Joint resource management and action scheduling for carrier-grade NFV

Łukasz Rajewski
Orange Polska
Warsaw, Poland
lukasz.rajewski@orange.com

Andrzej Bęben
Warsaw University of Technology
Warsaw, Poland
abeben@tele.pw.edu.pl

Ajay Mahimkar
AT&T Labs - Research
Bedminster, NJ, USA
mahimkar@research.att.com

***Abstract*—Software upgrades, reconfiguration, and deployment of new services happen so frequently in the carrier-grade VNF-based networks that operators need automated orchestration and change management systems. The current approaches deal independently with each service and concentrate mainly on resource allocation or optimization of the change execution processes. We propose a new Joint Resource Allocation and Scheduling (JRAS) approach, where the orchestrator manages changes, addressing the diverse requirements and dependencies of all services together. These features result in significant gain achieved by JRAS compared to other approaches, as shown by the performed numerical analysis.**

***Index Terms*—NFV, orchestration, virtualization, change management, task scheduling**

I. Introduction

There is a growing trend in telecommunication infrastructures to migrate to software-based, fully virtualized solutions benefiting from Cloud Computing and Network Function Virtualization (NFV) [1], [2]. The "softwarization" of the telecommunication infrastructure makes it much more: i) *flexible*, allowing for fast deployment, reconfiguration, and upgrade of services, ii) *scalable*, due to dynamic VNF performance adjustment, and, iii) *open* to enable automated Change Management (CM) processes.

These features are attractive for carrier-grade networks, where operation teams almost continuously apply changes [3]. The migration towards virtual appliances enables fully automated CM procedures reducing human-originated errors [4]. However, any changes must be carefully introduced into the network in a bid to minimize the risk of breakdown and misbehavior of the offered services. Given the large scale and complexity of carrier-grade networks, the design of the CM process is a serious challenge for the network operators.

The main challenge in the development of automated CM systems is the design of an efficient orchestration approach that will schedule and execute Life Cycle Management (LCM) actions for both currently running and future planned services. The orchestrator must reconcile diverse service requirements, minimize allocated resources and operation costs [5]. So, it plans and triggers the execution of CM actions, resolves potential conflicts, avoids service degradation, and satisfies Zero Downtime Deployment (ZDD) objectives [6].

The project was partially funded by POB Research Centre Cybersecurity and Data Science of Warsaw University of Technology within the Excellence Initiative Program - Research University (ID-UB).

978-3-903176-32-4

The orchestration approaches widely used, e.g., in ONAP, OSM MANO [7], or Kubernetes [8], assume that CM processes are planned and executed independently for each offered service. They ignore the impact of other actions concurrently performed by the orchestrator, which can potentially conflict or cancel out the execution of actions, and increase the expected execution time. The common practice is to use the *closed loop* paradigm, where actions are triggered whenever the analysis of monitored Key Performance Indicators (KPI) indicates the need for the system's state change [9]–[11]. As a consequence, any corrective actions are taken *a posteriori*, so they are usually late and inadequate to the actual situation. They also require coordination between instances [12], [13].

We propose a new orchestration approach, called Joint Resource Allocation and Scheduling (JRAS) orchestration. It is responsible for scheduling CM actions by taking all actions together for all orchestrated VNF service instances. The JRAS objectives are: i) schedule and execute VNF Components' (VNFC) allocation actions and traffic distribution actions in the best way, ii) guarantee the lowest possible service disruption and the highest Service Level Agreement Satisfaction (SLAS), iii) effectively utilize compute and network resources, and iv) assure the low cost of performed CM operations.

As traffic demand changes over time, the JRAS algorithm continuously manages VNF instances by adjusting their VNFC placement to the changing traffic demands and schedules all CM actions taking into account: i) available resources and their cost, ii) quality of instantiated services, iii) changing demands of additional orchestration actions required for service maintenance, iv) relations and constraints given by these actions like conflicts with other actions or time of their execution, v) features or limitations of orchestration engine (e.g. only one action at the same time executed for VNF instance).

Our paper is organized as follows: Section II presents the analysis of the related work and motivation for our approach. In Section III, we formulate the JRAS problem as a mixed-integer programming problem, and then we present the proposed orchestration method. The results of performance evaluation showing the benefits of our joint approach over the others are described in Section IV. Finally, Section V summarizes the paper and gives an outline for future work.

II. Problem statement & motivation

Automated Change Management involves three primary steps: (i) designing the CM process for a given type of change and service, (ii) scheduling of changes involving activities, and (iii) executing the change process. Today, CM processes

are only partially automated and typically are focused on ad hoc point solutions. Typically, these are not readily scaled or re-used. Change coordination is critical across groups to minimize the risk of multiple activities causing unnecessary customer impact and effectively manage operational resources. The authors of [14] proposed the solution of time-based scheduling and coordination of CM operations for physical network functions. The paper also defines the CM process as a composition of different building blocks that must be well-coordinated, having in mind that their composition into workflows depends on the use case. Nevertheless, the proposed solution shows that the proper CM algorithm can reduce the overall process duration and the risk of network failure, which is critical in carrier-grade environments.

