Robust Multi-Resource Allocation Against Controller Failures in Network Slice Provisioning

Laiza de Lara, Francesca Fossati, Stephane Rovedakis

Robust Multi-Resource Allocation Against Controller Failures in Network Slice Provisioning

Laiza de Lara, Francesca Fossati, Stéphane Rovedakis

***Abstract*—We present solutions for reliable mulit-resource management for network infrastructures. We propose enhancements to four distributed multi-resource allocation policies for a network slice provisioning for robustness against resource controllers failures. We define two cascading and two parallel computation approaches; their comparison, using a simulation approach, show that the parallel ones perform better in terms of fault resiliency and provisioning latency, while a cascading approach using a controller pre-ordering policy is able to better reallocature resources by recomputation upon failures.**

***Index Terms*—5G orchestration, multi-resource allocation, fault-resiliency.**

I. Introduction

The multi-domain integration of infrastructure resources beyond the radio access is a key challenge in the migration to the fully-fledged standalone 5G [1]. In fact in the standalone 5G, the integration of 5G core network functions, their distribution and agile orchestration, and end-to-end slicing, are factors that challenge the conventional logic of network resource provisioning [2]. In particular with network slicing, multiple resources of multiple domains, as for instance RAN, transport and core domains, are composed for granting an end-to-end service-level-agreement [3].

Therefore, the conventional single-resource allocation paradigm is challenged by the multi-domain nature of network slicing. Under a multi-resource allocation view, dependency among resources of a network slice is to be considered; the operator should not allocate the full capacity to a link bearer associated to an eMBB (enhanced Mobile BroadBand) slice user, for instance, when the user cannot get the corresponding necessary bandwidth in the radio spectrum access [4]. Given the much higher key-performance-indicators scales in 5G with respect to 4G, and the congestion at the radio access or computing or backhauling link levels (depending on the type of service that has to be provided), resources should not be wasted, and their allocation shall instead be fine-tuned in an agile fashion. Indeed, related resources they depend upon may not be available due to resource congestion situations along a slice chain, among users within a slice, or among slices.

Taking into account multi-resource dependencies is an important challenge for the 5G slice life-cycle management, for both the preparation and commissioning phases as per ETSI specifications [5]. In the literature [6], [7], resource allocation is often associated as a specific subproblem within a broader definition of the slice embedding problem (decomposed in resource discovery, virtual network mapping, and resource allocation steps). Accordingly, software stacks, as Open-Radio Access Network (ORAN) and Open Network Automation Platform (ONAP), are expected to expose inter-domain and inter-resource controller interfaces to support resource management such as the one presented in this article.

Multi-resource allocation is a well-known problem in cloud computing; orchestrators allocate computing-power-unit (CPU), live memory, storage, accelerators, while handling multi-resource constraints [8]. Cloud systems recently even more penetrate the network stack, with cloud-native and server-less fabrics being explored for function deployment [9]. Having such a centralized orchestration for cross-domain network slice resource management would, however, imply a strong collaboration among resource providers, likely not possible to achieve. Moreover, the distributed and long-distance nature of the end-to-end slice spanning radio, transport, and multiple computing domains (NFV, edge Cloud, remote Cloud) also pushes toward the adoption of distributed approaches [10]. The distribution is also needed to ensure fault-resiliency and isolation among resource domains, so that if one provider fails the other can continue operating a service in a best-effort way. Fairness and efficiency trade-offs have to be considered when trading a level of allocation with the related system efficiency [11].

In this article, we compare distributed multi-resource allocation policies, in terms of fault resiliency and efficiency when reallocating resources upon resource domain failure.

II. Background

Network resource allocation can be addressed as a single-resource problem, when only one resource has to be shared among the users, or as a multi-resource one, when there are at least two resources to share and at least one of them is under congestion [12]. Given a total amount of available resource to be shared and a resource demand associated to each user, an *allocation* defines the portion of resource demand given to every user, satisfying desirable mathematical properties [10].

