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

Millnert, Victor; Eker, Johan; Bini, Enrico

2016

Document Version:
Publisher's PDF, also known as Version of record

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

Victor Millnert*, Johan Eker*†, Enrico Bini‡
*Lund University, Sweden
†Ericsson Research, Sweden
‡Scuola Superiore Sant'Anna, Pisa, Italy

***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 technical report we are investigating the one referred to as Virtual Network Functions Forwarding Graphs, see Figure 1.

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?

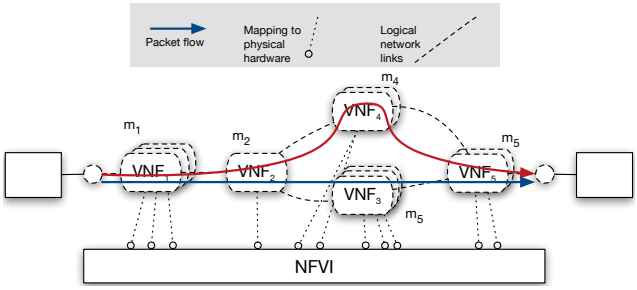

Figure 1: Several virtual networking functions (VNF) are connected together to provide a set of services. A packet flow is a specific path through the VNFs. Connected VNFs are referred to as virtual forwarding graphs or service chains. The VNFs are mapped onto physical hardware, i.e. compute nodes and network fabrics and this underlying hardware infrastructure is referred to as NFVI.

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 technical report 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 works

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 MILP with the goal of maximising 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

To analyse the resource management problem described in Section 1, we model an abstract version of Figure 1 with the one shown in Figure 2. In our model we consider each VNF simply as a *function* that is processing requests. Within each function there are a number of *machines* running (which in Section 1 would correspond to virtual machines).

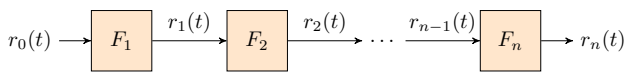

Figure 2: Illustration of the service-chain.

2.1. Input model

The service chain is composed by n service functions. The i-th function, denoted by F_i, receives requests at an *incoming rate* $r_{i-1}(t)$. Then, the *cumulative arrived requests* is

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

We model incoming requests and service speeds of each functions by a fluid approximation. In fact, in [22] they used recent advances in NFV-technology to process requests with a throughput of about 10 million requests per second. We believe this to show that the possible discretization error when using a fluid approximation is indeed negligible.

Finally, each request needs to pass through the entire service-chain within an end-to-end deadline, denoted $D^{\max}$.

2.2. 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. Here one should note that due to the fluid approximation we made earlier, our analysis will assume that a request is processed in parallel by all present machines in the function. Again, with the requests entering at a rate of millions per second along with them being very small we believe that this is a good abstraction. At time t there are $m_i(t)$ machines ready to serve the requests, each with a *nominal speed* of $\bar{s}_i$ (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 processing requests is denoted $s_i(t)$. The *cumulative served requests* is defined as

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

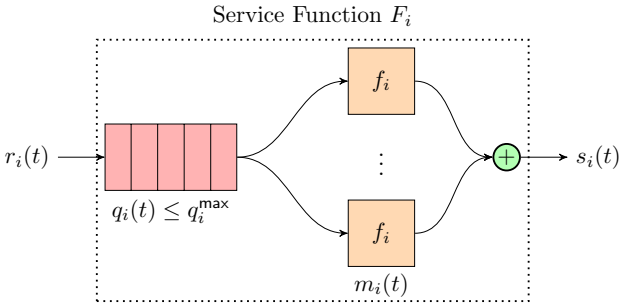

Figure 3: Illustration of the structure and different entities of the service chain.

At time t the number of requests stored in the queue is defined as the *queue length* $q_i(t)$:

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

Each function has a fixed *maximum-queue capacity* $q_i^{\max}$, 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 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 $\hat{D}_{1,n} \leq D^{\max}$.