The NFV paradigm brings new challenges for CM operation as VNF-based services are very dynamic in terms of their topology, allocated compute, and network resources. In carrier-grade NFV environments, services compete for resources, and they have demanding SLA requirements - including guaranteed continuity of service operation. These services need constant maintenance (like reconfiguration, scaling, upgrade), but the CM operations cannot impact SLA. Multiple orchestration actions might conflict on the same or dependent VNF instances and hence their coordination is required to avoid errors in the CM process. For stateful VNFs, the orchestrator has to carefully handle the state and traffic to minimize the disruption to the service during CM operations. For example, in a build-and-replace approach [15], new VNF instances are created, then the traffic is migrated, and finally the old VNF is deleted. For stateless VNFs, this method can accomplish hitless upgrades, but it requires sophisticated coordination of the order of executed orchestration operations. It is also demanding from the perspective of consumed resources. A complete service upgrade process can double the volume of resources consumed by service during the operation's execution. If we want to use resources efficiently, there should be no reserve of resources for orchestration operations executed in the near future. As a result, virtualized services can run with uncoordinated use of resources that can impact the SLAS of other coexisting services. To ensure proper SLAS, the orchestrator continuously needs to adapt virtualized services by their scaling and redistribution of traffic.

The problem of VNF placement is well studied in the recent literature and the spectrum of solutions for dynamic placement of VNFs with resource optimization for SFC is very broad [16]–[20]. These solutions address the optimal use of resources or the end-to-end quality of the service based on optimization, heuristic or AI approaches. However, they do not focus on the root cause of bad VNF performance that may lead to the necessity of VNF placement or traffic redirection. A good example of how network update operations and resource allocation can be coordinated is presented in [21]. The mentioned software upgrade example shows that orchestration operations may have many more dependencies. Therefore, there is a need to address them in the context of any CM or LCM operations and their influence on service quality.

The closed-loop orchestrations aim to improve the overall service quality. They react to any service changes based on the delayed feedback information. This approach is sufficient for the adaptation of the service to slow changes, but CM operations are typically unpredictable and bursty in nature. It results in sudden changes that are hard to overcome *a posteriori* for closed-loop automation. A detailed description of closed-loop automation and its current implementation in ONAP is presented in [22].

Major carriers and vendors are involved in enhancing and streamlining the CM process using an open-source platform-based approach in ONAP [23]. The key idea is the design of the CM process from predefined building blocks and its scheduling for orchestrator's execution based on predefined criteria. Unfortunately, the CM solution in ONAP is not integrated with its orchestration engine and is not aware of any orchestration workflows executed at the same time.

We formulate a new problem of joint scheduling of CM and LCM operations with the management of resources for VNFC placement. The proposed solution eliminates the negative impact of their execution by the orchestrator. It dynamically adjusts the consumed resources and utilizes them properly to execute CM and LCM operations in the best moment, from the service quality perspective. It also dynamically schedules actions to find the best execution plan based on actions' properties and their relations. We also want to reduce the uncertainty that such operations bring in for closed-loop automation in NFV to let it deal only with inter-orchestration changes of slow or more predictable characteristics.

III. The JRAS orchestration approach

In this section, we formulate the Joint Resource Allocation and Scheduling (JRAS) problem as a mix-integer programming problem, and then we present the JRAS orchestration approach that manages changes in a carrier-grade network.

A. The model of VNF-enabled infrastructure

Let us consider an exemplary VNF-enabled carrier-grade infrastructure, which is modeled as a graph $G(N, E)$, where N is a set of data centers with elements $n_l \in N$, indexed by their locations $l = 1, \ldots, k, k = |N|$, and symbol $|N|$ is the cardinal of the set N. The set of edges E covers all links $e_l \in E$ connecting data center n_l to an over-provisioned backbone transport network. The over-provisioning assumption allows us neglecting the network topology and details about the actual traffic distribution. It can be relaxed in further works.

The resources provided by a given data center are described by $dc_{l,r} \in \mathbb{Z}^+$, where l denotes data center location, while $r \in R, R = \{c, m, b\}$ indicates a specific type of resources. We limit their number to just three basic types that are: i) computing power (c), expressed by the number of available CPUs, ii) the amount of RAM memory (m) expressed in GBs, and iii) the bandwidth (b) of access link e_l expressed in Gbps. One may further extend the considered set of resources at the cost of increased computation complexity. The unit transmission costs between two data centers are modeled by a nonnegative square symmetric distance matrix TC with elements $tc_{l_1,l_2} \in \mathbb{R}^+$ proportional to the estimated pairwise distance between the data center locations l_1, l_2.