Classically, allocation problems are formulated as a convex optimization aimed to maximize an aggregate utility. The problem is not trivial when the resource is *congested*, i.e., when the sum of all the demands of the users is higher than the available resource. In this case the *congestion level*, defined as the ratio between the sum of the demands and the available amount for the given resource has value greater than one.

Laiza De Lara and Stéphane Rovedakis are with Cnam, Cedric, 75003 Paris, France. Email: {firstname.lastname}@cnam.fr

Francesca Fossati is with Univ. Paris Saclay, Centrale-Supelec, France. francesca.fossati@l2S.centralesupelec.fr

This work was funded by the ANR-18-CE25-0012 project.

When multiple resources are combined for provisioning a slice, multi-resource approaches shall be used to avoid resource wastage in case of congestion. In literature, we can find two approaches: a centralized one where a multi-domain orchestrator has a view on all the resources, receives the users demands and calculates the allocation, or a distributed one where providers calculate their own allocation without sharing confidential information and the final allocation is calculated thanks to a collaboration between providers [10].

In multi-resource allocation, several resources have to be shared among a set of users (or tenants). Each user can obtain a portion of every resource to get its service running. Therefore, for each user we have to take into account a set of single-resource demands, one per resource. Given the available amounts for the resources to be shared and the batch of demands, a *multi-resource allocation policy* defines for each user the allocated amount given for every resource. As in [11], [13] one can assume a *linear relationship* between the resources, e.g., if a user can only obtain half of its demands for a resource, then a directly proportional reduction does apply for all the other resources. Thus, the amount of the resources allocated to each user in a multi-resource setting can be stated as a unique allocation rate of its demands for all the resources.

A. Centralized multi-resource allocation policies

In the centralized case, the *Dominant Resource Fairness (DRF)* policy is a well-known policy [8]: it generalizes the MMF one for a multi-resource context; for every user, it computes the share of each resource allocated to that user and the allocation of a user should be determined by the user's dominant share (the maximum share that the user has been allocated on any resource). Different users may have different dominant resources; in a nutshell, DRF seeks to maximize the minimum dominant share across all users.

Other centralized multi-resource allocation policies can be considered; a description of the state of the art is given in [10].

B. Distributed resource allocation policies

In standards slice life-cycle management, multi-domain resource allocation operations can be precisely framed within the commissioning phase, which includes the creation of the network slice instance within which all needed resources are allocated and configured to satisfy the network slice requirements. In the literature, distributed resource allocation is often not explicitly identified as a distinct step within slice/network embedding or function placement problems. For instance, [14] proposes a distributed algorithm based on a multi-agent approach to compute the mapping of virtual nodes and links on the physical network managed by a single provider. [15] considers the network embedding problem over multiple infrastructure providers and provides a distributed protocol for coordinating the computation of the embedding across the participating providers. These works address the mapping problem of virtual networks on a physical infrastructure, but they do not explicitly consider multi-resource dependencies nor maximum resource allocation budgets in the resource allocation at the function or virtual network granularity; this would be indeed required to integrate the multi-resource allocation policies we consider in this paper.

Our previous work [10] proposes distributed multi-resource allocation policies for slicing. In this context, following a distributed setting, several resource controllers, as for instance radio intelligent controllers (RIC), software-defined-network (SDN) controllers and cloud stack and network-function-virtualization (NFV) orchestrators, collaborate in order to compute the ratio of each resource to allocate to every user. Two directions are proposed: one follows a cascading approach, while the other exploits parallelism. In cascading approaches, each provider sends information about its allocation rates to the following one, as so on so forth across all the providers resource controllers, the allocation rates are adjusted taking into account the congestion level of each resource. The provider order can be pre-established or can depend on the provider congestion level. In parallel policies, after the diffusion of the allocation rates between the providers, the allocation is calculated using a consensus algorithm. The policies presented in [10] do not take into account the occurrence of failures and the impact on the slice resource allocation management. In this article, we fill this gap by presenting enhanced versions of previous policies that are robust against failures.

