mathcal{M}^r)$ and the selection of elements that do not exceed a penalty threshold. Then, these selected elements are crossed to generate a new improved population which allows reducing the expected penalties over

the iterations. To simplify the notation, we denote by $\mathcal{X}$ the vector of the requests values for both memory and CPU instead of the pair $(\mathcal{C}^r, \mathcal{M}^r)$. In the optimization process, $\mathcal{X}$ is considered as a discrete random variable. We consider an upper bound B of the unknown parameters which correspond to the maximum capacities of the nodes and we assume that the set of possible values of the parameters is $\{0, \Delta, 2\Delta, \ldots, J\Delta = B\}$ for a chosen unit Δ. The smaller the value of Δ the finer the granularity, and therefore the better the results. However, this will increase the computation time.

The probabilistic distribution of $\mathcal{X}$ is denoted by α, seen as an $2V \times J$ matrix. This distribution will be used to explore the solution space of the searched parameters. Indeed, α is initialized by a uniform distribution to explore equitably all the solution space. The general idea of the algorithm is to find the best distribution α that minimizes the expected penalty. For that, we will fix a maximum value of penalty denoted by P_{max} that we will try to minimize along the iterative process. The first step of the algorithm is to initialize P_{max} by a large value and α by an uniform distribution. Then, we will try to find a new distribution whose generated population denoted $\mathcal{X}^\alpha$ has an expected penalty lower than P_{max}. For this, we will generate a large number of vectors $\mathcal{X}^\alpha$ according to distribution α. Then, knowing the distribution of the future demand of resources $\mathcal{C}$ and $\mathcal{M}$, we compute the penalty expectation for each configuration x in $\mathcal{X}^\alpha$. Afterwards, the elements that have a value lower than P_{max} are selected to compute a new improved distribution.

To ensure the diversity of the new population, the percentage of selected elements must be greater than a minimum threshold noted D. Otherwise, the algorithm can converge quickly to a local optimum. The probability of observing value j on link ℓ, $\alpha(\ell, j)$, is computed by dividing the number of samples where link ℓ takes value $j\Delta$ by the total number of selected samples $|\mathcal{X}^{\text{selected}}|$ as described in (2).

$$\alpha(\ell, j) = \frac{|x \in \mathcal{X}^{selected}, x[\ell] = j\Delta|}{|\mathcal{X}^{selected}|}. \tag{2}$$

This process is repeated until the convergence of α or until reaching a maximum value of iterations.

The second step is to adjust P_{max} depending on the outcome of the first step. If α converges, P_{max} is reduced and the process restarts to try a more accurate target. If not, the procedure is repeated with a larger P_{max}. To reduce P_{max}, it is multiplied by an adaptation factor β, with values between 0 and 1. This factor is set at the initialization step before the execution of the procedure. The previous value of P_{max} is stored in a new variable P'_{max} at each adaptation. If α does not converge at the first step, P_{max} is reinitialized with P'_{max}. Afterwards, β is increased using (3):

$$\beta = \frac{1 + \beta}{2}. \tag{3}$$

The two steps are reiterated until P_{max} converges (i.e. until β becomes very close to 1). This process is globally described in Algorithm 1.

Note that our optimization approach is inspired by another algorithm called the Evolutionary Sampling Algorithm (ESA) [17] proposed for the inference of monitoring metrics. However, the principle remains adaptable to other optimization problems such as the one treated in this paper.