The modeled NFV infrastructure will host an arbitrary number of independent VNF service instances, $s \in S$. We model each service s as a directed acyclic graph following a Service Function Chain concept [24]. It defines the workflow and dependencies between VNFC component types, $v \in V$

of particular service s. In the considered case, a service s is composed of two types of VNFC components that are: i) a virtual Service Component (vSC) handling users' requests, e.g., a web server or DNS resolver, and ii) a virtual Load Balancer (vLB) responsible for redirecting requests to the appropriate vSC. Setting-up a new VNFC instance v consumes a certain amount of resources $fr_{v,r} \in \mathbb{Z}^+, r \in R, s \in S$ but offers additional service capacity $fc_v \in \mathbb{R}^+, v \in V$. The fc_v capacity indicates the upper bound of traffic carried by the VNFC instance with satisfying the predefined SLA (Service Level Agreement) level. Therefore, the capacity offered by a given VNF can be scaled up and down by tuning the number of running VNFC instances. Note that we intentionally follow a conservative, linear approximation of service capacity to keep our model as generic as possible. Otherwise, one must analyze the details of the offered services to consider the impact of multiplexing gain. Once the offered traffic exceeds the VNF service capacity, the overloading traffic is lost, leading to the degradation of the agreed SLA.

Service instances $s \in S$, by design, handle the traffic demands generated by users. We assume that users send their requests (or their requests are redirected) to the nearest data center dc_l. Therefore, we describe the traffic demand by a nonnegative service demand matrix SD with elements $sd_{l,s} \in \mathbb{R}^+$ corresponding to the estimates of some pairwise user location $l : 1 = 1, \ldots, k$ and service instance $s \in S$.

B. Orchestration process

The orchestration process continuously adjusts the number and placement of deployed VNFC instances to the current traffic demand. It satisfies the SLA offered by VNF services and minimizes the costs of compute and network resources. The orchestrator performs actions at a given moment on a given instance of VNFC. Therefore, we assume a discrete-time model where the system state is described in a given time slot $t \in T, T = 1, 2, \ldots, w$, of iteration $i \in \mathbb{Z}^+$. The value w of time slots in i^{th} iteration can be arbitrary but should be short enough to limit the computation effort. Depending on the required changes in the VNF service's state orchestrator also performs different sequences of actions in each iteration. For example, increasing traffic demands require scaling up the performance of a given service. It requires instating new VNFC components or enforcing a change in demand allocation among existing VNFC instances.

In our model, we consider several orchestration actions (OA) $a \in A$ that the orchestrator can perform on the VNF service instances. The set A covers three basic actions: i) Scale-Out (SO) to increase the performance of a given service, ii) Scale-In (SI) to reduce performance by evicting some VNFC instances, iii) Traffic Distribution (TD) to rearrange the allocation of users' demand. Basically, they are initiated by the orchestrator to adjust the performance of the VNF services. The set A covers also additional orchestration actions (AOA), denoted as $a_1, a_2, \ldots$, that can be defined arbitrarily as a set $A' \in A \setminus \{SI, SO, TD\}$. They load the orchestrator system, so they disturb the execution of basic actions performed in parallel on VNFC. Therefore, we model AOA demand as a nonnegative action demand matrix AD with elements $ad_{a,l,s,v} \in \mathbb{Z}^+$ expressing the number of AOA actions $a \in A'$ waiting to be perform on the VNFC component v, the service s, placed at l. As one can imagine a large number of AOA, instead of explicitly defining them, we focus on a specific set of AOA's properties: i) **Evict VNFC** feature causes that VNFC instance cannot handle traffic in time of action's execution, ii) **Lock VNF** causes the inability of any other action execution on all VNFC instances belonging to VNF, iii) **Lock RES** feature makes compute and network resources unavailable when VNFC is not created yet. Finally, iv) **Duration** feature defines the number of time slots required to execute an action.

Note that the orchestrator schedules the execution of all OA on the same infrastructure. Therefore, the main challenge of orchestration scheduling is to avoid conflicts, service break-outs, and unnecessary resource reservations among all running services and their VNFC instances.

C. Joint Resource Allocation and Scheduling

JRAS is an orchestration algorithm designed to manage changes in a carrier-grade infrastructure. In advance of the currently used orchestration algorithms, it takes into account the impact of the execution of other orchestration actions during the optimization of the scheduling process. The proposed JRAS algorithm uses three main decision variables.

The component placement (cp) $cp_{l,s,v,t} \in \mathbb{Z}^+$ describes the number of allocated VNFC instances. It is indexed by a given service instance s and a given VNFC v, deployed in a given location l at the time slot t. This variable is used to allocate VNFC instances in the infrastructure.

The demand allocation (da) $da_{l_1,l_2,s,v,t} \in \mathbb{R}^+$ describes the amount of demands generated by users located in l_1 to the service s that are served by the VNFC components v located in l_2. This variable determines how user demands described in the demand matrix SD are allocated to VNFC instances deployed in particular data centers.