III. Enhanced multi-resource allocation policies under controller failures

In practice, faults can happen during the distributed multi-resource computation process, for example due to a software failure, so that a loss of signaling messages may occur, at different providers. In this section, we describe the enhanced version of the algorithms proposed in [10] to deal with failures.

A. Problem statement

The general problem we address can be formalized as follows. Let $N = \{1, ..., n\}$ be the set of tenants, $M = \{1, ..., m\}$ be the set of available resources and $P = \{1, ..., p\}$ be the set of resource providers. We consider $|P| \leq |M|$ and only one provider for each resource, so that each provider can provide more than one resource. The allocation problem is represented as a pair (D, R), where D is a $n \times m$ matrix with d_{ij} equal to the quantity of resource $j \in M$ demanded by tenant $i \in N$ and $R = (r_1, ..., r_m)$ is a vector of positive numbers r_j equal to the amount of each available resource $j \in M$. It is important to note that each provider k only knows the quantity of available resources r_j and the submatrix D_k of demands for the resources $j \in M$ it manages.

The allocation, i.e., the outcome of the problem, is represented by a matrix A with components $a_{ij} = d_{ij} \cdot x_i$ where $x = (x_1, ..., x_n)$, $0 \leq x_i \leq 1 \ \forall i \in N$, is the vector of the percentage of demand allocated to each tenant. The allocation is challenging if it exists a resource $j \in M$ such that $\sum_{i=1}^{n} d_{ij} > r_j$, i.e., if the resource is not covering the demands of the users. A solution A is admissible if it is Pareto efficient, i.e., if for any other allocation A' we have $a'_{kj} > a_{kj}$ for some $k \in N$ and $j \in M$ implies $a'_{k'j} < a_{k'j}$ for some $k' \in N, k \neq k'$.

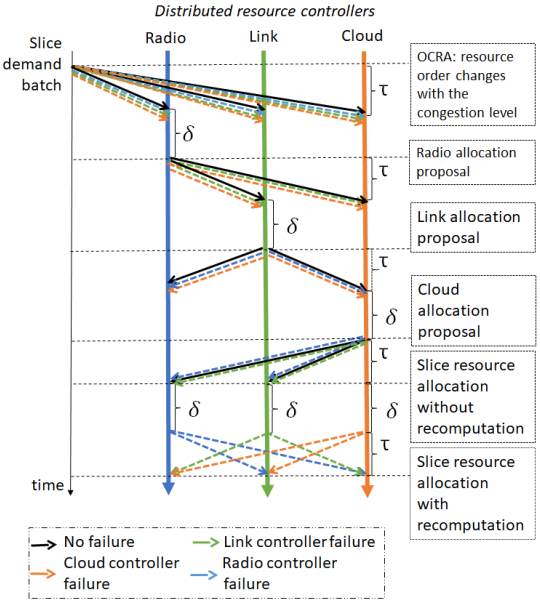

Fig. 1: CRA and OCRA policy chronogram. δ is the computing time and τ is the transfer time.

We assume that each resource provider is prone to a failure and is able to automatically recover from a failure after a given amount of time. We consider two types of failure: (i) the controller failure, i.e., a failure during the actual computation at controller and (ii) the loss of messages during the communication of the rates between two controllers. Only one failure at a time is considered in our analysis because we take into account failures at the resource controller system level, assuming each controller is in a different provider domain and failures are not prone to be propagated across providers, as the slice provisioning has to guarantee high availability.

We refer to *slice resource allocation without recomputation* to the solution obtained when a provider is down, and to *slice resource allocation with recomputation* to the solution obtained once the failed provider has recovered, after a failure recovery procedure, because a provider takes into account the information exchanged by the other providers in order to complete the computation of the allocation rates. The possible consequence of a provider failure, in the case without recomputation, is the loss of its admissible solutions, which cannot be computed by the other providers since they do not know the amount of available resource associated to the failed provider. The impact of a failure can be more or less disruptive, depending on the allocation policy and the congestion level of the failed provider.

