Cost minimization of network services with buffer and end-to-end deadline constraints

Victor Millnert
Lund University, Sweden

Enrico Bini
Scuola Superiore Sant'Anna, Pisa, Italy

Johan Eker
Ericsson Research, Sweden
Lund University, Sweden

ABSTRACT

Cloud computing technology provides the means to share physical resources among multiple users and data center tenants by exposing them as virtual resources. There is a strong industrial drive to use similar technology and concepts to provide timing sensitive services. One such is virtual networking services, so called services chains, which consist of several interconnected virtual network functions. This allows for the capacity to be scaled up and down by adding or removing virtual resources. In this work, we develop a model of a service chain and pose the dynamic allocation of resources as an optimization problem. We design and present a set of strategies to allot virtual network nodes in an optimal fashion subject to latency and buffer constraints.

1. INTRODUCTION

Over the last years, cloud computing has swiftly transformed the IT infrastructure landscape, leading to large cost-savings for deployment of a wide range of IT applications. Some main characteristics of cloud computing are resource pooling, elasticity, and metering. Physical resources such as compute nodes, storage nodes, and network fabrics are shared among tenants. Virtual resource elasticity brings the ability to dynamically change the amount of allocated resources, for example as a function of workload or cost. Resource usage is metered and in most pricing models the tenant only pays for the allocated capacity.

While cloud technology initially was mostly used for IT applications, e.g. web servers, databases, etc., it is rapidly finding its way into new domains. One such domain is processing of network packages. Today network services are packaged as physical appliances that are connected together using physical network. Network services consist of interconnected network functions (NF). Examples of network functions are firewalls, deep packet inspections, transcoding, etc. A recent initiative from the standardisation body ETSI (European Telecommunications Standards Institute) addresses the standardisation of virtual network services under the name Network Functions Virtualisation (NFV) [1]. The expected benefits from this are, among others, better hardware utilisation and more flexibility, which translate into reduced capital and operating expenses (CAPEX and OPEX).

A number of interesting use cases are found in [2], and in this paper we are investigating the one referred to as Virtual Network Functions Forwarding Graphs, see Figure 1.

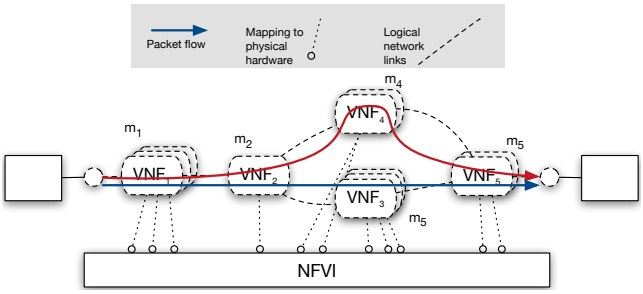

Figure 1: Several virtual networking functions (VNF) are connected together to provide a set of services. Packet flow through a specific path the VNFs (a virtual forwarding graph). The VNFs are mapped onto physical hardware referred to as NFVI.

We investigate the allocation of virtual resources to a given packet flow, i.e. what is the most cost efficient way to allocate VNFs with a given capacity that still provide a network service within a given latency bound? The distilled problem is illustrated as the packet flows in Figure 1. The forwarding graph is implemented as a chain of virtual network nodes, also known as a service chains. To ensure that the capacity of a service chain matches the time-varying load, the number of instances m_i of each individual network function VNF_i may be scaled up or down.

The contribution of the paper is

- a mathematical model of the virtual resources supporting the packet flows in Figure 1,
- the set-up of an optimization problem for controlling the number of machines needed by each function in the service chain,
- solution of the optimization-problem leading to a control-scheme of the number of machines needed to guarantee that the end-to-end deadline is met for incoming packets under a constant input flow.

Related work

There are a number of well known and established resource management frameworks for data centers, but few of them

explicitly address the issue of latency. Sparrow [3] presents an approach for scheduling a large number of parallel jobs with short deadlines. The problem domain is different compared to our work in that we focus on sequential rather than parallel jobs. Chronos [4] focuses on reducing latency on the communication stack. RT-OpenStack [5] adds real-time performance to OpenStack by usage of a real-time hypervisor and a timing-aware VM-to-host mapping.

The enforcement of an end-to-end (E2E) deadline of a sequence of jobs to be executed through a sequence of computing elements was addressed by several works, possibly under different terminologies. In the holistic analysis [6, 7, 8] the schedulability analysis is performed locally. At global level the local response times are transformed into jitter or offset constraints for the subsequent tasks.

