

Optimal control of VNF deployment and scheduling

Mark Shifrin, Erez Biton, Omer Gurewitz

Abstract—Managing network-related resources involves sophisticated trade-off between flexibility, high performance standards, latency demands which adhere to service level agreements (SLAs) and cost constraints. Network functioning virtualization (NFV) opens new challenges to the remote network management which combines activation and control of virtual machines (VMs) according to variable demand. Currently, this functionality is being handled by harnessing the traditional orchestration algorithms using suboptimal heuristics. We model the problem of virtual network function (VNF) allocation by queuing system with flexible number of queues, which captures variety of constraints including queue deployment and displacement, delay cost, holding cost, scheduling reward and fine. Next, we model the system by Markov decision process (MDP) and numerically solve it to find the optimal policy. We show analytically and by simulations that the optimal policy possesses decision thresholds which depend on several parameters.¹

I. INTRODUCTION

Recent network functioning virtualization (NFV) development suggests that networking-related tasks which previously used to run on-premises will be moved to the cloud, [1].

The breakthrough that the NFV concept represents may cause a paradigm shift for many stakeholders including network operators, solution vendors, service integrators, providers, and service users. However, several issues continue to hinder the management of NFV infrastructure (NFVI) resources. First, the NFV concept has many types of NFVI resource requirements in order to launch requested network services. The NFVI includes not only physical and virtual resources in a single data center but also network resources between multiple data centers. Therefore, software network appliances would be deployed in the NFVI with various hardware and software requirements, quality of service (QoS) provisioning levels, and/or negotiated service level agreements (SLAs).

Enterprise which decides to virtualise any of her network services needs first to *deploy* (“*build*”) the corresponding service, in order to be able to *run* VNF instances. The deployment involves having leased a certain amount of designated hardware, for CPU, memory and networking. Once deployed, the enterprise has to pay per time of usage for the leased resources, even if they are not loaded. The process of deployment can take time, and an additional deployment cost. Hence, having idle resources is undesired. On the other hand, the delay involved with the deployment or lack of space for

the new tasks can cause some VNF instances to be rejected from running which would imply a fine to the enterprise. Note that in some cases, the *displacement* (“*destroy*”) operation, i.e., the process of releasing the resources can also incur a cost. An additional basic demand is to facilitate scaling of VNF. That is, the amount of virtual resources allocated to a certain VNF is dynamically increased (via a scale out operation) or decreased (scale in operation) according to the demand from the network service (e.g., number of flows a firewall handles). In scale out/in operations, we increase/decrease (i.e., deploy/displace) the number of VMs that are allocated to the VNF, respectively. Note that we do not consider the scale up and down operation, which are less common in NFV use cases. (In scale up/down operation we can, for example, add/remove CPU cores to a given VM). Clearly, increasing the number of VMs allocated to a VNF would improve the performance of the VNF, yet would consume more resources. Accordingly, our scaling decision should minimize the consumed resources, while keeping with the application required SLA. Hence, the decision whether to scale out or in, typically performed by the VNF manager (VNFM), remains an important problem that needs to be addressed, especially in dynamic scenarios.

In conjunction with the scaling decisions, we are also facing a load balancing challenge, steering the traffic flows to the different VMs and balancing the load between them. In this study, we tackle both the scaling decision and the load balancing strategy as a single problem.

The problem which the enterprise has to solve is to find *dynamic policy of VM deployment, displacement and scheduling of incoming VNF tasks*, as a function of the set of costs and the number of VNF instances currently being served in the already active VMs.

To this end, we model the problem by *queuing system* with a dynamic (but limited) number queues; each queue stands for VM running VNF instances. We assume that incoming VNF tasks have constant average rate. Upon each arrival event, the Decision Maker (DM) decides to which VM the arriving task should be scheduled, or whether it should be rejected. We allow to deploy a new VM at arrival times, as long as the limit of active VMs is not reached. At departure from a queue (i.e., running of VNF task ends) which has been left empty, DM decides whether to keep that queue alive or to destroy it. In order to reflect the load at busy VMs we introduce a delay cost which (not necessarily linearly) increases with the load. Deployment and displacement of VM consume additional resources represented by corresponding fixed costs. Once a VM was deployed, the enterprise is charged for the

¹This research was partially supported by European Union Horizon 2020 Research and Innovation Programme SUPERFLUIDITY, Grant Agreement 671566.

reserved resources with fixed per-unit of time reservation and maintenance cost. We term this *keep-alive or the holding cost*.

The DM aims to find a policy which will maximize total income in the long run. While the set of costs described above implies no trivial policy could exist, some of the parameters have contradicting impacts which may dictate certain properties of the policy. For example, in the case where the delay cost function sharply increases with a load, the DM will attempt to balance the scheduling process among as many VMs as possible. On the other hand, in the case where keep-alive cost is comparatively high, the DM may prioritize minimization of the active queues number.

In this paper, we treat this problem by introducing a control model based on MDP formulation. The solution of the MDP provides the optimal policy. Once applied, the policy reveals the average number of active queues, average number of tasks in each queue. This allows the enterprise to assess the demands and to plan ahead. Moreover, the solution to the MDP is expressed by value function which indicates what are the costs and revenues the enterprise will receive in the long run, for a given VNF scenario, modeled by the corresponding set of parameters.