B. Cascade allocation policies

We present the steps of the Cascade Resource Allocation (CRA) policy and formal description is given in Algorithm 1 (for normal operation, one provider failure or message loss):

- All the providers process a batch of slice demands, composed of a requested amount for each resource, for each demand. Except the first in the chain, the others wait the allocation rate computed by the preceding provider.
- The first provider in the chain calculates the allocation rate x for its resources[1] and sends the allocation vector to the next provider in the chain.
- For the following providers, if the preceding provider's allocation rates are received within a given timeout t_p[2], the provider check the admissibility of the proposed allocation and in case it can not be accepted it proposes a new allocation, computed based on its local information and the allocation rate from the preceding providers. In case of failure, after the time-out has expired the next provider in the chain after the failed provider automatically takes over the cascading allocation process.
- In case of failures causing disconnection of a resource controller, when the failed provider recovers, it receives the allocation rates from the other controllers and checks if it can allocate the corresponding allocation rate respecting multi-resource constraints, considering the initial received demands and the amount of provided resource. If it is not possible, i.e., we have an allocation we refer to as not *admissible*, the provider computes new allocation rates by considering as upper bounds the ratios computed by the other providers.

Algorithm 1 CRA

Input: $R, D, N, M, P, t, timeout$
Output: x

Each provider $k \in P$ receives D_k, i.e., the submatrix of demands for the resources provided by k
for $k = 1 : p$ **do**
  if $timeout$ of k is not expired **then**
    if $\sum_{i=1}^{n} d_{ij}x_i > r_j$, for at least one j provided by k **then**
      Provider k updates x (old values of x_i are upper bound)
    else
      Provider k accepts x
    end if
  else
    Provider k calculates x eventually considering the received proposal of the previous not failed providers
  end if
  Provider k sends x to the other providers
end for
if one provider k was down **then**
  if $\sum_{i=1}^{n} d_{ij}x_i > r_j$, for at least one j provided by k **then**
    Provider k updates x (old values of x_i are upper bound)
  else
    Provider k accepts x
  end if
  Provider k sends x to the other providers
end if

Figure 1 illustrates the CRA policy steps when we consider three resource providers. For this first algorithm, the provider order is meant to be pre-established; e.g., first radio, then link and cloud providers. When a failure occurs at the last provider in the chain (e.g., the cloud provider in Figure 1) CRA sequence to compute the allocation rates after recovery is not changed, so the allocation rate provided by the failed provider does not change in comparison with the normal system. However, if the failure occurs ahead, e.g., at the radio or link provider, then the computation path is modified and the allocation rates computed by the failed provider are different.

[1]Each provider decides which allocation rule to adopt (e.g., MMF, proportional, DRF, ...) to allocate its own resources.

[2]We can define a timeout vector t containing the timeout of each provider.

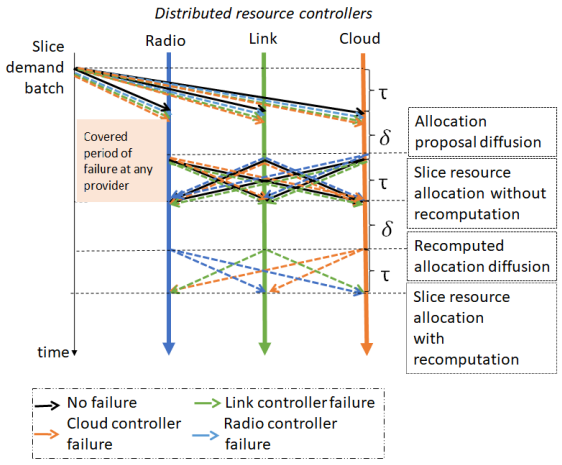

Fig. 2: PRA-1 chronogram. δ is the computing time and τ is the transfer time.