A second approach to guarantee an E2E deadline is to split a constraint into several local deadline constraints. While this approach avoids the iteration of the analysis, it requires an effective splitting method. Di Natale and Stankovic [9] proposed to split the E2E deadline proportionally to the local computation time or to divide equally the slack time. Later, Jiang [10] used time slices to decouple the schedulability analysis of each node, reducing the complexity of the analysis. Such an approach improves the robustness of the schedule, and allows to analyse each pipeline in isolation. Serreli et al. [11, 12] proposed to assign local deadlines to minimize a linear upper bound of the resulting local demand bound functions. More recently, Hong et al [13] formulated the local deadline assignment problem as a Mixed-Integer Linear Program (MILP) with the goal of maximizing the slack time. After local deadlines are assigned, the processor demand criterion was used to analyze distributed real-time pipelines [14, 12].

In all the mentioned works, jobs have non-negligible execution times. Hence, their delay is caused by the preemption experienced at each function. In our context, which is scheduling of virtual network services, jobs are executed non-preemptively and in FIFO order. Hence, the impact of the local computation onto the E2E delay of a request is minor compared to the queueing delay. This type of delay is intensively investigated in the networking community in the broad area *queuing systems* [15]. In this area, Henriksson et al. [16] proposed a feedforward/feedback controller to adjust the processing speed to match a given delay target.

Most of the works in queuing theory assumes a stochastic (usually markovian) model of job arrivals and service times. A solid contribution to the theory of deterministic queuing systems is due to Baccelli et al. [17], Cruz [18], and Parekh & Gallager [19]. These results built the foundation for the *network calculus* [20], later applied to real-time systems in the *real-time calculus* [21]. The advantage of network/real-time calculus is that, together with an analysis of the E2E delays, the sizes of the queues are also modelled. As in the cloud computing scenario the impact of the queue is very relevant since that is part of the resource usage which we aim to minimize, hence we follow this type of modeling.

2. PROBLEM FORMULATION

Section 1. We consider a *service-chain* consisting of n *functions* $F_1, \ldots, F_n$, as illustrated in Figure 2. Packets are flowing through the service-chain and they must be processed by each function in the chain within some end-to-end deadline, denoted by $D^{\max}$. A *fluid model* is used to approximate the packet flow and at time t there are $r_{i-1}(t) \in \mathbb{R}^+$ packets per second (pps) entering the i'th function and the the *cumulative arrived requests* for this function is

$$R_{i-1}(t) = \int_0^t r_{i-1}(\tau)\, d\tau. \tag{1}$$

In a recent benchmarking study it was shown that a typical virtual machine can process around 0.1–2.8 million packets per second, [22]. Hence, in this work the number of packets flowing through the functions is assumed to be in the order of millions of packets per second, supporting the use of a fluid model.

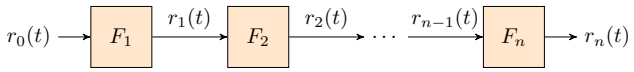

Figure 2: Illustration of the service-chain.

2.1 Service model

As illustrated in Figure 3, the incoming requests to function F_i are stored in the queue and then processed once it reaches the head of the queue. At time t there are $m_i(t) \in \mathbb{Z}^+$ machines ready to serve the requests, each with a *nominal speed* of $\bar{s}_i \in \mathbb{R}^+$ (note that this nominal speed might differ between different functions in the service chain, i.e. it does not in general hold that $\bar{s}_i = \bar{s}_j$ for $i \neq j$). The *maximum speed* that function F_i can process requests at is thus $m_i(t)\bar{s}_i$. The rate by which F_i is actually processing requests at time t is denoted $s_i(t) \in \mathbb{R}^+$. The *cumulative served requests* is defined as

$$S_i(t) = \int_0^t s_i(\tau)\, d\tau. \tag{2}$$

At time t the number of requests stored in the queue is defined as the *queue length* $q_i(t) \in \mathbb{R}^+$:

$$q_i(t) = \int_0^t (r_{i-1}(\tau) - s_i(\tau))d\tau = R_{i-1}(t) - S_i(t). \tag{3}$$

Each function has a fixed *maximum-queue capacity* $q_i^{\max} \in \mathbb{R}^+$, representing the largest number of requests that can be stored at the function F_i.