A. Related Work

Challenges of VNF scheduling by means of software-defined networks (SDN) were introduced in [2]. Technical details of how the VNF could be deployed are discussed in [3]. Integration of NFV and SDN was also addressed in [4] in the context of 5G mobile core networks. In [5], queue scheduling algorithm named 2DFQ separates requests with different costs and different size across different worker threads. It also extends to the case where the costs are variable or unpredictable. Optimal scheduling in hybrid cloud environment was studied in [6]. The solution provided by the authors employs MDP as well. In [7], joint optimization of NFVI resources under multiple constraints results in approximate algorithm which aims at improving the capability of NFV management and orchestration. This work provides static placement of NFV-related resource. The problem of VNF scheduling, aiming at optimizing the performance latency, was treated by linear programming in [8].

A great deal of work provides results on queue scheduling and optimization problems. Most of works deal with fixed number of queues or queue networks, e.g. [9], [10], [11] or take this number to the limit, e.g. [12], [13], [14] and references therein. However, we treat a specific scenario, where the number of queues is finite but flexible, incorporating unique features of queue destruction and deployment costs. These features correspond to the VNF scheduling which we aim to optimize in the sense of overall cost minimization. Therefore, we develop a model which fits our goal and treat it by MDP.

II. SYSTEM DESCRIPTION

We assume our cloud NFV infrastructure (the NFVI) can host a finite yet flexible number of VMs. For simplicity, we assume that an instance of a VNF is deployed in one VM, with two instances deployed in two VMs and so forth. That is, in a scale out operation, adding a VM translates to adding another instance of the VNF. We further assume, that the load balancer is deployed in a single VM and can handle all traffic demands irrespective of the number of VNF instances (i.e., number of VMs), see Figure 1 for the schematic presentation. The total demand of a VNF (e.g., from a firewall) is modeled by Poisson process of arriving tasks with average rate λ . The incoming tasks are scheduled into one of the active queues or rejected from service, according to load balancing policy. Each VM handles the traffic demands (e.g., flows handled by the firewall) in a parallel manner. Namely, the total processing rate is equally shared between all task. Hence, the tasks never wait but are rather immediately processed. The VM's limited resources allow processing of limited number of tasks, denoted by B . For simplicity, we assume all VMs are identical. We assume each VM has exponential service times with maximal average rate μ . For simplicity, we assume that service allocation and reallocation in each VM has no time overhead. Since the minimum of exponential rates is equal to their sum, the total processing rate is always equal to μ . Hence, each VM is modeled by queue with buffer size B , having up to B servers with total processing rate equal to μ . While this model reflects the ability of VMs to provide concurrent resource sharing, our model can be easily adopted for the FIFO service, with only minor changes. Note that for the correspondence with the mathematical model, which will follow, we use terms VM and queue interchangeably.

Each admitted task gives a fixed reward, while rejected task incur a fine, denoted by $\{r, f\}$, respectively. Each queue can be deployed and destroyed dynamically, according to the demand. Intuitively, the number of queues may infinitely grow for some sets of parameters. For example, in the case where it holds $\lambda \gg \mu$, while rejected tasks incur high fines. On the other hand, the amount of available queues is expected to be bounded by general system limitations. We will assume henceforth that the number of active queues is limited. Note that in the case where no VM is currently deployed, or the existing VMs (queues) are full, the incoming task is rejected. Hence, in the case deployment time is instantaneous, rejection can be avoided by immediate queue deployment.

Denote the vector of active queues at time t by $\mathbf{q}(t)$, where $\mathbf{q}(t)$ is in a state-space denoted by \mathbb{Q} . The components correspond to the number of tasks in queues and are denoted by $q_i(t)$, where i is general indexing of the queues. While for simplicity we assumed $\mu_i = \mu$, the extension to the system with different processing rates is straightforward. The limit of number of queues is given by $|\mathbf{q}| = \mathbf{n}$, for some system-related given \mathbf{n} . The queues are limited by $-1 \leq q_i(t) \leq B$,

$i \in \{1, \dots, |q|\}$, here state "−1" means the queue is inactive. Note that this is different from state 0 which means the queue is active but empty.

1) *Delay cost structure*: Denote by h the delay cost per unit of processing time, associated with a number of hosted tasks. In particular, at i th queue at time t the cost equal to $h_i(t)$ per unit of time is inflicted. The delay cost model is represented by function which increases with the number of hosted tasks at VM. This structure reflects a load increase (and ,consequently, the overall queuing time) as the number of residing tasks at VM grows. In particular,

$$h_i(t) = q_i(t)\eta(q_i(t))h, \quad (1)$$

Where $\eta > 1$ for all possible q_i values and is positive increasing with q_i . Namely, more busy VMs function with higher delay. Hence, this cost structure penalizes for having busy VMs. Therefore, the delay cost model captures parallel processing, where the delay cost is considered for every task. Note that if we assume $\eta(i) = 1$ for all i , the cost will degenerate to the simple linear model. Observe that delay cost is the lowest, once tasks are equally dispersed over queues.