The action plan (ap) $ap_{l,a,s,v,t} \in \mathbb{Z}^+$ denotes the number of orchestration actions of a type a performed on the VNFC component v belonging to the service s and running at a given place l during the time slot t. This variable determines the schedule of actions performed by the orchestrator.

The objective of the JRAS approach is to maximize the overall SLA satisfaction and effectively use the available resources during the entire CM process. Therefore, we define JRAS as an optimization problem of finding the minimum value of the objective function $f(.)$ defined by (1).

$$
\begin{aligned}
f(.) = \alpha * \sum_{t\in T}\sum_{s\in S}\sum_{l_1=1}^{k}\left(sd_{l_1,s} - \sum_{v\in V}\sum_{l_2=1}^{k} da_{l_1,l_2,s,v,t}\right) \\
+ \beta * \sum_{t\in T}\sum_{s\in S}\sum_{\in V}\sum_{l_1=1}^{k}\left(cp_{l_1,s,v,t} * \sum_{r\in R} fr_{v,r} * ci_{v,r}\right) \\
+ \gamma * \sum_{t\in T}\sum_{s\in S}\sum_{v\in V}\sum_{l_1=1}^{k}\sum_{l_2=1}^{k} da_{l_1,l_2,s,v,t} * tc_{l_1,l_2} \\
+ \delta * \sum_{s\in S}\sum_{v\in V}\sum_{l=1}^{k}\sum_{a\in A'}\left(ad_{a,l,s,v} - \sum_{t\in T} ac_{a,l,s,v,t}\right).
\end{aligned}
\tag{1}
$$

It covers: i) the cost of the unsatisfied volume of demand expressed by the difference between offered and allocated demands, ii) the cost of resources consumed by running VNFC, that covers VNFC handling requests and currently

blocked, iii) the transmission costs between VNFC locations, and iv) the cost of waiting AOA. Moreover, $\alpha, \beta, \gamma, \delta$ are normalizing (into $0-1$ space) and weighting factors enabling tuning the algorithm behavior, $ci_{r,v}$ denotes the cost of r-type resource used by the VNFC instance, while $ac_{a,l,s,v,t}$ is an auxiliary variable determining how many of planned actions $ap_{l,a,s,v,t}$ are finalized in the slot t. In our model, several constraints limit the feasible solutions. Below, we present only the most important ones because of limited paper space.

The VNFC can be allocated in a given data center only if there are enough available compute and network resources.

Therefore, in each time slot t, the sum of resources allocated for all services s and their VNFC components v located in a given data center l must be less or equal to the number of available resources. We call (2) a resource conservation rule.

$$\sum_{s\in S}\sum_{v\in V} cp_{l,s,v,t} * fr_{v,r} \leq dc_{l,r}, \forall t, \forall r \in R, l = 1, \ldots, k. \quad (2)$$

The user requests arrive at the nearest data center and are served there, or they are redirected to other data centers according to the availability of VNFC instances. The demand allocation must not exceed the service demands as well as the VNF service capacity offered by VNFC instances dedicated to this service in a given data center. The (3) defines the traffic conservation rule, where $bl_{a,s,v,t}$ is an auxiliary variable describing the number of temporary unavailable VNFC instances (**Evict VNFC** property) of service s and type v.

$$\begin{aligned}
&\sum_{l_2=1}^{k}\sum_{v\in V} da_{l_1,l_2,s,v,t} \leq sd_{l_1,s}, \forall t, s \in S, l_1 = 1, \ldots, k,\\
&\sum_{l_1=1}^{k}\sum_{v\in V} da_{l_1,l_2,s,v,t} \leq \sum_{v\in V}\left[cp_{l_2,s,v,t} - \sum_{a\in A} bl_{a,s,v,t}\right]\\
&\qquad * fc_{s,v}, \forall t, \forall s \in S, l_2 = 1, \ldots, k.
\end{aligned} \quad (3)$$

The VNFCs are out of the service due to currently executed CM actions. Let us recall that some AOA can require exclusive access to the entire VNF to keep configuration's consistency until AOA are finished, e.g. during the software upgrade. The number of allocated VNFC instances may change only as a results of performed CM actions. Therefore, the system state at the beginning of the time slot $t+1$ depends only on the system state at the beginning of the previous time slot t and the number and type of executed actions on VNF service s at location l (4).

$$\forall t,\ \forall s \in S,\ \forall v \in V,\ l = 1, \ldots, k,$$
$$cp_{l,s,v,t+1} = cp_{l,s,v,t} + \begin{cases} 1, & \text{if } a = SO,\\ -1, & \text{if } a = SI,\\ 0, & \text{otherwise.}\end{cases} \quad (4)$$

The orchestrator can not execute conflicting actions (**Lock VNF** property) in the same slot t on a given VNF service instance s at location l as defined by (5).