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 the famous paper [4], Google profiled where the latency in a data center occurred. They showed that less than 1% (¡$1\mu s$) 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 very 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. Hence, once a request has reached the head of the queue and is processed it immediately exits the function and enters the next function in the chain, or exit the chain if exiting the final function. We thus assume that no request is lost in the communication links, and that there is no propagation delay. Therefore, 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.

2.3. 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 utilisation. Reserving this resource results in a cost, and due to the hard reservation, the cost is 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

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

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 $\jmath_i^{\mathsf{q}}$ and can be seen as the cost for having a queue with the capacity of one request. This cost come from the fact that physical storage needs to be reserved so 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

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

2.4. Problem definition

The aim of this technical report is to control the number $m_i(t)$ of machines running at stage 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:

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

with J_i^{c} and J_i^{q} as in (4) and (5), respectively. In this technical report 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

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

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

In Section 3 we are going to make a general consideration about the on/off scheme of each machine, in presence of a constant input rate r. Later in Sections 4 and 5, the optimal design problem of (6) is solved, under a different set of assumptions.

3. Machine switching scheme

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

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

of machines running the 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:

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

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

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

- $\mathsf{off} \to \mathsf{on}$: 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:

- $\mathsf{on} \rightarrow \mathsf{off}$: 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.

In the presence of a constant input, it is straightforward 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. 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 under two different set of assumptions. In Section 4, we will approximate the service functions with linear lower-bounds, which allows us to find a period T_i of each function. Note that the lower-bound approximation incurs in some pessimism in the solution. In Section 5 we will assume that every function will switch on/off its additional machine with the same period, T. For this case we will derive the optimal period T.

4. Linear approximation of service

In this section, the service functions are approximated by linear lower-bounds. This choice allows us finding an explicit solution to the switching periods T_i of each function. Inevitably, the solution incurs in some pessimism due to the approximation.

If the cumulative served requests (2) is lower-bounded by a linear function, as illustrated in Figure 4, the maximum size of the queue at function F_i is attained exactly when the function switches on its extra machine, $q_i^{\mathsf{max}} = q_i(t_i^{\mathsf{on}})$:

$$q_i^{\mathsf{max}} = (r - \bar{m}_i\bar{s}_i)T_i^{\mathsf{off}} = \bar{s}_i T_i \rho_i(1-\rho_i), \tag{14}$$

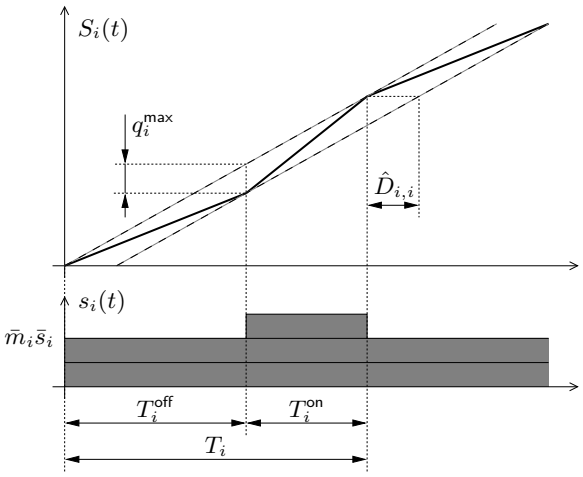

Figure 4: Linear approximation to the cumulative served requests.

while the maximum introduced delay is

$$\hat{D}_{i,i} = \frac{\bar{s}_i}{r}T_i\rho_i(1-\rho_i) = \frac{q_i(t_i^{\mathsf{on}})}{r}.$$