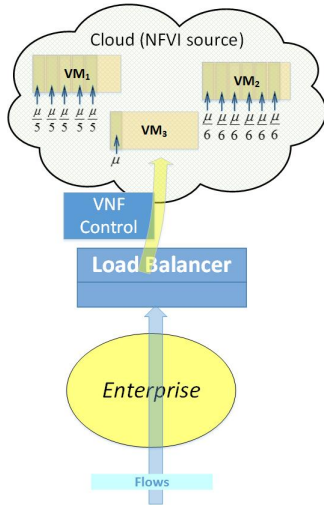


Fig. 1. System scheme, containing up to 3 VMs with maximal capacity of 7 tasks

In the next section, the MDP problem formulation follows.

III. MDP FORMULATION

Denote the scheduling vector \mathbf{u} of length \mathbf{n} , such that $u_i(t) \in \{0, 1\}$ and $0 \leq \sum_{i=1}^n u_i(t) \leq 1$. The sum is equal to 0 in the case the decision was a rejection. Otherwise, the scheduling decision at t is expressed by i , for which $u_i(t) = 1$. The *queue activation policy* is applied at each task arrival, is described by action vector $\mathbf{b}(t) = \{b_i(t)\}$, $b_i(t) \in \{1, 0\}$, where the possible values for b_i stand for "deploy", and "not deploy", respectively. The *queue destruction policy* is applied at each departure event and is described by action vector

$\mathbf{d}(t) = \{d_i(t)\}$, $d_i(t) \in \{1, 0\}$. Hence, the action space consists of the triplet $\{\mathbf{u}, \mathbf{b}, \mathbf{d}\}$.

In what follows we will deal with counting processes of the general form: $V(t) = \sup\{m ; \sum_{i=0}^m v(i) \leq t\}$, where $v(i)$ is the time between increment (e.g., arrival time, service time) $i - 1$ and i , for some process V . Denote the arrivals counting process as $A(t)$ and task completion processes as $D_i(t)$. Define the following indicator functions:

Definition 3.1 (Queue indicators). For $1 \leq i \leq \mathbf{n}$

Inactive queue :

$$\mathbf{I}_i^i(q) = 1 \quad \text{iff} \quad q_i = -1$$

Empty but active queue:

$$\mathbf{I}_i^e(q) = 1 \quad \text{iff} \quad q_i = 0$$

Queue with exactly one task:

$$\mathbf{I}_i^o(q) = 1 \quad \text{iff} \quad q_i = 1$$

Full queue:

$$\mathbf{I}_i^f(q) = 1 \quad \text{iff} \quad q_i = B$$

None of the above (denoted as normal):

$$\mathbf{I}_i^n(q) = 1 \quad \text{iff} \quad 2 \leq q_i < B$$

The building cost β is applied each time a queue is activated. The destroy cost ψ is applied each time a queue is destroyed. In most general form, these actions are allowed to be taken at arrival events and any departure events, that is, once the counting processes A and D_i increase. The corresponding cost functional discounted with discount factor γ , for policy π is given by:

$$\begin{aligned} J^\pi = & \int_0^\infty e^{-\gamma t} \left[-(\mathbf{b}(t) \cdot \beta - \right. \\ & \left. \mathbf{d}(t) \cdot \psi) (dA(t) + \sum_i^n dD_i(t)) - \right. \\ & \left. \sum_i^n (h_i(t) + \kappa * (1 - \mathbf{I}_i^i)) dt + \right. \\ & \left. (\mathbf{u}(t) \cdot r - f * (1 - \sum_i^n \mathbf{u}_i(t))) dA(t) \right], \end{aligned} \quad (2)$$

where the first part stands for the queue activation cost (J_b^π) and deactivation cost (J_d^π), the second part stands for queue holding cost (J_h^π) and delay cost (J_κ^π), while the third part stands for the scheduling rewards (J_r^π) and rejection fines (J_f^π). The cost is otherwise written by its components

$$J^\pi = -J_b^\pi - J_d^\pi - J_h^\pi - J_\kappa^\pi + J_r^\pi - J_f^\pi$$

The value function associated with initial state q is given by

$$V_q = \max_{\pi} J^\pi(q)$$

We now write the Bellman equation for a simplified and more realistic scenario assuming that build operations can be only done at arrivals, while destroy operations can be only

done at departures. Denote by e_i vector of length \mathbf{n} with value 1 at i th coordinate and zeros in all other coordinates. The Bellman equation reads

$$V_q = \left[\sum_i \mathbf{I}_i^n \mu_i V_{q-e_i} + \sum_i \mathbf{I}_i^f \mu_i V_{q-e_i} + \sum_i \mathbf{I}_i^o \mu_i \max\{V_{q-e_i}, V_{q-2e_i} - \psi\} + \lambda \max \left\{ \max_{i, \mathbf{I}_i^f=0} V_{q+e_i + \mathbf{I}_i^1 e_i} - \beta * \mathbf{I}_i^1 + r, V_q - f \right\} + C(q) \right] \delta_q, \quad (3)$$

where the cost function $C(q)$ is calculated by $C(q) = \sum_i^n h_i + \kappa * (1 - \mathbf{I}_i^1)$. Note that in the case where all queues are full, the outcome of the inner maximization is empty. In this case, the second term in outer maximization is selected. The derivation of the equation above is in the Appendix A.