$$\forall t,\ \forall s \in S,\ l = 1, \ldots, k,$$
$$\sum_{a\in A} ap_{l,a,s,v,t} \leq \begin{cases} 1, & \text{if conflicting OA,}\\ cp_{l,s,v,t}, & \text{otherwise.}\end{cases} \quad (5)$$

Therefore, the number of executed actions equals 1 for any scheduled conflicting action or can not exceed $cp_{l,s,v,t}$ for any other ones. Moreover, the number of AOA finished during a given interaction i can not exceed $ad_{a,l,s,v}$ as defined by (6).

$$\forall s \in S, \forall v \in V, \forall l, l = 1, \ldots, k$$
$$\sum_{t\in T}\sum_{a\in A'} ac_{a,l,s,v,t} \leq ad_{a,l,s,v}. \quad (6)$$

D. Other orchestration strategies

1) The Common Resource Allocation (CRA): algorithm focuses on traditional traffic and component placement problems, trying to distribute traffic demands optimally over available compute and network resources. CRA schedules AOA on the available VNFC independently from the placement algorithm that is not aware of AOA. As a consequence, the execution of AOA may exclude some VNFC from traffic scheduling. It severely impacts VNF service quality because CRA does not compensate for a sudden VNFC shortage.

2) The Closed Loop Orchestration (CLO): receives ideal feedback information about the impact of AOA on VNF service quality, and it receives it with constant delay. CLO uses this information for enhanced, in comparison to CRA, placement. It compensates for the execution of AOA and to reduce a sudden service quality drop. In reality, closed-loop solutions do not have such precise information about the root cause of the problems, especially for unpredictable ones. Observed KPI may be affected by different inter-orchestration problems, and such feedback information may lead only to incomplete compensation of AOA.

IV. Performance evaluation

The objective of our experiments is to evaluate the JRAS orchestration approach and to compare its effectiveness to the described before CRA and CLO algorithms. All these approaches aim to adjust the performance of the running VNF service instances to the changing traffic demands and execute as many as possible of AOA. The latter may impact the operability of the VNF instance and the quality of service offered by this VNF. In consequence, we expect dependent on the algorithm trade-off between consumed compute and network resources, service quality, and efficiency of AOA execution. The effectiveness of CRA, CLO, and JRAS is analyzed in the exemplary NFV carrier-grade infrastructure loaded by a daily pattern of traffic demands, independent arrival of different orchestration tasks, like software upgrades, that introduces background load into the orchestration system.

In further experiments, we compare the efficiency of three algorithms by the juxtaposition of the VNF service quality. We analyze their cost in terms of consumed resources, and we also evaluate complexity base on the time of their execution. For that, we performed simulations of each algorithm in different traffic load conditions and with a variable number of AOA submitted to the orchestrator. Therefore, we consider four simulation scenarios that differ in assumed load conditions:

- **fix-fix** - fixed both action load and traffic load level
- **rnd-fix** - variable action load and fixed traffic load level
- **fix-rnd** - fixed action load and variable traffic load level
- **rnd-rnd** - variable both action load and traffic load level

A. Assumptions

We consider a service composed of independent VNF instances of a generic stateless type. The service is defined by a graph with two basic VNFC component types that are: i) vLB and ii) an arbitrary server component handling users' requests - in our simulations vDNS VNFC type. VNFC instances are deployed in the exemplary telco network consisting of five data centers . We measure the resource allocation cost with the consumed CPU, RAM, and throughput that a singular VNFC can handle. Each VNFC type has its CPU, RAM, and throughput requirements $fr_{v,r}$ and each data center has its CPU, RAM, and throughput limits. It results in corresponding VNFC maximal capacity dependent on its type. In our experiments, the vLB VNFC consumes twice more resources than vDNS VNFC and offers over twice higher throughput.

In the simulations, the distribution of service demands $sd_{l,s}$ across different data centers is not equal while they have the same capacity in terms of compute and network resources. In consequence, some data centers may host more VNFC than required to compensate for temporal VNFC shortage in some other location. The initial configuration of service demands $sd_{l,s}$ and action load $ad_{a,l,s,v}$ levels were selected empirically to do VNFC placements in the **fix-fix** scenario with maximum possible service quality offered. In scenarios when traffic or action load increases, algorithms may use resources from other data centers that cause a drop in the service quality. Moreover, initial VNFC placement is not equal on each data center and assures merely 50% of traffic demand satisfaction. It requires immediate VNFC placement at the beginning of the simulation to reach satisfaction with service demands.