The *queueing delay*, depends on the status of the queue as well as on the service rate. We denote by $D_{i,j}(t)$ the time

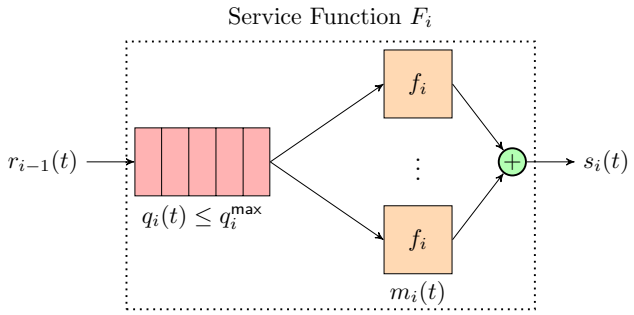

Figure 3: Illustration of the structure and different entities of the service chain.

taken by a request from when it enters function F_i to when it exits F_j, with $j \geq i$, where t is the time when the request exits function F_j:

$$D_{i,j}(t) = \inf\{\tau \geq 0 : R_{i-1}(t-\tau) \leq S_j(t)\}.$$

The *maximum queueing delay* then is $\hat{D}_{i,j} = \max_{t\geq 0} D_{i,j}(t)$. The requirement that a requests meets it end-to-end deadline is $\hat{D}_{1,n} \leq D^{\mathsf{max}}$.

To control the queueing delay, it is necessary to control the service rate of the function. Therefore, we assume that it is possible to change the maximum service-rate of a function by changing the number of machines that are on, i.e. changing $m_i(t)$. However, turning on a machine takes Δ_i^{on} time units, and turning off a machine takes Δ_i^{off} time units. Together they account for a *time delay*, $\Delta_i = \Delta_i^{\mathsf{on}} + \Delta_i^{\mathsf{off}}$, associated with turning on/off a machine.

In 2012 Google profiled where the latency in a data center occurred, [4]. They showed that less than 1% ($\approx 1\mu s$) of the latency occurred was due to the propagation in the network fabric. The other 99% ($\approx 85\mu s$) occurred somewhere in the kernel, the switches, the memory, or the application. Since it is difficult to say exactly which of this 99% is due to processing, or queueing, we make the abstraction of considering queueing delay and processing delay together, simply as queueing delay. Furthermore, we assume that no request is lost in the communication links, and that there is no propagation delay. Hence the concatenation of the functions F_1 through F_n implies that the input of function F_i is exactly the output of function F_{i-1}, for $i = 2, \ldots, n$, as illustrated in Figure 2.

2.2 Cost model

To be able to provide guarantees about the behaviour of the service chain, it is necessary to make *hard reservations* of the resources needed by each function in the chain. This means that when a certain resource is reserved, it is guaranteed to be available for utilization. Reserving this resource results in a cost, and due to the hard reservation, the cost does not dependent on the actual utilisation, but only on the resource reserved.

The *computation cost* per time-unit per machine is denoted $\jmath_i^{\mathsf{c}}$, and can be seen as the cost for the CPU-cycles needed by one machine in F_i. This cost will also occur during the time-delay Δ_i. Without being too conservative, this time-delay can be assumed to occur only when a machine is started. The *average computing cost* per time-unit for the whole function F_i is then

$$J_i^{\mathsf{c}}(m_i(t)) = \lim_{t\to\infty} \frac{\jmath_i^{\mathsf{c}}}{t} \int_0^t m_i(s) + \Delta_i \cdot (\partial_- m_i(s))_+ ds \qquad (4)$$

where $(x)_+ = \max(x, 0)$, and $\partial_- m_i(t)$ is the left-limit of $m_i(t)$:

$$\partial_- m_i(t) = \lim_{a\to t^-} \frac{m_i(t) - m_i(a)}{t-a},$$

that is, a sequence of Dirac's deltas at all points where the number of machines changes. This means that the value of the left-limit of $m_i(t)$ is only adding to the computation-cost whenever it is positive, i.e. when a machine is switched on.

The *queue cost* per time-unit per space for a request is denoted $\jmath_i^{\mathsf{q}}$. This can be seen as the cost that comes from the fact that physical storage needs to be reserved such that a queue can be hosted on it, normally this would correspond to the RAM of the network-card. Reserving the capacity of q_i^{max} would thus result in a cost per time-unit of

$$J_i^{\mathsf{q}}(q_i^{\mathsf{max}}) = \jmath_i^{\mathsf{q}} q_i^{\mathsf{max}}. \qquad (5)$$