IV. OPTIMAL POLICY STUDY

Denote two states a and b , with queue vectors given by q^a and q^b . Define *state domination* by $q^a \succeq q^b$ if and only if $q_i^a \geq q_i^b, \forall i \in \{1, \dots, \mathbf{n}\}$. Moreover, we assume that a and b have the *same number of idle queues* (this restriction is motivated by the impact of build and destroy costs). We next show the following lemma:

Lemma 4.1 (Value function domination). *For any $q^a \succeq q^b$ it holds $V(q^a) \leq V(q^b)$.*

Proof. To show the statement define two systems, denoted by a and b . At time t the state of each system is denoted by a_t and b_t . Next, we stochastically couple these systems. Namely, we apply the same policy to both states, denoted as π_{ab} . The policy is optimal for the system a , while system b mimics all actions from system a . In particular, a task which is scheduled in a is scheduled in b into the same queue. In the case where the rejection applies in both systems, the task is rejected. In the case where there is a forced rejection (that is, the corresponding queue in a was full), then it applies to b only in the case the same queue was full as well. Each served task departs concurrently from a and b . In the case where during departure in a the same queue in b was empty - nothing happens in b . The "destroy" and "build" decision also follow the same policy and are executed concurrently. (Hence, system b will keep alive queues even if empty as long as they are kept in a). Now denote the first time τ when both states become equal, that is $a_\tau = b_\tau$. Clearly, by definition of π_{ab} it holds $e^{-\gamma\tau} V(a_\tau) = e^{-\gamma\tau} V(b_\tau)$. Next, as for the rewards in time interval $[a, \tau)$, it holds $r(a_t) \leq r(b_t)$ and as for the fines it holds $f(a_t) \geq f(b_t)$. Finally, by π_{ab} , all holding costs and costs incurred by build and destroy actions which accumulate to the total cost according to (2) are higher in a . Therefore,

$$V(b_0) \leq J^{\pi_{ab}}(b_0) \leq V(a_0)$$

Hence, the statement in the lemma follows. \square

Denote the average number of tasks, (resp. the variance) in the system, by $\overline{q(t)}$, (resp. by $\widehat{q(t)}$). Clearly, for any q^a, q^b

such that $\overline{q^a} = \overline{q^b}$ and $\widehat{q^a} < \widehat{q^b}$, for the optimal policy π and *partial cost* of the form $J_p^\pi = -J_h^\pi + J_r^\pi - J_f^\pi$ it holds $J_p^\pi(q^a) \geq J_p^\pi(q^b)$.

To this end, for any $q^b \succeq q^a$ define non-negative difference function $\alpha(\cdot, \cdot)$ such that

$$V(q^b) + \alpha(q^a, q^b) = V(q^a) \quad (4)$$

Note that $\alpha(q^a, q^b)$ is non-negative by Lemma 4.1. We now present the following proposition, which states a *threshold policy* in action "build".

Proposition 4.1. *If by optimal policy in state q^a at arrival event it holds $b_i(q^a) = 1$, then for all states such that $q^b \succeq q^a$ holds $b_i(q^b) = 1$.*

The proof appears in Appendix B.

V. NUMERICAL RESULTS

In this section, we present simulation results which both show additional threshold properties of optimal policies, and provide a comprehensive study of the value functions and the corresponding policies. We explored a system with five identical queues.

Figure 2 demonstrates the impact of the keep-alive queue cost. The simulation was run with negligible delay cost, rejecting fine equal to 10, $\lambda = 4$ and $\mu_i = 1, \forall i$. Buffer limit was 4 tasks. One sees that there is a small region of keep-alive queue cost, where the number of active queues declines. The trade-off in this simulation is the accumulatively paid fine against the accumulatively paid keep-alive queue cost. As long as the queues are identical, there is no preference which queue should be kept idle. That is why the number of active queues declines to zero within a very small interval of keep-alive queue cost. Hence, once it is more affordable to pay the fines by rejecting all incoming tasks rather than maintaining the VMs (i.e., the queues) the queues stay idle at all times. Figure 3 shows simulation of the delay cost h , while

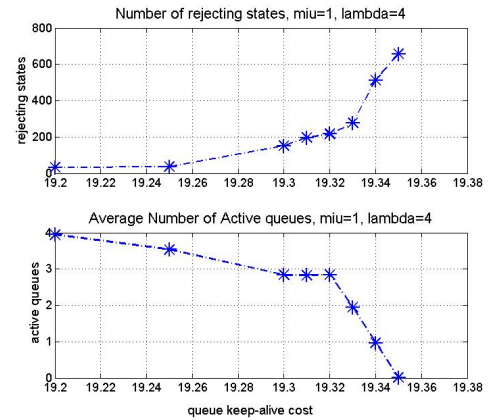


Fig. 2. Impact of the allocated VNF cost

the keep-alive cost was fixed equal to 1, the buffer size of each queue was 6. The arrival intensity was 4.75 and $\mu_i = 1, \forall i$. Rejecting fine was equal to 10. The delay constants were $\eta_1 = 1, \eta_2 = 1.8, \eta_3 = 2.5, \eta_4 = 3.5, \eta_5 = 4.5, \eta_6 = 5.5$. By cost model in (1), one expects that the tasks will be balanced in the queues. Observe that in this simulation, the keep-alive cost was low enough to allow that. Indeed, all queues were active while the total average number of tasks declined with the delay cost. Observe that in this simulation, the interval of the varying cost was significantly larger. This stems primarily from the fact that η_1 is small. Note that the lower graph in both simulations shows the number of rejecting states, out of the states-space.