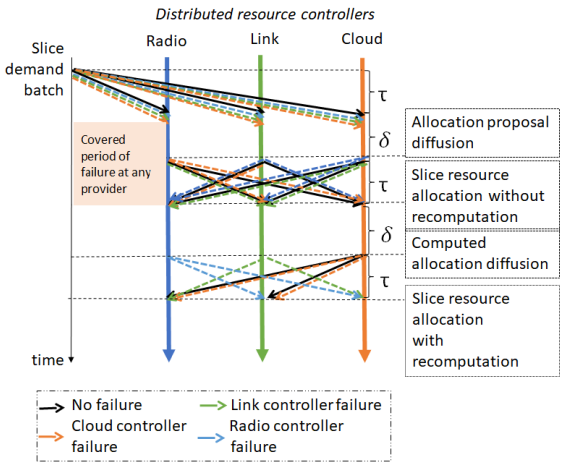

Fig. 3: PRA-2 algorithm chronogram. δ is the computing time and τ is the transfer time.

The second policy we present is the Ordered Cascade Resource Algorithm (OCRA). It works similarly to CRA but, unlike CRA, the pathway between the nodes is not pre-defined. OCRA holds a *multi-domain orchestrator* able to re-order the resource provider pathway considering the congestion levels, to try to (partially) avoid resource re-allocation. In this setting, two more steps are added before the first step of the CRA: (i) each resource provider computes its congestion level (maximum of the congestion levels if many resources) and sends it to the multi-domain orchestrator, (ii) the orchestrator orders the resources from the most to the least congested ones, and it sends the order to the resource providers.

Most of the elements described for CRA are similar except the order considered in the computation path, and the additional delay that could be induced by the ordering process. Similarly to the CRA case, if the failed provider is the least congested, the original (pre-fault) order is not changed and the system does not have to recompute when this provider gets up.

C. Parallel allocation policies

In the previous policies, the computation of the resource allocation is done following a weakly distributed manner. Indeed, the resource allocation is computed according to a defined sequence among the resource providers, which implies a high dependency between providers. Instead, as illustrated in Figure 2 and 3, the allocations calculated by the providers are broadcasted and used to obtain the final allocation thanks to a consensus algorithm. Actually, the parallel approach can be declined in two different policies named PRA-1 and PRA-2.

In PRA-1, each provider simply broadcasts to all the other providers its computed allocation rates, and the final rates are obtained immediately after, in the consensus phase, by taking the minimum one among those computed by the providers for each user; even if faster, this does not guarantee a Pareto-efficient solution [10] and satisfying Pareto efficiency grants strategic stability and tenant satisfaction for the long-run.

In case a provider fails, the consensus algorithm defines the allocation rate based on the available providers. When the failed provider recovers, it receives the allocation rate from the system and if not satisfied, the provider performs the allocation of its resources using the rule that suits it; this ends up being the final allocation of the system, without the need for the consensus algorithm to validate it.

PRA-2 introduces an additional round in order to compute a Pareto-efficient solution and the time to compute the allocation is increased in normal operation. Every provider broadcasts to the other providers: (i) the computed allocation rates, (ii) its congestion level, i.e., the maximum ratio between the requested resources and the available resources, and (iii) the resource share for each resource it provides for each user (i.e., ratio between the demand of a user and the available resource).

In the consensus phase, the provider with the most congested resource can identify itself and computes the final allocation rates using a multi-resource rule. Note that CRA, OCRA and PRA-2 provide Pareto-efficient solutions [10].

For PRA-2, after the information exchanged among providers in the first phase the most congested provider is responsible to compute the final multi-resource allocation. In case of failure of one provider we set as hypothesis for the parallel case that the recover is just after the messages broadcast. This means that for the PRA-2 we do not have to consider slice resource allocation without recomputation. Therefore, we do not need extra time to provide the solution because the recovered provider can take the decision. It follows also that the allocation provided is always admissible.