By setting the variable x_i and constants a_i, b_i, and c as

$$\begin{cases} x_i = \hat{D}_{i,i} = \dfrac{\bar{s}_i}{r}T_i\rho_i(1-\rho_i), & a_i = \jmath_i^{\mathsf{q}} r, \\ b_i = \jmath_i^{\mathsf{c}}\Delta_i\dfrac{\bar{s}_i}{r}\rho_i(1-\rho_i), & c = D^{\mathsf{max}}, \end{cases} \tag{15}$$

the optimal design problem of (6) can be formulated as

$$\begin{aligned} \text{minimize } & J = \sum_{i=1}^{n}\left(a_i x_i + b_i\frac{1}{x_i}\right) + J^{\mathsf{lb}} \\ \text{such that } & \sum_{i=1}^{n} x_i \leq c \\ & x_i \geq 0 \end{aligned} \tag{16}$$

with J^{lb} being the cost lower bound as in (7). First, we check the unconstrained solution, which is

$$\frac{\partial J}{\partial x_i} = 0 \quad \Rightarrow \quad x_i = \sqrt{\frac{b_i}{a_i}} \tag{17}$$

If constraint (16) holds at the solution of (17), the optimum is unconstrained and the corresponding optimal cost is

$$J' = 2\sum_{i=1}^{n}\sqrt{a_i b_i} + J^{\mathsf{lb}}.$$

Otherwise, the constraint (16) must be explicitly enforced. In this case the solution is found via Lagrange multiplier. Let λ be the multiplier of the constraint (16), then the solution is

$$x_i = \sqrt{\frac{b_i}{a_i + \lambda}} \tag{18}$$

with cost

$$J'' = \sum_{i=1}^{n}\sqrt{a_i b_i}\left(\sqrt{\frac{a_i}{a_i+\lambda}} + \sqrt{\frac{a_i+\lambda}{a_i}}\right) + J^{\mathsf{lb}} \geq J' \tag{19}$$

with the multiplier λ being the unique positive solution of

$$\sum_{i=1}^{n}\sqrt{\frac{b_i}{a_i+\lambda}} - c = 0 \tag{20}$$

Finally, the switching-period T_i is given by

$$T_i = \frac{r}{\bar{s}_i\rho_i(1-\rho_i)}x_i$$

and the maximum queue-size q_i^{max} are given by Eq.(14). Notice that, for all i such that Eq. (12) holds true, then

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

TABLE 1: Parameters of the example.

there is physically no time to switch the additional machine off and then on again ($T_i^{\mathsf{off}} < \Delta_i$). For all these machines the cost is computed as $\bar{m}_i + 1$ machines are always on (as in Eq. (13)) and not by (11).

Example. Let us apply the described design methodology to a simple example of a service chain with two functions. We assume an incoming rate $r = 17$ of requests per second with an E2E-deadline of $D^{\mathsf{max}} = 0.02$. The parameters of the functions are reported in Table 1. 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 switch a third one on/off periodically.

From (15), the parameters needed to formulate the optimization problem of (16) are: $a_1 = 8.5$, $a_2 = 8.5$, $b_1 = 2.94 \times 10^{-3}$, $b_2 = 4.12 \times 10^{-3}$, and $c = 0.02$. Also, from (7) the cost lower bound is $J^{\mathsf{lb}} = 34$.

The unconstrained solution of (17) is then given by $x_1 = 18.6 \times 10^{-3}$ and $x_2 = 22.0 \times 10^{-3}$. Such a solution, however, violates E2E deadline constraint since

$$x_1 + x_2 = 40.6 \times 10^{-3} > c = 0.02.$$

Therefore, the constrained solution must be explored.

When solving the constrained solution, the Lagrange multiplier $\lambda = 26.6$ is the solution of (20). From (18), this gives the solution of $x_1 = 9.16 \times 10^{-3}$, and $x_2 = 10.8 \times 10^{-3}$, resulting in the periods $T_1 = 186.8 \times 10^{-3}$ and $T_2 = 210.6 \times 10^{-3}$. Note that the off-time for the two functions are $T_1^{\mathsf{off}} = 31.1 \times 10^{-3}$ and $T_2^{\mathsf{off}} = 184.2 \times 10^{-3}$, which are both larger than $\Delta_i = 0.01$. Note that the E2E-delay for this solution is exactly the E2E-deadline. Finally, from Eq. (14) we find that the maximum queue-sizes for this solution are $q_1^{\mathsf{max}} = 155.7 \times 10^{-3}$ and $q_2^{\mathsf{max}} = 184.3 \times 10^{-3}$. Finally, from (19) the cost for the solution is $J'' = 34.871$. 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 periods T_1 and T_2 as well as the maximum queue-sizes q_1^{max} and q_2^{max}. In a real setting the incoming traffic will likely be around million requests per second, [22].