Additional thresholds can be seen in "build" and "destroy" actions. For example, if β and/or ψ are high if compared to keep-alive cost, the optimal policy acts to leave all queues active, even if empty. To conclude, the set of system pa-

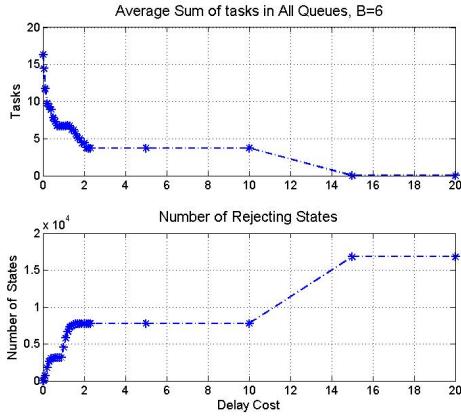


Fig. 3. Impact of the delay cost

rameters will determine if the optimal policy will prioritize balanced and mostly active queues or a small number of active and comparatively busy queues. The values of β and ψ will determine if keeping alive the empty queues is economically reasonable.

A. Value function presentation and discussion.

The graphical presentation of value function, where $B = 6$ is seen in Figure 4. Horizontal axis values stand for states enumeration. See that the function forms 8 large regions. Each region corresponds to the fixed state of q_0 . The first region corresponds to q_0 being idle, the second region corresponds to q_0 being active and empty, etc. Observe that as q_0 fills up, the entire corresponding region has lower values. Observe next that each region contains 8 subregions, corresponding to the states of q_1 . These subregions also exhibit lower values as q_1 fills up. The same drift is identified in other queues. Figures 5,6 demonstrate the closeups of selected sub-regions and states.

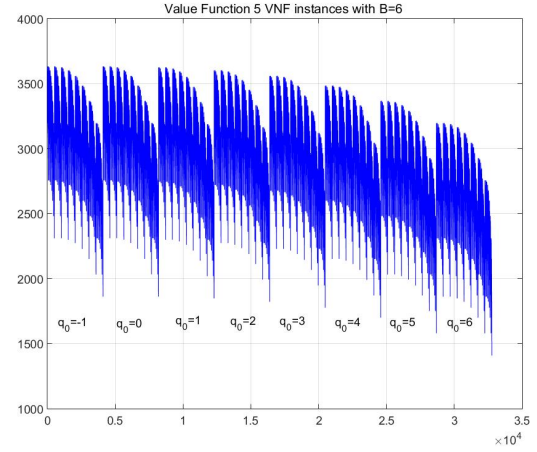


Fig. 4. Value function - all states

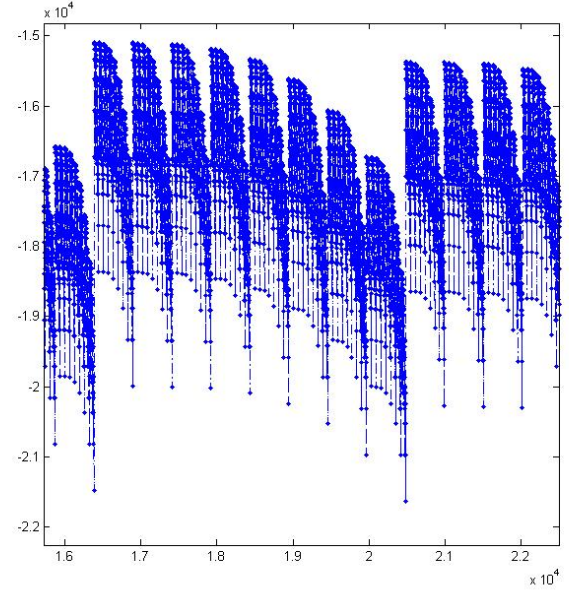


Fig. 5. Value function - closeup

APPENDIX

A. Derivation of Bellman equation

Note that we assume the deployments only occur at arrivals, while destructions occur only at service ends. Write the cost