Algorithm 2 and 3 formalize the two parallel algorithms for p providers in case of normal operation or one failure.

To conclude the section we mention that both the approaches have advantages and disadvantages. In particular: cascading approaches have high delay budget (total time to compute the allocation) but low message complexity, while parallel ones have low delay budget but high message complexity. A more detailed analysis is given in [10].

IV. Fault-tolerance evaluation

For the policy evaluation, we simulate a setting where providers are prone to failures during the computation of the multi-resource allocation.

We simulate a system composed of three providers (radio, link and cloud) managing the allocation of four different

Algorithm 2 PRA-1

Input: $R, D, N, M, P, timeout$
Output: x
Each provider $k \in P$ receives D_k, i.e., the submatrix of demands for the resources provided by k
Each provider $k \in P$ calculates the value of x
Each provider sends the allocation vector to each of the other providers
When the $timeout$ is expired a consensus algorithm provides the final value of x considering the received allocations
if one provider k was down **then**
  if $\sum_{i=1}^{n} d_{ij}x_i > r_j$, for at least one j provided by k **then**
    Provider k updates x (old values of x_i are upper bound)
  else
    Provider k accepts x
  end if
  Provider k sends x to the other providers
end if

Algorithm 3 PRA-2

Input: R, D, N, M, P
Output: x
Each provider $k \in P$ receives D_k, i.e., the submatrix of demands for the resources provided by k
Each provider $k \in P$ calculates the value of x
Each provider sends a batch of information to each of the other providers
A consensus algorithm provides the final value of x.

resources: resource blocks for the radio provider, bitrate for the link provider, and vCPU and memory for the cloud provider.

For the slice demands, we used demands generated from Amazon EC2 templates as in [10] completed with a number of Resource Blocks (RB) corresponding to the associated throughput[3]. In particular when the link resource is 10 Gbps the RB number obtained is 135 and when the link resource is 25 Gbps the RB number is 273.

In our simulations, we generated 200 problems having five tenants, defining a total of 1000 demands. For each problem, we selected random demands based on the Amazon instance types (considering the associated RB amount) in order to obtain a congestion level between 0.6 and 1.6, defined randomly for every provider. Note that we did not correlate the instance type with the failure occurrence[4]. We consider the proportional allocation rule for the radio and link resource providers and the DRF rule for the cloud provider.

As the providers are prone to a failure, we have considered the case with (i) normal operation (no failure during the computation), and the cases when a failure occurs at the (ii) radio, (iii) link or (iv) cloud provider.

A. Numerical results

We evaluate the robustness of the proposed policies for the failure scenarios. We consider the *number of non-admissible solutions* (i.e., situations in which a slice cannot be allocated) when a failure occurs in the system during the computation of the allocation. Moreover, from the tenants' point of view, we analyze the *impact of a failure on the allocation* We compare the resource allocation rates in a normal operation

[3]we used state of the art 5G radio dimensioning tools: https://5g-tools.com/5g-nr-throughput-calculator/

[4]we have 196 problems with at least one congested provider and each provider is congested in more than half of the cases: 125 times for the radio, 110 times for the link, and 170 times for the cloud ones.

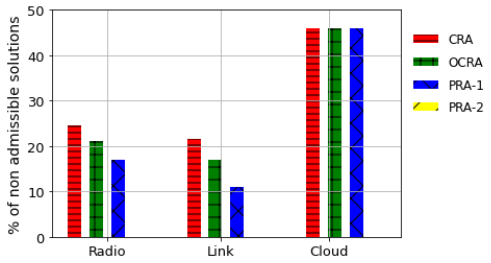

Fig. 4: Non-admissible solutions after the failure of a provider.

by the allocation rates computed: (i) when we have missing information from a failed provider in the 3 scenarios (i.e., without recomputation) and (ii) by the failed provider that recovered in the 3 scenarios (i.e., with recomputation).