5. Design of machine-switching period

In the previous section, the service functions were approximated by a linear lower-bound, which allowed us to find a period T_i for each function. However, such an approximation leads to an extra cost. In this section, the exact expression of the service functions will be considered. Since the exactness of the service functions leads to an increases in the complexity, the design problem of (6) will be solved while 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). As proved later in Lemma 1 and Lemma 2, the maximum queue size q_i^{max} of any function F_i and the E2E delay $\hat{D}_{1,n}$ are both 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.

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

$$J(T) = aT + \sum_{i:T<\overline{T}_i} \jmath_i^{\mathsf{c}}(1-\rho_i) + \sum_{i:T\geq\overline{T}_i} \jmath_i^{\mathsf{c}} \frac{\Delta_i}{T} + J^{\mathsf{lb}}, \quad (21)$$

where J^{lb} is the lower bound given by (7) and $a = \sum_{i=1}^{n} \jmath_i^{\mathsf{q}} \alpha_i$, 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

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

with δ_i opportune constants, given in Lemma 2.

The cost $J(T)$ of (21) 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:

$$\mathcal{C} = \{\overline{T}_i : \overline{T}_i < c\} \cup \{0\} \cup \{c\}. \quad (22)$$

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

$$\mathcal{C}^* = \{c_k^* : k = 1, \ldots, p-1,\ c_k < c_k^* < c_{k+1}\} \quad (23)$$

with

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

Then, the optimal period is given by

$$T^* = \arg \min_{T \in \mathcal{C} \cup \mathcal{C}^*} \{J(T)\}. \quad (24)$$

Next, we illustrate an example of solution of the design problem. Later, Lemma 1 and Lemma 2 provide the expression of maximum queue size q_i^{max} and the E2E delay $\hat{D}_{1,n}$, as function of the switching period T.

Example. As in Section 4, we use an example to illustrate the solution of the optimization problem of a service chain containing two functions. The input to the service-chain has a rate of $r_0(t) = r = 17$. Every request has an E2E-deadline of $D^{\mathsf{max}} = 0.02$. The parameters of the two functions are reported in Table 1.

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 Equations (8)–(9)). Also, as in the example of Section 4, $\bar{m}_1 = \bar{m}_2 = 2$, $\rho_1 = 0.833$, and $\rho_2 = 0.125$. This in turn 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 (21) 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 (22) containing the boundary is

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

To compute the set $\mathcal{C}^*$ of interior points with derivative equal to zero defined in (23), which is needed to compute the period with minimum cost from (24), 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

$$c_1^* = \sqrt{\frac{\jmath_2^{\mathsf{c}} \Delta_2}{a}} = 0.318,$$

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

$$c_2^* = \sqrt{\frac{\jmath_1^{\mathsf{c}} \Delta_1 + \jmath_2^{\mathsf{c}} \Delta_2}{a}} = 0.421 > c = 0.281.$$

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 $J(T^*) = 34.7$. It should noted that this solution provides a lower cost than the one found by the linear approximation (in Section 4), that is 34.871. This, however, is not true in general.

To conclude the example we show in Figure 5 the state-space trajectory for the two queues. There one can see how the two queues grows and shrinks depending on which of the two functions has their additional machine on. 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 million requests per second, [22].