Algorithm 1 Kubernetes Resource Scheduler (KRS)

INPUT: $\mathcal{C}^d$, $\mathcal{M}^d$, $\mathcal{M}$, $\mathcal{C}$
OUTPUT: α

Initialize α (α^0), $\mathcal{P}^{max}$ and β
iter = 0 ▷ iteration index
— Step 1:
while iter $\leq$ itermax **do**
  iter = iter +1
  Generate n random samples of solutions according to distribution α.
  Select the ones that respect $\mathbb{E}_{penalty}(X) < P_{max}$; put them in set $\mathcal{X}^{\text{selected}}$.
  If $|\mathcal{X}^{\text{selected}}|/|\mathcal{X}^{\text{rand}}| < D$, repeat Step 1.
  Compute the new α (α^{iter}) from the selected samples:
    $\alpha^{\text{iter}}(\ell, j) = \frac{|x \in \mathcal{X}^{\text{selected}},\, x[\ell]=j|}{|\mathcal{X}^{\text{selected}}|}$
  if $\alpha^{iter} \approx \alpha^{iter-1}$ **then**
    Break ▷ α has converged
— Step 2:
if α has converged **then**
  $\mathcal{P}'_{max} = \mathcal{P}_{max}$
  α' = α
  $\mathcal{P}_{max}$ = $\mathcal{P}_{max}$ β
  iter = 0
  Repeat from Step 1
else
  β = $(1 + \beta)/2$
  $\mathcal{P}_{max} = \mathcal{P}'_{max}$
  if $\beta \approx 1$ **then**
    Return α ▷ $\mathcal{P}_{max}$ has converged
  else
    $\mathcal{P}_{max} = \mathcal{P}'_{max}\beta$
    iter = 0
    Repeat from Step 1

V. Evaluation

In this section we evaluate the performances of our KRS module compared to the Kubernetes best effort mode where the resource scheduling parameters are not configured on the CNF. For this evaluation, we use simulation as follows. At the beginning, for a given cluster with a known number of nodes, we generate randomly the memory and CPU capacity values for these nodes. Then, on each node, we vary the number of CNF deployed between 10 and 20. For each CNF, the future demand of resource is considered as a random variable $(\mathcal{C}^d, \mathcal{M}^d)$ following a Poisson distribution. The parameters of this distribution are randomly generated. The penalty for stopping the CNF is randomly generated and

takes normalized values between 0 and 1. Then, the KRS algorithm is performed to find the correct CNF configuration. The outcome of this algorithm will be the α distribution which often converges to an exact value. Finally, the penality expectation for the obtained configuration is calculated and compared to the one given by the Kubernetes best effort mode.

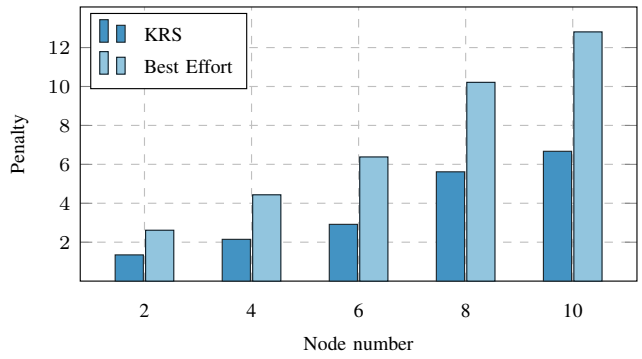

Fig. 3: Comparing the accumulated penalty for KRS and the best effort Kubernetes mode for different node number in the cluster

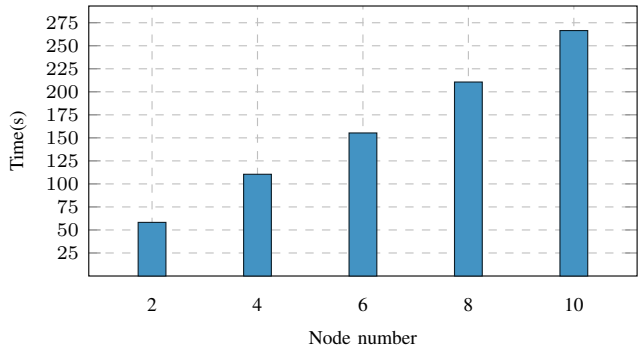

Fig. 4: Comparing the computing time for KRS and the best effort Kubernetes mode for different node number in the cluster