2.3 Problem definition

The aim of this paper is to control the number $m_i(t)$ of machines running in function F_i, such that the total average cost is minimized, while the E2E constraint D^{max} is not violated and the maximum queue sizes q_i^{max} are not exceeded. This can be posed as the following problem:

$$\begin{aligned} \text{minimize } & J = \sum_{i=1}^{n} J_i^{\mathsf{c}}(m_i(t)) + J_i^{\mathsf{q}}(q_i^{\mathsf{max}}) \\ \text{subject to } & \hat{D}_{1,n} \leq D^{\mathsf{max}} \\ & q_i(t) \leq q_i^{\mathsf{max}}, \quad \forall t \geq 0, \quad i = 1, 2, \ldots, n \end{aligned} \qquad (6)$$

with J_i^{c} and J_i^{q} as in (4) and (5), respectively. In this paper the optimization problem (6) will be solved for a service-chain fed with a *constant incoming rate* r.

A valid lower bound J^{lb} to the cost achieved by any feasible solution of (6) is found by assuming that all functions are capable of providing exactly a service rate r equal to the input rate. This is possible by running a fractional number of machines $r/\bar{s}_i$ at function F_i. In such an ideal case, buffers can be of zero size ($\forall i,\ q_i^{\mathsf{max}} = 0$), and there is no queueing delay ($\hat{D}_{1,n} = 0$) since service and the arrival rates are the same at all functions. Hence, the lower bound to the cost is

$$J^{\mathsf{lb}} = \sum_{i=1}^{n} \jmath_i^{\mathsf{c}} \frac{r}{\bar{s}_i}. \qquad (7)$$

Such a lower bound will be used to compare the quality of the solution found later on.

In Section 3 we make a general consideration about the on/off scheme of each machine in presence of a constant input rate r. Later in Section 4, the optimal design problem of (6) is solved.

3. MACHINE SWITCHING SCHEME

In presence of an incoming flow of requests at a constant rate $r_0(t) = r$, a number

$$\bar{m}_i = \left\lfloor \frac{r}{\bar{s}_i} \right\rfloor \qquad (8)$$

of machines running in function F_i must always stay on. To match the incoming rate r, in addition to the $\bar{m}_i$ machines always on, another machine must be on for some time in order to process a request rate of $\bar{s}_i \rho_i$ where ρ_i is the *normalized residual request* rate:

$$\rho_i = {}^{r}/_{\bar{s}_i} - \bar{m}_i, \qquad (9)$$

where $\rho_i \in [0, 1)$.

In our scheme, the extra machine is switched on at a *desired on-time* t_i^{on}:

- off → on: function F_i switches on the additional machine when the time t exceeds t_i^{on}.

Since the additional machine does not need to always be on, it could be switched off after some time. The off-switching is also based on a time-condition, the *desired stop-time* t_i^{off}, i.e. the time-instance that the machine should be switched off, and is given by:

$$t_i^{\mathsf{off}} = t_i^{\mathsf{on}} + T_i^{\mathsf{on}}.$$

where T_i^{on} is the duration that the machine should be on for, and something that needs to be found. The off-switching is then triggered in the following way:

- on → off: function F_i switches off the additional machine when the time t exceeds t_i^{off}.

Note that this control-scheme, in addition with the constant input, result in the extra machine being switched on/off *periodically*, with a period T_i. We thus assume that the extra machine can process requests for a time T_i^{on} every period T_i. The time during each period where the machine is not processing any requests is denoted $T_i^{\mathsf{off}} = T_i - T_i^{\mathsf{on}}$. Notice, however, that the actual time the extra machine is consuming power is $T_i^{\mathsf{on}} + \Delta_i$ due to the time-delay for starting a new machine.

In the presence of a constant input, it is straight-forward to find the necessary on-time during each period—in order for the additional machine to provide the residual processing capacity of $r - \bar{m}_i\bar{s}_i$, its on-time T_i^{on} must be such that

$$T_i^{\mathsf{on}}\bar{s}_i = T_i(r - \bar{m}_i\bar{s}_i),$$

which implies

$$T_i^{\mathsf{on}} = T_i\rho_i, \quad T_i^{\mathsf{off}} = T_i - T_i^{\mathsf{on}} = T_i(1-\rho_i). \tag{10}$$