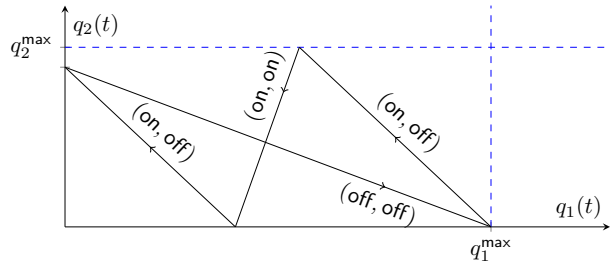

Figure 5: State-space trajectory for the example in Section 5. (on, off) **correspond to F_1 having its additional machine on, while F_2 has its extra machine off.**

Next we derive the expression of the maximum queue size $q_i^{\max}$ as function of the switching period T.

Lemma 1. *The maximum queue size $q_i^{\max}$ at function F_i is*

$$q_i^{\max} = T \times \alpha_i, \tag{25}$$

where

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

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

Proof: The queue size over time $q_i(t)$ is a continuous, piecewise-linear function, since both the input and the service rates are piecewise constant, and the queue size is defined by Eq. (3). Hence, if at t^* the function $q_i(t)$ takes its maximum value, it must necessarily happen that $\partial q_i(t)/\partial t \geq 0$ in a left-neighbourhood of t^* and $\partial q_i(t)/\partial t \leq 0$ in a right-neighbourhood of t^*.

To find the value of $\partial q_i(t)/\partial t$, one needs to distinguish among the four possible cases, Case (1a), Case (1b), Case (2a), and Case (2b), depending on the nominal speeds $\bar{s}_{i-1}$ and $\bar{s}_i$, as is shown in Table 2. These cases, in turn, determine the sign of $\partial q_i(t)/\partial t$, as summarised in Table 3. Note that for $F_i = F_1$, one should consider the input as $F_{i-1} = F_0$ with $\bar{s}_0 = r$, leading to $\bar{m}_0 = 1$ and $\rho_0 = 0$, which would then belong to Case (2b).

Case (1a)	$(\bar{m}_i + 1)\bar{s}_i \geq (\bar{m}_{i-1} + 1)\bar{s}_{i-1}$	$\bar{m}_i\bar{s}_i \geq \bar{m}_{i-1}\bar{s}_{i-1}$
Case (1b)	$(\bar{m}_i + 1)\bar{s}_i < (\bar{m}_{i-1} + 1)\bar{s}_{i-1}$	
Case (2a)	$(\bar{m}_i + 1)\bar{s}_i \geq (\bar{m}_{i-1} + 1)\bar{s}_{i-1}$	$\bar{m}_i\bar{s}_i < \bar{m}_{i-1}\bar{s}_{i-1}$
Case (2b)	$(\bar{m}_i + 1)\bar{s}_i < (\bar{m}_{i-1} + 1)\bar{s}_{i-1}$	

TABLE 2: The four possible cases that one needs to distinguish among. Each case is a function of the nominal speeds $\bar{s}_i$ and $\bar{s}_{i-1}$.

$m_{i-1}(t)$	$\bar{m}_{i-1}$	$\bar{m}_{i-1}+1$	$\bar{m}_{i-1}+1$	$\bar{m}_{i-1}$
$m_i(t)$	$\bar{m}_i$	$\bar{m}_i$	$\bar{m}_i+1$	$\bar{m}_i+1$
Case (1a)	≤ 0	≥ 0	≤ 0	≤ 0
Case (1b)	≤ 0	≥ 0	> 0	≤ 0
Case (2a)	> 0	≥ 0	≤ 0	≤ 0
Case (2b)	> 0	≥ 0	> 0	≤ 0

TABLE 3: Sign of $\partial q_i(t)/\partial t$ as function of the number of on-machines within F_{i-1} and F_i.

Next, the maximum queue-size $q_i^{\max}$ will be derived for each case. We will also derive the best time for each function to start its additional machine, i.e. t_i^{on}.