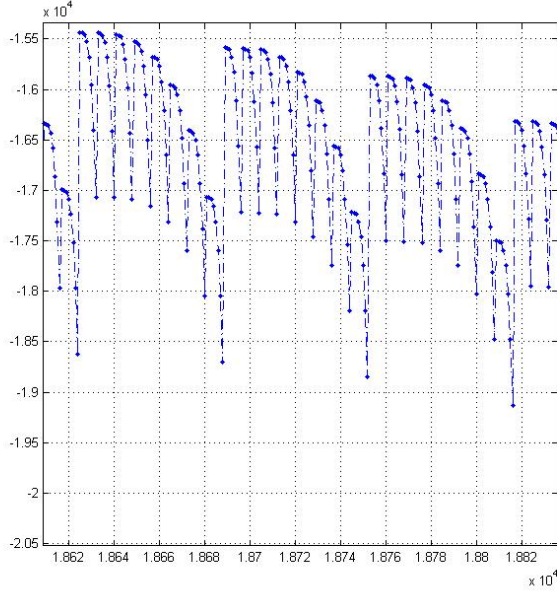


Fig. 6. Value function - closeup

function as follows:

$$J = \int_0^\infty e^{-\gamma t} \left[(\mathbf{b}(t) \cdot \beta + \mathbf{u}(t) \cdot \mathbf{r} - f * (1 - \sum_i^n \mathbf{u}_i(t))) d\mathbf{A}(t) + (\mathbf{d}(t) \cdot \psi \mathbf{D}(t)) + \sum_i^n (h_i^d(t) + \kappa * (1 - \mathbf{I}_i^i)) dt \right],$$

We take infinitesimal θ such that only one or no event can happen (probability of more than one is negligible), and use the dynamic programming principle.

$$J(q) = \int_0^\theta e^{-\gamma t} \left[(\mathbf{b}(t) \cdot \beta + \mathbf{u}(t) \cdot \mathbf{r} - f * (1 - \sum_i^n \mathbf{u}_i(t))) d\mathbf{A}(t) + (\mathbf{d}(t) \cdot \psi \mathbf{D}(t)) + \sum_i^n (h_i^d(t) + \kappa * (1 - \mathbf{I}_i^i)) dt \right] + e^{-\gamma \theta} J(q(\theta)) = J^\theta + e^{-\gamma \theta} J(q(\theta)),$$

Expanding for all options of $J(q)$ after the time θ we have:

$$J^\theta + e^{-\gamma \theta} \left[\lambda \theta J(q + \mathbf{u}) + \sum_i^n \mu J(q - e_i) + (1 - \lambda \theta - \sum_i^n \mu) J(q) \right] \quad (5)$$

We use the following derivation:

$$\mathbb{E} \int_0^\theta e^{-\gamma t} dA(t) = \lambda \int_0^\theta e^{-\gamma t} dt = \lambda \frac{1 - e^{-\gamma \theta}}{\gamma} = \lambda \theta, \quad (6)$$

where we substituted $A = V$, i.e. the arrival process. In what follows we use $e^{-\gamma \theta} \simeq 1 - \gamma \theta$ as $\theta \rightarrow 0$. Calculate first the time used for a service in the infinitesimal interval $[0, \theta]$.

$$T_i(t) = \int_0^\theta (1 - \mathbf{I}^e(t))(1 - \mathbf{I}^i(t)) dt$$

Next, denote the potential service time process which counts exponentially distributed service times as $S_i(t)$, Then

$$D_i(t) = S_i(T_i(t))$$

Hence, as long as interval $[0, \theta]$ is small, write

$$\mathbb{E} \int_0^\theta e^{-\gamma t} dD_i(t) = (1 - \mathbf{I}^e)(1 - \mathbf{I}^i) \mu_i = \tilde{\mu}_i$$

We can now write the costs incurred in $[0, \theta]$

$$J^\theta = \lambda \theta (\mathbf{u} \cdot \mathbf{r} - \mathbf{d} \cdot \beta - f(1 - \sum_i^n \mathbf{u}_i)) - \tilde{\mu} \theta (\mathbf{d} \cdot \psi) - C(q) \theta$$

We now turn to the second term of (5),

$$(1 - \gamma \theta) \left[\lambda \theta J(q + \mathbf{u}) + \sum_i^n \mu J(q - e_i) + (1 - \lambda \theta - \sum_i^n \tilde{\mu}) J(q) \right]$$

multiply all sides by γ and mind that $\theta^2 \simeq 0$.

$$\begin{aligned} \gamma J(q) &= \gamma (\lambda \theta (\mathbf{u} \cdot \mathbf{r} - \mathbf{b} \cdot \beta - f(1 - \sum_i^q \mathbf{s}_i)) - \tilde{\mu} \theta (\mathbf{d} \cdot \mathbf{d}) - C(q) \theta) + \\ \gamma (1 - \gamma \theta) &\left[\lambda \theta J(q + \mathbf{s}) + \sum_i^n \mu \theta J(q - e_i) + (1 - \lambda \theta - \sum_i^n \tilde{\mu} \theta) J(q) \right] = \\ \gamma (\lambda \theta (\mathbf{u} \cdot \mathbf{r} - \mathbf{b} \cdot \beta - f(1 - \sum_i^q \mathbf{s}_i)) - \tilde{\mu} \theta (\mathbf{d} \cdot \mathbf{d}) - C(q) \theta) &+ \\ \left[\lambda \theta \gamma J(q + \mathbf{u}) + \sum_i^n \tilde{\mu} \theta \gamma J(q - e_i) + \right. & \\ \left. (\gamma - \lambda \theta \gamma - \sum_i^n \tilde{\mu} \theta \gamma) J(q) - \gamma \theta J(q) \right] & \end{aligned}$$