The Service Level Agreement Satisfaction (SLAS) metric (7) represents the effectiveness of orchestration algorithms. Unsatisfied service demands, or the one that is satisfied in a non-optimal data center, reduces its value. SLAS is defined in (7), where MAX_TC is the maximum value of tc_{l_1,l_2}.

$$SLAS(.) = \sum_{t\in T}\sum_{l_1\in L}\sum_{s\in S}\sum_{v\in V}\sum_{l_2\in L} da_{l_1,l_2,s,v,t} * (1 - tc_{l_1,l_2}/MAX\_TC)/sd_{l_1,v}. \quad (7)$$

In our experiments, we evaluate fifty consecutive simulation iterations that correspond to five days of network operation with visible daily traffic pattern for variable traffic scenarios. Each iteration is divided into six time slots where the last slot t of iteration i defines the initial state of the system for iteration $i+1$ in the first slot. The simulation was split into iterations to allow for a change of the traffic and action load conditions over time and also to enable the acquisition of results due to the exponential growth of execution time with the number of time slots. The implemented algorithms and the simulation conditions assure that $ad_{a,l,s,v}$ demands are always satisfied.

Variable patterns of traffic and action load also have periodic and sinusoidal characters. Their amplitude was fitted empirically to reflect their daily changes. The goal was assuring the feasible solution for each algorithm in each iteration of the simulation. The period of demand and actions' changes differ. It is motivated by the need to obtain the accumulation of resources required for VNFC instance placement in different simulation intervals. A load of additional actions is not distributed equally over all the time slots of the iteration.

TABLE I
TYPE OF ACTION AND ITS IMPACT ON VNF AND VNFC

	Scale-Out	Scale-In	Traffic Dist	AOA
Evict VNFC		•		•
Lock VNF	•	•		
Lock RES	•			
Duration [time slot]	2	2	1	1

When the AOA load is constant, it is distributed proportionally to the number of available VNFC instances - which fluctuates.

CRA algorithm executes SO, SI, and TD orchestration actions for the optimal placement of VNFC instances required to satisfy service demand $sd_{l,s}$. CLO and JRAS algorithms embed CRA, but they also orchestrate AOA. Each OA used in our experiments has properties defined in Section III that summary is shown in Table I. Moreover, only one SO or SI operation can be executed at once for VNF what slows down the scaling of VNFC instances. The execution of evaluated AOA removes VNFC from the pool of VNFCs able to handle the traffic. Algorithms should build capacity reserve (CLO) or scale VNFCs in advance (JRAS) to avoid the reduction of the service's capacity. Both are possible only when the data center has sufficient resources. The execution of AOA also distorts the optimal service demand distribution.

B. Comparison of JRAS to other approaches

In the first experiment, we focus on the performance evaluation of the JRAS algorithm. We compare its effectiveness to the reference CLO and CRA orchestration approaches. The experiments were performed in different load conditions related to the offered service demands and AOA load.

Fig. 1 shows the obtained results with statistics of SLAS for all fifty iterations i of simulations for CRA, CLO, and JRAS algorithms, and all four considered scenarios. Each bar is the average SLAS (dot) with the standard deviation (thick line) and the max/min values' range of SLAS (thin line). Both CLO and JRAS outperform the CRA algorithm. CRA does not eliminate the negative influence of AOA. In all scenarios, the JRAS solution also outperforms the CLO algorithm, not only in terms of the higher average SLAS value but mainly by much lower SLAS variance. JRAS also is much less fragile to variable traffic or action load than CLO due to the real-time scheduling of AOA, in contrast to the feedback-related reservation of VNFCs capacity for AOA of CLO. As a consequence, for CLO the slots for the execution of AOA are predetermined. The orchestrator is not aware of AOA in real-time and does not know the actual AOA execution needs.

In all scenarios, including **fix-fix**, the optimization of resource consumption is ongoing and all the algorithms try to minimize used resources resulting in a AOA distribution change. JRAS orchestrates AOA in each of the six timeslots of the iteration along with SO, SI, and TD when CLO only performs a flat reservation of capacity for AOA.

C. Analysis of JRAS orchestration actions

In the next experiment, we focus on the analysis of orchestration actions taken by JRAS vs. other approaches in response to changing traffic load or changing load of AOA. Fig. 2 is an exemplary time plot showing how averaged SLAS changes over time (upper plot) for JRAS or CLO algorithms correlated

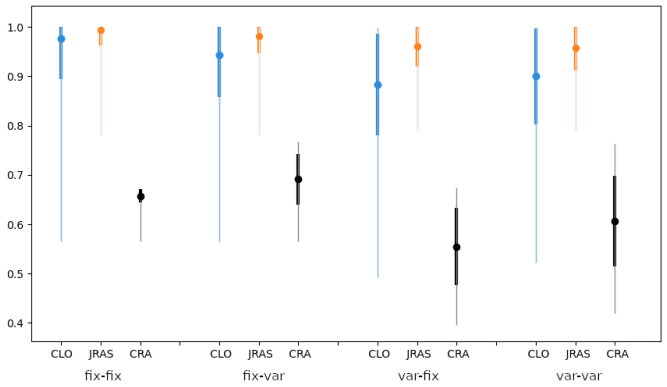

Fig. 1. The average SLAS comparison of JRAS vs. CLO and CRA in four scenarios: a) fixed action and traffic load, b) fixed actions and variable traffic, c) variable actions and fixed traffic, and d) variable traffic and actions.