With each additional machine being switched on/off periodically, it is also straightforward to find the computation cost for each function. If $\bar{m}_i + 1$ machines are on for a time T_i^{on}, and only $\bar{m}_i$ machines are on for a time T_i^{off}, then the cost J_i^{c} of (4) becomes

$$J_i^{\mathsf{c}} = \jmath_i^{\mathsf{c}}\left(\frac{T_i^{\mathsf{on}} + \Delta_i}{T_i} + \bar{m}_i\right) = \jmath_i^{\mathsf{c}}\left(\bar{m}_i + \rho_i + \frac{\Delta_i}{T_i}\right) \tag{11}$$

if $T_i^{\mathsf{off}} \geq \Delta_i$. If instead $T_i^{\mathsf{off}} < \Delta_i$, that is if

$$T_i < \overline{T}_i := \frac{\Delta_i}{1-\rho_i}, \tag{12}$$

then there is no time to switch the additional machine off and then on again before the new period start. Hence, we keep the last machine on, even if it is not processing packets, and the computing cost becomes

$$J_i^{\mathsf{c}} = \jmath_i^{\mathsf{c}}\left(\bar{m}_i + \rho_i + \frac{T_i^{\mathsf{off}}}{T_i}\right) = \jmath_i^{\mathsf{c}}(\bar{m}_i + 1). \tag{13}$$

Next, using this control-scheme, the optimization problem of (6) will be studied and solved under the assumption that every function will switch on/off its additional machine with the same period, T.

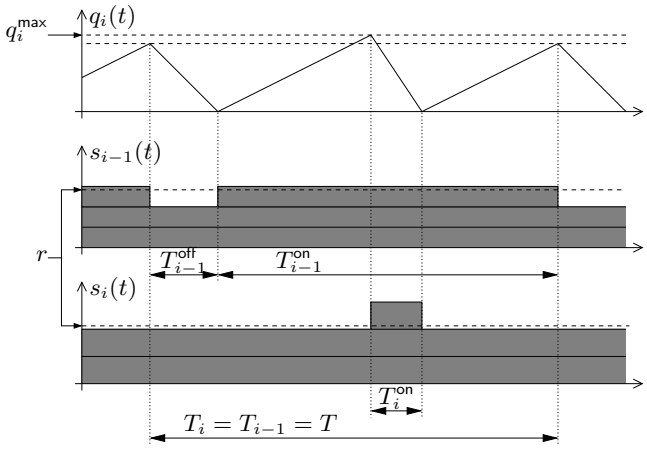

Figure 4: Example of an on/off-switching scheme when function F_i and F_{i-1} switch on/off their additional machine with the same period and how it affects the queue-size $q_i(t)$ of the functions.For this example: $r = 17$, $\bar{s}_{i-1} = 6$, $\bar{s}_i = 8$, $T = 120$, $T_{i-1}^{\mathsf{on}} = 100$, $T_i^{\mathsf{on}} = 15$, $q_i^{\max} = 90$.

4. DESIGN OF MACHINE-SWITCHING PERIOD

In this section we solve the optimization problem (6) under the assumption of a constant input. In order to somewhat reduce the complexity of the solution we also make the assumption of letting every function switch its additional machine on/off with the same period, $T_i = T$. The common period T of the schedule, by which every function switches its additional machine on/off, is the only design variable in the optimization problem (6). In Lemma 1 and Lemma 2 below, the maximum queue size $q_i^{\max}$ of any function F_i and the end-to-end delay $\hat{D}_{1,n}$ are both shown to be proportional to the switching period T. The intuition behind this fact is that the longer the period T is, the longer a function will have to wait with the additional machine being off, before turning it on again. During this interval of time, each function is accumulating work and consequently both the maximum queue-size and the delay grows with T.

Figure 4 illustrate how two functions in a service-chain, F_i and F_{i-1}, switch on/off their additional machine with the same period T. However, one should note that they are not on for the same duration. In this example, the input rate to the service-chain is $r = 17$, the nominal service-rate of the first and second function is $\bar{s}_{i-1} = 6$ and $\bar{s}_i = 8$ respectively. The machine-switching period is $T = 120$, and the on-time for the two additional machines are $T_{i-1}^{\mathsf{on}} = 100$ and $T_i^{\mathsf{on}} = 15$ respectively. The maximum queue-size needed for the second machine is $q_i^{\max} = 90$.

In order to solve the optimization problem one need two ingredients. The first ingredient is the expression for the maximum queue-size needed for a given period T:

Lemma 1. *With a constant input rate $r_0(t) = r$, along with all functions switching on/off their additional machine with a common period T, the maximum queue size $q_i^{\max}$ at function F_i is*

$$q_i^{\max} = T \times \alpha_i, \tag{14}$$

where

$$\alpha_i = \max\big\{\rho_i\big(\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})\big), \\ (1-\rho_{i-1})(\bar{s}_{i-1}\rho_{i-1} - \bar{s}_i\rho_i), \\ \rho_{i-1}\big(\bar{s}_{i-1}(1-\rho_{i-1}) - \bar{s}_i(1-\rho_i)\big), \\ (1-\rho_i)(\bar{s}_i\rho_i - \bar{s}_{i-1}\rho_{i-1})\big\},$$

with ρ_i as defined in (9), and T being the period of the switching scheme, common to all functions.