Fig. 3 illustrates the performance of the two approaches, namely the KRS module and the best effort mode. The number of nodes in the cluster varies between 2 and 10. For each configuration we make 50 different tests with different input data and then make the average of the results (the penalty and the computing time) . The tests show that our solution always performs better than best effort mode. For example, for tests done with 6 nodes in the cluster, the penalty expectation for the best effort mode is more than 6 while it is less than 3 with the KRS module. This result is expected since the best effort mode does not differentiate between the deployed functions and they are treated equally in the resource allocation. Concerning the computing complexity, Fig. 4 shows the required time to compute these parameters. We can see that it evolves linearly with the number of nodes. This is because the algorithm processes each node independently of the others, and it can lead to miss the optimal decisions globally, but it ensures the horizontal scalability of the solution. Meanwhile, regarding the vertical scalability of the algorithm, i.e. when we have a large number of CNFs on the same node, the computation time increases significantly. This remains a point to improve in our approach.

Another important parameter in this optimization process is the number of generated samples X^{rand} to explore the solution space. This variable belongs to the algorithm parameters that should be fine-tuned to get good results and avoid the convergence to local optima. Fig. 5 and Fig. 6 depicts the evolution of the accuracy and the computing time according to the generated samples X^{rand}. The results show that increasing the number of generated samples enhances the accuracy until reaching the best performance when the solution space is well explored. From this point on, increasing this value will only increase the computing time. Thus, a tradeoff must always be found to achieve the best performance with the least processing effort.

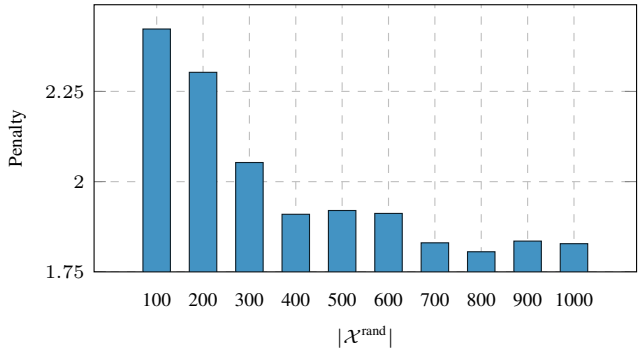

Fig. 5: Evaluating the KRS algorithm accuracy for different values of $|\mathcal{X}^{\text{rand}}|$

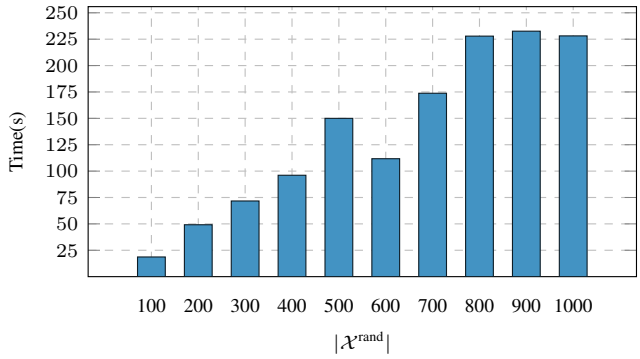

Fig. 6: Evaluating the KRS algorithm computing time for different values of $|\mathcal{X}^{\text{rand}}|$

VI. Conclusions

In this paper, we study the resource scheduling problem, mainly concerning memory and CPU, in a cloud native

environment. We particularly focus on the Kubernetes technology and its main features for resource scheduling. The proposed algorithm, called KRS, enables efficient provisioning for memory and CPU to avoid resource shortage, especially for the most critical CNFs. The KRS module considers the consumption profile of each CNF and allocates the appropriate resource with a trade-off between efficiency and risk. The optimization process is based on a statistical method that applies the principles of genetic algorithms. The results comparing our approach with the Kubernetes best effort mode show an interesting gain.