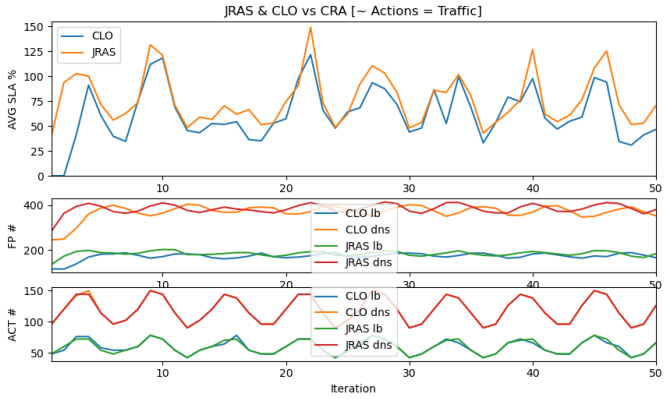

Fig. 2. The time plot of orchestration actions for JRAS, CLO approaches, where: a) the SLAS gain vs. CRA approach, b) number of allocated vLB and vDNS instances, c) number of performed additional orchestration actions.

with the CRA algorithm's SLAS. In the Function Placement (FP) plot, we can observe how the orchestrator allocates the vLB and vDNS VNFC instances. On the bottom plot (ACT) we can also observe how many AOA each algorithm executes over time. Here we present only the most prominent **var-fix** scenario from the four conducted.

The ACT plot in Fig. 2 shows the amount of AOA executed for VNF in each iteration of the simulation. The AOA change has a sinusoidal form, and the amount of executed AOA is proportional to the number of VNFC instances of vDNS or vLB network functions. The same change pattern is visible on the FP plot where the number of executed AOA follows the changes of the AOA load. The amount of vLB or vDNS VNFC instances changes sinusoidal and follows the current action load conditions - for CLO with visible feedback delay. The FP plot also shows the before-mentioned relation of resource requirements for both types of VNFC. Therefore, we can notice a doubled amount of vDNS VNFC instances required - compared to vLB. The SLAS plot shows that both CLO and JRAS algorithms significantly (over 100%) improve the SLAS, in comparison to the CRA approach, and the negative influence of AOA on SLAS can be minimized. The SLAS and FP plots show the CLO feedback delay observed in the first iterations of the simulation when AOA are not mitigated yet. The JRAS

TABLE II
AVG./STD. EXEC. TIME [MIN] FOR EACH ALGORITHM AND SCENARIO

	fix-fix	fix-var	var-fix	var-var
CRA	48 / 19	23 / 7	32 / 10	18 / 5
CLO	51 / 16	21 / 7	12 / 2	14 / 5
JRAS	137 / 40	70 / 17	56 / 12	46 / 9

gain is the highest when the AOA load is increasing and drops when the AOA load is going down. The FP plot also shows that JRAS does not consume more resources over time. Instead, it allows optimally using them. The feedback delay of CLO always delivers historical and inaccurate information about the AOA load level when JRAS utilizes real-time information of incoming AOA. It allows scheduling their execution at the best moment from the viewpoint of current resource allocation.

D. Assessment of algorithms complexity

We analyzed the complexity of JRAS by measuring the time each orchestration algorithm calculates the optimum solution. The algorithms' implementation utilizes the open-source MiniZinc framework [25]. The simulations were performed on a single thread mode with Coin-OR CBC solver [26]. Our goal was to have fair and comparable conditions for algorithms' evaluation, so the simulations were executed on the same server (40 x 2,3GHz CPU, 128GB RAM). We run all 12 scenarios at the same time and each simulation could utilize 100% power of a singular CPU. The simulations were repeated 60 times and the 95th percentile of average values of the obtained algorithms' execution time are shown in Table II.

The results show that the complexity of JRAS model calculation is up to three times worse than CLO or CRA approaches, but it does not significantly differ in particular scenarios. Nevertheless, further evaluation of the complexity of JRAS is required, considering the influence of different initial conditions and the heuristic, non-solver-based implementation of the algorithms.

V. SUMMARY AND FURTHER WORKS

In this paper, we propose a new Joint Resource Allocation and Scheduling orchestration approach. This method deals with the allocation and scheduling of change and life cycle management operations in NFV environments, jointly with VNFC placement. JRAS manages changes, addresses diverse requirements and dependencies of all services together. It enhances and stabilizes the quality of virtualized service, as has been shown by the performed experiments. The proposed solution eliminates short-term changes introduced by intra-orchestration, CM, or LCM operations, that can produce false feedback for closed-loop solutions. As a consequence, JRAS enables their adoption in NFV environments for inter-orchestration changes of slow nature that are easier to predict. At the same time, the complexity of JRAS is of the same order as other analyzed algorithms. In future work, we plan to extend the JRAS model to ensure its generality, to implement its heuristic algorithm, and to support Containerized Network Functions (CNF). We will also analyze the performance of JRAS by taking into account different topologies, relationships between VNFC types, varying workloads, and action properties.