1) Solutions admissibility: In this section, we analyze the percentage of solutions (computed by the providers after a failure) which are admissible, i.e., for every resource the sum of the users allocations must not be higher than the available resource. Non-admissible allocations lead to the recomputation of new allocation rates by considering as upper bounds the ratios computed by the failed system, so higher non-admissible allocation rates lead to higher overhead in the multi-resource allocation process, namely in terms of provisioning delay.

Figure 4 presents for every algorithm and each scenario the percentage of non-admissible solutions obtained in our simulations. Parallel approaches have better performance compared to the cascade ones. In fact, as previously explained PRA-2 do not provide non-admissible solution because it is composed of two communication phases, and the failed provider recovers before the second phase so that it can compute an admissible solution. Furthermore, PRA-1 achieves low non-admissible solutions rates compared to cascade approaches.

In the case of cloud failure, all the policies (except for PRA-2) achieve higher and similar percentages of non-admissible solutions with 46% for CRA, OCRA and PRA-1. Furthermore, the percentage of non-admissible solutions is smaller for a radio and a link failure in comparison to a cloud failure. This is a direct consequence of a higher number of problems in which the cloud provider is the most congested, and it is also accentuated by the fact that this provider manages two resources. The same comparison can be made between a radio and a link failure. Therefore, we can conclude that the more congested is a provider suffering from a failure, the highest is the impact on the computed allocation. Furthermore, PRA-1 achieves low non-admissible solutions compared to cascade approaches in case of a radio and a link failure, nonetheless it is important to notice that PRA-1 does not lead to a Pareto efficient solution contrary to CRA, OCRA and PRA-2 [10].

2) Allocation upon failure: In this section we consider the robustness of CRA, OCRA and PRA-1 for the three failure scenarios (PRA-2 is excluded since there is no missing information because of a provider failure). We analyze how far is the allocation computed by the system after a failure for each provider from the allocation obtained in normal operations (without failure). The failure of a provider implies missing information on the amount of available resource managed by the failed provider and the users' demands related to the missed resource(s).

	Type of failure	CRA		OCRA		PRA-1		PRA-2	
		Increase	Decrease	Increase	Decrease	Increase	Decrease	Increase	Decrease
Without recomputation	Radio	26%	11%	20%	6%	26%	0%	-	-
	Link	24%	11%	18%	8%	22%	0%	-	-
	Cloud	20%	0%	25%	17%	40%	0%	-	-
With recomputation	Radio	16%	11%	10%	7%	16%	0%	-	-
	Link	18%	11%	10%	7%	20%	0%	-	-
	Cloud	0%	0%	18%	0%	30%	0%	-	-

TABLE I: Allocation rates changes without or with recomputation upon failure with respect to the case without failure.

Table I presents, the difference (in percentage) between the allocation rates obtained after each provider failure and the allocation computed in a normal operation. When upon a provider failure, the allocation rates increase, we can obtain an *invalid allocation* since the system is trying to allocate more resources than in the normal operation. On the other hand, if the system decreases the allocation rate of a user, the allocation is valid but may be less efficient. In fact, an increase can generate invalid allocations if the allocation rates is not reduced by the same proportion for other users. Invalid allocations can also lead to a non-admissible solution because of over-allocations for several users. Therefore, to determine the most fault-tolerant algorithm, we evaluate the percentage of increase and decrease in the rates upon a failure.

Considering the two first lines of Table I, we see that again parallel algorithms have better performances. PRA-1 suffers the less from the two types of failure. Furthermore it can only have an increase of the allocation rates, since the decision is taken as the minimum allocation between the proposed ones.

Contrarily, we can observe that after the failure of the cloud provider, the least impacted algorithm is CRA with a modification of around 20%: only an increase is noticed due to the fact that the failed node is exactly in the last position on the computation path considered by this algorithm. PRA-1 and OCRA show almost the same higher percentage change for the allocation rates.

In general, we can notice that the failure of the highest congested provider impacts a high number of allocations.