The utility of our algorithm appears in the post deployment phase of the CNFs to improve their coexistence in the same node and reduce the interference effects. However, Kuberentes offers various other parameters for the customization of the deployment phase. To optimize the overall resilience of network services, it is necessary to think about it in all stages of the service lifecycle from the design, the deployment, the operation phase, and finally, the shutdown. All these steps require specific supplementary in-depth studies.

ACKNOWLEDGEMENTS

This work has been partially supported by the European Union's H2020 MonB5G (grant no. 871780) project.

REFERENCES

[1] E. A. Brewer, "Kubernetes and the path to cloud native," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 167. [Online]. Available: https://doi.org/10.1145/2806777.2809955

[2] K. David and H. Berndt, "6g vision and requirements: Is there any need for beyond 5g?" *IEEE Vehicular Technology Magazine*, vol. 13, no. 3, pp. 72–80, 2018.

[3] A. Boudi, M. Bagaa, P. Pöyhönen, T. Taleb, and H. Flinck, "Ai-based resource management in beyond 5g cloud native environment," *IEEE Network*, vol. 35, no. 2, pp. 128–135, 2021.

[4] G. Mayer, "Restful apis for the 5g service based architecture," *J. ICT Stand.*, vol. 6, no. 1-2, pp. 101–116, 2018. [Online]. Available: https://doi.org/10.13052/jicts2245-800x.617

[5] S. Sharma, R. Miller, and A. Francini, "A cloud-native approach to 5g network slicing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 120–127, 2017.

[6] T. Taleb, A. Ksentini, and B. Sericola, "On service resilience in cloud-native 5g mobile systems," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 483–496, 2016.

[7] P. L. Vo, M. N. H. Nguyen, T. A. Le, and N. H. Tran, "Slicing the edge: Resource allocation for ran network slicing," *IEEE Wireless Communications Letters*, vol. 7, no. 6, pp. 970–973, 2018.

[8] R. Weingärtner, G. B. Bräscher, and C. B. Westphall, "Cloud resource management: A survey on forecasting and profiling models," *Journal of Network and Computer Applications*, vol. 47, pp. 99–106, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804514002252

[9] C. Canali and R. Lancellotti, "Scalable and automatic virtual machines placement based on behavioral similarities," *Computing*, vol. 99, no. 6, pp. 575–595, 2017. [Online]. Available: https://doi.org/10.1007/s00607-016-0498-5

[10] C. Anderson, "Docker [software engineering]," *IEEE Software*, vol. 32, no. 3, pp. 102–c3, 2015.

[11] V. Medel, O. Rana, J. a. Bañares, and U. Arronategui, "Modelling performance & resource management in kubernetes," in *Proceedings of the 9th International Conference on Utility and Cloud Computing*, ser. UCC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 257–262. [Online]. Available: https://doi.org/10.1145/2996890.3007869

[12] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 351–359.

[13] W. Zhang, X. Ma, and J. Zhang, "Research on kubernetes' resource scheduling scheme," in *Proceedings of the 8th International Conference on Communication and Network Security, ICCNS 2018, Qingdao, China, November 02-04, 2018*. ACM, 2018, pp. 144–148. [Online]. Available: https://doi.org/10.1145/3290480.3290507

[14] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "KEIDS: kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4228–4237, 2020. [Online]. Available: https://doi.org/10.1109/JIOT.2019.2939534

[15] M. Adhikari and S. N. Srirama, "Multi-objective accelerated particle swarm optimization with a container-based scheduling for internet-of-things in cloud environment," *J. Netw. Comput. Appl.*, vol. 137, pp. 35–61, 2019. [Online]. Available: https://doi.org/10.1016/j.jnca.2019.04.003

[16] E. Kim, K. Lee, and C. Yoo, "On the resource management of kubernetes," in *2021 International Conference on Information Networking (ICOIN)*, 2021, pp. 154–158.

[17] M. Rahali, J. Sanner, and G. Rubino, "Unicast inference of additive metrics in general network topologies," in *27th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2019, Rennes, France, October 21-25, 2019*. IEEE Computer Society, 2019, pp. 107–115.