See that $\gamma J(q)$ on both sides cancels out and denote $\delta_q = (\sum_i^n \tilde{\mu} + \lambda + \gamma)^{-1}$. Write:

$$\begin{aligned} \gamma \theta \delta_q^{-1} J(q) &= \\ \gamma \theta (\lambda (\mathbf{u} \cdot \mathbf{r} - \mathbf{b} \cdot \beta - f(1 - \sum_i^n \mathbf{u}_i)) - \tilde{\mu} (\mathbf{d} \cdot \psi) - C(q)) &+ \\ \gamma \theta \left[\lambda J(q + \mathbf{s}) + \sum_i^n \tilde{\mu} J(q - e_i) \right] & \end{aligned}$$

Divide all by $\gamma\theta$ and multiply by δ_q

$$J(q) = (\lambda(\mathbf{u} \cdot \mathbf{r} - \mathbf{b} \cdot \beta - f(1 - \sum_i^n \mathbf{u}_i)) - \tilde{\mu}(\mathbf{d} \cdot \psi) - C(q))\delta_q + \left[\lambda J(q + \mathbf{u}) + \sum_i^n \tilde{\mu} J(q - e_i) \right] \delta_q$$

Arrange according to the processes:

$$J(q) = \delta_q \lambda \left[\mathbf{u} \cdot \mathbf{r} - \mathbf{b} \cdot \beta - f(1 - \sum_i^q \mathbf{u}_i) + J(q + \mathbf{u}) \right] + \left[\sum_i^n \tilde{\mu} J(q - e_i) - \tilde{\mu}(\mathbf{d} \cdot \psi) \right] \delta_q - C(q)\delta_q$$

Observe that by definition, $\mathbf{u}, \mathbf{b}, \mathbf{d}$ incorporate (in corresponding feasible cases) the scheduling, build and destroy decisions, respectively. Hence, this equation is identical to (3), where the feasibility is ensured by the corresponding indicators.

B. Proof of proposition 4.1

Proof. Denote the optimal policy π . Clearly, there are more tasks in the system in state b than in state a . Assume that acting as in π brings a to a' and b to b' , while acting otherwise (i.e., scheduling into an existing queue) brings a to a'' and b to b'' .

At arrival, in state q^a it holds $V(q^{a'}) + r - \beta > V(q^{a''}) + r$, that is, it is sub-optimal to allocate the new task into already active queue. (Without loss of generality we assume that there is an active queue which is not full in q^a .) Observe that it holds $q^{b'} \succeq q^{a'}$ and $q^{b''} \succeq q^{a''}$. Substitute this in (4)

$$V(q^{b'}) + \alpha(q^{a'}, q^{b'}) + r - \beta > V(q^{b''}) + \alpha(q^{a''}, q^{b''}) + r$$

To prove that

$$V(q^{b'}) + r - \beta > V(q^{b''}) + r, \quad (7)$$

it is sufficient to show that

$$\alpha(q^{a'}, q^{b'}) \leq \alpha(q^{a''}, q^{b''}) \quad (8)$$

which means the policy π is also optimal for state b . In order to prove that $\alpha(q^{a'}, q^{b'}) < \alpha(q^{a''}, q^{b''})$, we resort to stochastic coupling, similarly to the proof of Lemma 4.1. Define four systems, denoted by a', b', a'', b'' . At time t , the states are denoted by a'_t, b'_t, a''_t, b''_t . Apply the optimal policies for systems b' and a'' . The policies π'_a and π'_b will be suboptimal and such that π''_a (resp. π'_b) will coincide with π'_a (resp. π'_b) in the shortest time possible. We aim to show that $\alpha(q^{a''}, q^{b''}) - \alpha(q^{a'}, q^{b'})$ consists of the partial costs due to the scheduling, rejection and delay, while the costs due to queue deployments, displacements and holding cancel out with exception of an arbitrarily negligible component. Then,

by the delay structure in (1), the lemma will follow from the following.

$$\begin{aligned} \alpha(q^{a''}, q^{b''}) &= V(q^{a''}) - V(q^{b''}) > \\ V^{\pi_{a''}}(q^{a''}) - V(q^{b''}) &\geq V(q^{a'}) - V^{\pi_{b'}}(q^{b'}) > \\ V(q^{a'}) - V(q^{b'}) &= \alpha(q^{a'}, q^{b'}) \end{aligned} \quad (9)$$

Next, we specify the suboptimal policies π'_a and π'_b . For simplicity, we assume that the same queues were selected for scheduling in a' and a'' ; and b' and b'' . Denote the queue which has more tasks in a'' , (resp. in b'') by $q_j^{a''}$, (resp. $q_j^{b''}$). Denote the newly activated queue which has exactly one task in a' , (resp. in b') by $q_k^{a'}$, (resp. $q_k^{b'}$). To this end, $\pi_{b'}$ mimics $\pi_{b''}$ in all actions, with exceptions of the following events which happen only once. In the case b'' activates queue k , and schedules into it, b' , if $q_k^{b'}$ was active, will schedule into queue j . For this purpose, in the case $q_j^{b''}$ fills up and rejects - the task will be rejected in b' as well, hence $q_j^{b'}$ will have one place left. Finally, in order to prevent the event where $q_k^{b'}$ fills up before it ever becomes empty and before $q_k^{b''}$ is activated at first time, $\pi_{b'}$ will schedule into $q_j^{b'}$ in this case. Note that these actions will make the systems equal. The system b' and b'' start to act identically (optimally) once all these systems' queues become equal.