PROOF. Due to limited space the proof is shown in a technical report published at Lund University Publications, [23]. [1] □

The expression of $q_i^{\max}$ in Eq. (14) suggests a property that is condensed in the next Corollary.

Corollary 1. *The maximum queue qize $q_i^{\max}$ at any function F_i is bounded, regardless of the rate r of the input.*

PROOF. From the definition of ρ_i in Eq. (9), it always holds that $\rho_i \in [0, 1)$. Hence, from the expression of (14), it follows that $q_i^{\max}$ is always bounded. □

The second ingredient needed to solve the optimal design problem is the expression of how the end-to-end delay relate to the switching period T.

Lemma 2. *With a constant input rate, $r_0(t) = r$, the longest end-to-end delay $\hat{D}_{i,n}$ for any request passing through functions F_1 thru F_n is*

$$\hat{D}_{1,n} = T \times \sum_{i=1}^{n} \delta_i. \tag{15}$$

with δ_i being an opportune constant that depends on r, $\bar{s}_i$, and $\bar{s}_{i-1}$.

PROOF. Due to limited space the proof is shown in a technical report published at Lund University Publications, [23]. □

Solution to the optimization problem

With these hypothesis, the cost function of the optimization problem (6) becomes

$$J(T) = aT + \sum_{i:T<\overline{T}_i} j_i^{\mathsf{c}}(1-\rho_i) + \sum_{i:T\geq\overline{T}_i} j_i^{\mathsf{c}}\frac{\Delta_i}{T} + J^{\mathsf{lb}}, \tag{16}$$

where J^{lb} is the lower bound given by (7) and $a = \sum_{i=1}^{n} j_i^{\mathsf{q}}\alpha_i$, where α_i is given by Lemma 1. Furthermore, $\overline{T}_i$ (defined in (12)) represents the value of the period below which it is not feasible to switch the additional machine off and then on again ($T < \overline{T}_i \;\Leftrightarrow\; T_i^{\mathsf{off}} < \Delta_i$). In fact, $\forall i$ with $T < \overline{T}_i$ we pay the full cost of having $\bar{m}_i + 1$ machines always on.

The deadline constraint in (6), can be simply written as

$$T \leq c := \frac{D^{\max}}{\sum_{i=1}^{n}\delta_i},$$

with δ_i given in Lemma 2.

The cost $J(T)$ of (16) is a continuous function of one variable T. It has to be minimized over the closed interval $[0, c]$. Hence, by the Weierstaß's extreme-value theorem, it has a minimum. To find this minimum, we just check all (finite) points at which the cost is not differentiable and the ones where the derivative is equal to zero. Let us define all points in $[0, c]$ in which $J(T)$ is not differentiable:

$$\mathcal{C} = \{\overline{T}_i : \overline{T}_i < c\} \cup \{0\} \cup \{c\}. \tag{17}$$

We denote by $p = |\mathcal{C}| \leq n + 2$ the number of points in $\mathcal{C}$. Also, we denote by $c_k \in \mathcal{C}$ the points in $\mathcal{C}$ and we assume they are ordered increasingly $c_1 < c_2 < \ldots < c_p$. Since the cost $J(T)$ is differentiable over the open interval (c_k, c_{k+1}), the minimum may also occur at an interior point of (c_k, c_{k+1}) with derivative equal to zero. Let us denote by $\mathcal{C}^*$ the set of all interior points of (c_k, c_{k+1}) with derivative of $J(T)$ equal to zero, that is

$$\mathcal{C}^* = \{c_k^* : k = 1, \ldots, p-1,\ c_k < c_k^* < c_{k+1}\} \tag{18}$$

with

$$c_k^* = \sqrt{\frac{\sum_{i:\overline{T}_i<c_{k+1}} j_i^{\mathsf{c}}\Delta_i}{a}}.$$

Then, the optimal period is given by

$$T^* = \arg\min_{T\in\mathcal{C}\cup\mathcal{C}^*}\{J(T)\}. \tag{19}$$

Next, we illustrate an example of how to use this to find a solution to the design problem (6).