REFERENCES

[1] S. Svorobej, M. Bendechache, F. Griesinger, and J. Domaschka, *Orchestration from the Cloud to the Edge*. Cham: Springer International Publishing, 2020, pp. 61–77.

[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network Function Virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[3] GSMA, "Considerations, Best Practices and Requirements for a Virtualised Mobile Network," GSMA, Tech. Rep., 2017. [Online]. Available: https://www.gsma.com/futurenetworks/wp-content/uploads/2017/05/Virtualisation.pdf

[4] M. Mushi, E. Murphy-Hill, and R. Dutta, "The human factor: A challenge for network reliability design," in *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, March 2015, pp. 115–118.

[5] M. Chiosi, D. Clarke, P. Cablelabs, and et al., "Network functions virtualisation (nfv) network operator perspectives on industry progress," 10 2013.

[6] U. Naseer, L. Niccolini, U. Pant, A. Frindell, R. Dasineni, and T. A. Benson, "Zero Downtime Release: Disruption-Free Load Balancing of a Multi-Billion User Website," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 529–541. [Online]. Available: https://doi.org/10.1145/3387514.3405885

[7] "Benchmarking open source NFV MANO systems: OSM and ONAP," *Computer Communications*, vol. 161, pp. 86 – 98, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366420305946

[8] J. Ellingwood, "An Introduction to Kubernetes," accessed: 2020-09-29. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes

[9] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[10] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing mape-k feedback loops for self-adaptation," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015, pp. 13–23.

[11] T. Meriem, R. Chaparadza, B. Radier, S. Soulhi, J. Lozano, and A. Prakash, "ETSI white paper no. 16 GANA - generic autonomic networking architecture reference model for autonomic networking, cognitive networking and self-management of networks and services," ETSI, Tech. Rep., 01 2017.

[12] J. Panerati, M. Maggio, M. Carminati, F. Sironi, M. Triverio, and M. D. Santambrogio, "Coordination of independent loops in self-adaptive systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 2, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2611563

[13] N. Oliveira and L. Barbosa, "Self-adaptation by coordination-targeted reconfigurations," *Journal of Software Engineering Research and Development*, vol. 3, 12 2015.

[14] M. A. Qureshi, A. Mahimkar, L. Qiu, Z. Ge, M. Zhang, and I. Broustis, "Coordinating rolling software upgrades for cellular networks," in *International Conference on Network Protocols, ICNP*, 2017.

[15] ETSI NFV, "GS NFV-REL 006 - V3.1.1 - Network Functions Virtualisation (NFV) Release 3; Reliability; Maintaining Service Availability and Continuity Upon Software Modification," ETSI, Tech. Rep., 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/006/03.01.01_60/gs_nfv-rel006v030101p.pdf

[16] H. Hawilo, M. Jammal, and A. Shami, "Network Function Virtualization-Aware Orchestrator for Service Function Chaining Placement in the Cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 643–655, 2019.

[17] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently Embedding Service Function Chains with Dynamic Virtual Network Function Placement in Geo-Distributed Cloud System," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2179–2192, 2019.

[18] Q. Zhang, Y. Xiao, F. Liu, J. C. S. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 731–741.

[19] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin, "Intelligent VNF orchestration and flow scheduling via model-assisted deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 279–291, 2020.

[20] G. Dandachi, A. De Domenico, D. T. Hoang, and D. Niyato, "An artificial intelligence framework for slice deployment and orchestration in 5G networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 858–871, 2020.

[21] Y. Liu, Y. Li, M. Canini, Y. Wang, and J. Yuan, "Scheduling multi-flow network updates in Software-Defined NFV systems," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 548–553.

[22] L. Fallon, J. Keeney, and R. K. Verma, "Autonomic closed control loops for management, an idea whose time has come?" in *2019 15th International Conference on Network and Service Management (CNSM)*, 2019, pp. 1–5.

[23] "Optimization Framework: Change Management Schedule Optimization," accessed: 2020-09-29. [Online]. Available: https://docs.onap.org/projects/onap-optf-cmso/en/stable/

[24] J. Zhang, Z. Wang, N. Ma, T. Huang, and Y. Liu, "Enabling Efficient Service Function Chaining by Integrating NFV and SDN: Architecture, Challenges and Opportunities," *IEEE Network*, vol. 32, no. 6, pp. 152–159, 2018.

[25] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "MiniZinc: Towards a Standard CP Modelling Language," in *Principles and Practice of Constraint Programming – CP 2007*, C. Bessière, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 529–543.

[26] "COIN-OR/CBC," accessed: 2020-09-29. [Online]. Available: https://github.com/coin-or/Cbc