Likewise, π'_a mimics π'_a in all actions, with exceptions of the following events which happen only once. In the case a' schedules into queue j , then a'' activates queue k and schedules into it, provided $q_k^{a'}$ was not destroyed before that. This prevents having a future rejection from $q_j^{a''}$ while not rejecting at $q_j^{a'}$ at the same time. In order to prevent the rejection event in a' but not in a'' during filling of the queue $q_k^{a'}$, once $q_j^{a'}$ fills up and rejects - the task will be rejected in a'' as well.

Note that in the case queue k gets empty before any tasks are scheduled, the event happens in both systems a' and b' , since they are stochastically coupled. Hence the corresponding cost difference to this event cancels out. We also assumed that by identity of all queues, the scheduling into active queues is balanced. Denote the time when both b' equals to b'' and a' equals to a'' by τ . Write

$$V^{\pi_{a''}}(q^{a''}) - V(q^{a'}) = J_p^{\pi_{a''}}(q^{a''}) - J_p(q^{a'}) - J_\tau(a'),$$

where $J_\tau(a')$ is a nonnegative component associated with having the additional queue till the time τ (or before). Similarly,

$$V^{\pi_{b'}}(q^{b''}) - V(q^{b'}) = J_p^{\pi_{b'}}(q^{b''}) - J_p(q^{b'}) - J_\tau(b'),$$

where $J_\tau(b')$ is a nonnegative component associated with having the additional queue till the time τ (or before). We use the delay structure in (1) to write

$$J_p^{\pi_{b'}}(q^{b''}) - J_p(q^{b'}) > J_p^{\pi_{a''}}(q^{a''}) - J_p(q^{a'}) \quad (10)$$

It is left to assess the impact of $J_p(q^{a'})$ and $J_\tau(b')$. Observe that these are average costs associated with the same events,

but differently discounted. In order to avoid complex analysis, we take the discount factor close enough to 1, such that the difference between the two costs will be negligible if compared to components in (10). By substitution in (9), the proposition follows. ■

REFERENCES

- [1] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow World Congress*, 2012, pp. 22–24.
- [2] J. F. Riera, E. Escalona, J. Batallé, E. Grasa, and J. A. García-Espín, "Virtual network function scheduling: Concept and challenges," in *Smart Communications in Network Technologies (SaCoNeT), 2014 International Conference on*. IEEE, pp. 1–5.
- [3] M. Schöller, M. Stiemerling, A. Ripke, and R. Bless, "Resilient deployment of virtual network functions," in *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE, pp. 208–214.
- [4] J. Costa-Requena, J. L. Santos, V. F. Guasch, K. Ahokas, G. Premsankar, S. Luukkainen, O. L. Pérez, M. U. Itzazelaia, I. Ahmad, M. Liyanage *et al.*, "Sdn and nfv integration in generalized mobile network architecture," in *Networks and Communications (EuCNC), 2015 European Conference on*. IEEE, pp. 154–158.
- [5] J. Mace, P. Bodik, M. Musuvathi, R. Fonseca, and K. Varadarajan, "2dfq: Two-dimensional fair queuing for multi-tenant cloud services," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pp. 144–159.
- [6] M. Shifrin, R. Atar, and I. Cidon, "Optimal scheduling in the hybrid-cloud," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pp. 51–59.
- [7] M. Yoshida, W. Shen, T. Kawabata, K. Minato, and W. Imajuku, "Morsa: A multi-objective resource scheduling algorithm for nfv infrastructure," in *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*. IEEE, pp. 1–6.
- [8] L. Qu, C. Assi, and K. Shaban, "Network function virtualization scheduling with transmission delay optimization," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pp. 638–644.
- [9] J. A. Van Mieghem, "Price and service discrimination in queuing systems: Incentive compatibility of $gc \mu$ scheduling," *Management Science*, vol. 46, no. 9, pp. 1249–1267, 2000.
- [10] M. J. Neely, E. Modiano, and C.-P. Li, "Fairness and optimal stochastic control for heterogeneous networks," *IEEE/ACM Transactions On Networking*, vol. 16, no. 2, pp. 396–409.
- [11] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1997, pp. 1–34.
- [12] A. Ganti, E. Modiano, and J. N. Tsitsiklis, "Optimal transmission scheduling in symmetric communication models with intermittent connectivity," *IEEE Transactions on Information Theory*, vol. 53, no. 3, pp. 998–1008, 2007.
- [13] J. M. Harrison and A. Zeevi, "Dynamic scheduling of a multiclass queue in the halfin-whitt heavy traffic regime," *Operations Research*, vol. 52, no. 2, pp. 243–257, 2004.
- [14] A. A. Puhalskii, M. I. Reiman *et al.*, "The multiclass $gi/ph/n$ queue in the halfin-whitt regime," *Advances in Applied Probability*, vol. 32, no. 2, pp. 564–595, 2000.