Example. We use an example to illustrate the solution of the optimization problem of a service chain with two functions. The input rate of the service-chain is $r_0(t) = r = 17$. Every request has an E2E-deadline of $D^{\max} = 0.02$. The parameters of the two functions are reported in Table 1.

i	$\bar{s}_i$	j_i^{c}	j_i^{q}	Δ_i
1	6	6	0.5	0.01
2	8	8	0.5	0.01

Table 1: Parameters of the example.

The input $r_0(t) = r$ can be seen as dummy function F_0 preceding F_1, with $\bar{s}_0 = r$, $\bar{m}_0 = 1$, and $\rho_0 = 0$. From (8) and (9) it follows that $\bar{m}_1 = \bar{m}_2 = 2$, and $\rho_1 = \frac{5}{6}$, $\rho_2 = \frac{1}{8}$, implying that both functions must always keep two machines on, and then periodically switch a third one on/off. This leads to $\overline{T}_1 = 60.0 \times 10^{-3}$ and $\overline{T}_2 = 11.4 \times 10^{-3}$, where $\overline{T}_i$ is the threshold period for function F_i, as defined in (12). From Lemma 1 it follows that the parameter a of the cost function (16) is $a = 0.792$, while from Lemma 2 the parameters δ_i determining the queuing delay introduced by each function, are $\delta_1 = 49.0 \times 10^{-3}$ and $\delta_2 = 22.1 \times 10^{-3}$, which in turn leads to

$$c = \frac{D^{\max}}{\delta_1 + \delta_2} = \frac{0.02}{71.1 \times 10^{-3}} = 281 \times 10^{-3}.$$

Since $\overline{T}_2 < \overline{T}_1 < c$, the set $\mathcal{C}$ of (17) containing the boundary is

$$\mathcal{C} = \{0, \underbrace{0.00114}_{\overline{T}_2}, \underbrace{0.060}_{\overline{T}_1}, \underbrace{0.281}_{c}\}.$$

[1] https://lup.lub.lu.se/search/publication/8c7b837e-bca3-4375-bb9d-28ce6bbc889a

To compute the set $\mathcal{C}^*$ of interior points with derivative equal to zero defined in (18), which is needed to compute the period with minimum cost from (19), we must check all intervals with boundaries at two consecutive points in $\mathcal{C}$. In the interval $(0, \overline{T}_2)$ the derivative of J is never zero. When checking the interval $(\overline{T}_2, \overline{T}_1)$, the derivative is zero at

$$c_1^* = \sqrt{\frac{j_2^c \Delta_2}{a}} = 0.318,$$

which, however, falls outside the interval. Finally, when checking the interval $(\overline{T}_1, c)$ the derivative is zero at

$$c_2^* = \sqrt{\frac{j_1^c \Delta_1 + j_2^c \Delta_2}{a}} = 0.421 > c = 0.281.$$

Hence, the set of points with derivative equal to zero is $\mathcal{C}^* = \emptyset$. By inspecting the cost at points in $\mathcal{C}$ we find that the minimum occurs at $T^* = c = 0.281$, with cost $J(T^*) = 34.7$. To conclude the example we show the state-space trajectory for the two queues in Figure 5. Again, it should be noted that this example is meant to illustrate how one can use the design methodology of this section in order to find the best period T. In a real setting the incoming traffic will likely be around millions of requests per second, [24].

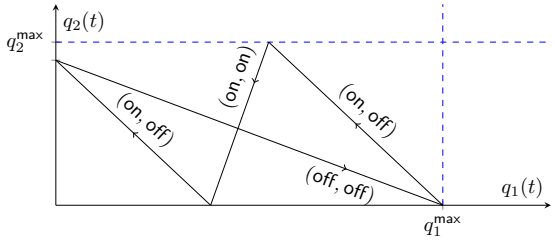

Figure 5: State-space trajectory for the example in Section 4. (on, off) correspond to F_1 having its additional machine on, while F_2 has its extra machine off.

5. SUMMARY

In this paper we have developed a general mathematical model for a service-chain residing in a Cloud environment. This model includes an input model, a service model, and a cost model. The input-model defines the input-stream of requests to each NFV along with end-to-end deadlines for the requests, meaning that they have to pass through the service-chain before this deadline. In the service-model, we define an abstract model of a NFV, in which requests are processed by a number of machines inside the service function. It is assumed that each function can change the number of machines that are up and running, but doing so is assumed to take some time. The cost-model defines the cost for allocating compute- and storage capacity, and naturally leads to the optimization problem of how to allocate the resources. We analyze the case with a constant input-stream of requests and derive control-strategies for this. This is a simplified case it will constitute the foundation of adaptive schemes to time-varying requests in the future.