3) Allocation after recomputation: The lines 4 to 6 of Table I present the difference between the recomputed allocation rates for every failure scenario and the allocation rates computed in a normal operation.

Among the three providers, the most differentiating results are obtained for the failure of the cloud provider. The algorithm which provides the best results is CRA, which is able to recover all the allocation rates and we reach the same allocation compared to a normal operation (recovery of 100%). It is due to the fact that the computation path is not modified compared to a normal operation. OCRA and PRA-1 recover almost the same percentage of invalid allocations. For the radio and link failure, we can observe similar results for the three algorithms. The increase of the allocation rate is reduced by 10 to 25% for CRA and PRA-1 algorithms, while a reduction of 50% is achieved by OCRA algorithm. So, OCRA algorithm allows to obtain the lower amount of invalid allocations after recomputation. Moreover, the three algorithms are able to maintain the same less efficient allocation.

V. Conclusion

We proposed in this paper a set of enhanced version of distributed multi-resource allocation algorithms that are resilient against resource controller failures. Extensive simulations of realistic settings show at which extent parallelization of the multi-resource allocation computation at different controllers is the best approach in terms of fault resiliency, while ordering cascading is able to compensate the most by recomputation upon failure. PRA-2 allows to reach the best compromise between fault resiliency and performance. It guarantees admissible solutions which are Pareto-efficient, while introducing no additional computation delay in case of a failure in comparison to a normal operation. Thus, this is one of the best distributed multi-resource allocation algorithm to use when the system is prone to failures, knowing that it suffer from high message complexity.

A challenge open for future work relates to the integration of the proposed framework in disaggregated Open RAN platforms and related integration with network and service function orchestration platforms.

References

[1] J. Yeung, "An overview of 5g network architecture," 2020. [Online]. Available: https://medium.com/analytics-vidhya/an-overview-of-5g-network-architecture-d82379862707

[2] G. Brown, "Service-based architecture for 5g core networks," 2017. [Online]. Available: https://www.3g4g.co.uk/5G/5Gtech_6004_2017_11_Service-Based-Architecture-for-5G-Core-Networks_HR_Huawei.pdf

[3] K. Katsalis *et al.*, "Network slices toward 5g communications: Slicing the lte network," in *IEEE Communications Magazine*, vol. 55, no. 8, 2017, p. 146154.

[4] X. Foukas *et al.*, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[5] ETSI TS 128 530 V15.1.0 (2019-04), "5g; management and orchestration; concepts, use cases and requirements," 3GPP TS 28.530 version 15.1.0.

[6] F. Esposito, I. Matta, and V. Ishakian, "Slice embedding solutions for distributed service architectures," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 6:1–6:29, 2013.

[7] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[8] A. Ghodsi *et al.*, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proceedings of NSDI 2011*.

[9] Huawei Technologies, "5g network architecture: A high-level perspective," pp. 1–8, 2020.

[10] F. Fossati, S. Rovedakis, and S. Secci, "Distributed algorithms for multi-resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2022.

[11] M. Leconte *et al.*, "A resource allocation framework for network slicing," in *IEEE INFOCOM*, 2018.

[12] P. Poullie *et al.*, "A survey of the state-of-the-art in fair multi-resource allocations for data centers," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 169–183, 2018.

[13] S. Lee *et al.*, "Resource management in service chaining," draft-irtf-nfvrg-resource-management-service-chain-03, IETF draft, 2013.

[14] I. Houidi, W. Louati, and D. Zeghlache, "A distributed virtual network mapping algorithm," in *Proceedings of IEEE International Conference on Communications, ICC 2008, Beijing, China, 19-23 May 2008*. IEEE, 2008, pp. 5634–5640.

[15] F. Samuel, M. Chowdhury, and R. Boutaba, "Polyvine: policy-based virtual network embedding across multiple domains," *J. Internet Serv. Appl.*, vol. 4, no. 1, pp. 6:1–6:23, 2013.