We plan to extend this work by allowing for a dynamic input as well as uncertainties in the true performance of the machines running in the functions, leading to the need of using a more advanced feedback loop to guarantee the desired performance.

Acknowledgements. The authors would like to thank Karl-Erik Årzén and Bengt Lindoff for the useful comments on early versions of this paper.

Source code. The source code used to compute the solution of the example in Section 4 can be found on Github at https://github.com/vmillnert/REACTION-source-code.

6. REFERENCES

[1] ETSI, "Network Functions Virtualization (NFV)," https://portal.etsi.org/nfv/nfv_white_paper.pdf, October 2012.

[2] ——, "Network Functions Virtualization (NFV); Use Cases," October 2013.

[3] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 69–84.

[4] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat, "Chronos: Predictable low latency for data center applications," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 9:1–9:14. [Online]. Available: http://doi.acm.org/10.1145/2391229.2391238

[5] S. Xi, C. Li, C. Lu, C. D. Gill, M. Xu, L. T. Phan, I. Lee, and O. Sokolsky, "RT-Open Stack: CPU resource management for real-time cloud computing," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 179–186.

[6] K. W. Tindell, A. Burns, and A. Wellings, "An extendible approach for analysing fixed priority hard real-time tasks," *Journal of Real Time Systems*, vol. 6, no. 2, pp. 133–152, Mar. 1994.

[7] J. Palencia and M. G. Harbour, "Offset-based response time analysis of distributed systems scheduled under EDF," in *15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2003.

[8] R. Pellizzoni and G. Lipari, "Holistic analysis of asynchronous real-time transactions with earliest deadline scheduling," *Journal of Computer and System Sciences*, vol. 73, no. 2, pp. 186–206, Mar. 2007.

[9] M. Di Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proceedings of the 15-th IEEE Real-Time Systems Symposium*, Dec. 1994, pp. 215–227.

[10] S. Jiang, "A decoupled scheduling approach for distributed real-time embedded automotive systems," in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006, pp. 191–198.

[11] N. Serreli, G. Lipari, and E. Bini, "Deadline assignment for component-based analysis of real-time transactions," in *2nd Workshop on Compositional Real-Time Systems*, Washington, DC, USA, Dec. 2009.

[12] ——, "The demand bound function interface of distributed sporadic pipelines of tasks scheduled by EDF," in *Proceedings of the 22-nd Euromicro Conference on Real-Time Systems*, Bruxelles, Belgium, July 2010.

[13] S. Hong, T. Chantem, and X. S. Hu, "Local-deadline assignment for distributed real-time systems," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 1983–1997, July 2015.

[14] A. Rahni, E. Grolleau, and M. Richard, "Feasibility analysis of non-concrete real-time transactions with edf assignment priority," in *Proceedings of the 16-th conference on Real-Time and Network Systems*, Rennes, France, Oct. 2008, pp. 109–117.

[15] L. Kleinrock, *Queueing Systems*. John Wiley & Sons, 1975.

[16] D. Henriksson, Y. Lu, and T. Abdelzaher, "Improved prediction for web server delay control," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, June 2004, pp. 61–68.

[17] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and linearity*. Wiley New York, 1992, vol. 3.

[18] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, Jan. 1991.

[19] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[20] J.-Y. Le Boudec and P. Thiran, *Network Calculus: a theory of deterministic queuing systems for the internet*, ser. Lecture Notes in Computer Science. Springer, 2001, vol. 2050.

[21] S. Chakraborty and L. Thiele, "A new task model for streaming applications and its schedulability analysis," in *Design, Automation and Test in Europe Conference and Exposition*, Mar. 2005, pp. 486–491.

[22] R. Bonafiglia, I. Cerrato, F. Ciaccia, M. Nemirovsky, and F. Risso, "Assessing the performance of virtualization technologies for nfv: a preliminary benchmarking," in *2015 Fourth European Workshop on Software Defined Networks*. IEEE, 2015, pp. 67–72.

[23] V. Millnert, J. Eker, and E. Bini, "Cost minimization of network services with buffer and end-to-end deadline constraints," p. 11, 09 2016. [Online]. Available: https://lup.lub.lu.se/search/publication/8c7b837e-bca3-4375-bb9d-28ce6bbc889a

[24] W. Zhang, T. Wood, and J. Hwang, "Netkv: Scalable, self-managing, load balancing as a network function," in *Proceedings of the 13th IEEE International Conference on Autonomic Computing*, 2016.