Case (1a). For this case, illustrated in Figure 6, the sign of $\partial q_i(t)/\partial t$ shown in Table 3, implies that $q_i(t)$ grows only when $m_i(t) = \bar{m}_i$ and $m_{i-1}(t) = \bar{m}_{i-1} + 1$. From this condition, the i-th queue can start to decrease either when $m_i(t) \to \bar{m}_i + 1$ or $m_{i-1}(t) \to \bar{m}_{i-1}$. In the first case, the rate of decrease is

$$\begin{aligned}-\partial q_i(t)/\partial t &= \big((\bar{m}_i + 1)\bar{s}_i - (\bar{m}_{i-1} + 1)\bar{s}_{i-1}\big) \\ &= \big(\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})\big),\end{aligned}$$

and such a state lasts for T_i^{on} (during the interval of length T_i^{on} in Figure 6). This therefore yields a local maximum of:

$$q_i(t_i^{\mathsf{on}}) = T\rho_i\big(\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})\big). \tag{26}$$

It is easy to verify that changing the on-time t_i^{on} to instead be later will yield a larger local maximum, and changing it to instead be earlier will yield a negative queue size. The given t_i^{on} is thus the optimal one, and can be expressed relative to t_{i-1}^{on} as:

$$t_i^{\mathsf{on}} = t_{i-1}^{\mathsf{on}} + T\rho_i \frac{\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})}{\bar{s}_{i-1}(1-\rho_{i-1}) + \bar{s}_i\rho_i} \tag{27}$$

On the other hand, the local maximum when $m_{i-1}(t) \to \bar{m}_{i-1}$ is determined by the interval of length T_{i-1}^{off}, as shown in Figure 6, that is

$$T_{i-1}^{\mathsf{off}}(\bar{m}_i\bar{s}_i - \bar{m}_{i-1}\bar{s}_{i-1}) = T(1-\rho_{i-1})(\bar{s}_{i-1}\rho_{i-1} - \bar{s}_i\rho_i).$$

By taking the maximum of the two local maxima, we find

$$\begin{aligned}q_i^{\max} = T\max\big\{ & \rho_i\big(\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})\big), \\ & (1-\rho_{i-1})(\bar{s}_{i-1}\rho_{i-1} - \bar{s}_i\rho_i)\big\}.\end{aligned}$$

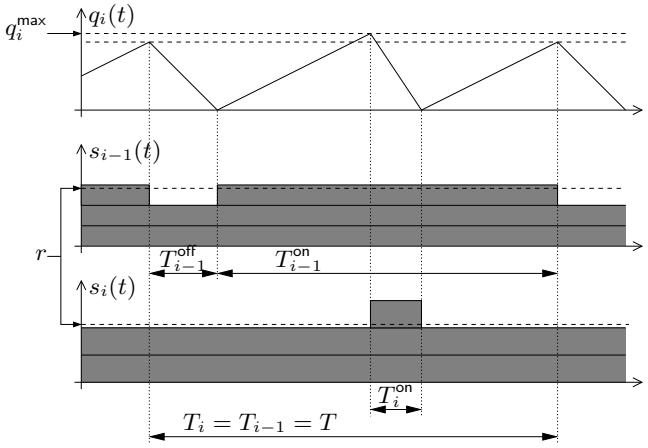

Figure 6: Case (1a): service schedule and queue $q_i(t)$. In this example: $r = 17$, $\bar{s}_{i-1} = 6$, $\bar{s}_i = 8$, $T = 120$, $T_{i-1}^{\text{on}} = 100$, $T_i^{\text{on}} = 15$, $q_i^{\max} = 90$.

Case (1b). As shown in Table 3, the queue size $q_i(t)$ grows if and only if $\bar{m}_{i-1}+1$ machines are running within function F_{i-1}. The maximum queue size, then, is attained at the instant when such a machine is switched off. To analyse this case, we distinguish between two cases: $T_i^{\text{on}} \geq T_{i-1}^{\text{on}}$ (illustrated in Figure 7) and $T_i^{\text{on}} < T_{i-1}^{\text{on}}$ (Figure 8). In both cases, to minimize $q_i^{\max}$, the function F_i must start the extra machine simultaneously as F_{i-1} start its additional machine in order to reduce the rate of growth of the i-th queue, i.e.

$$t_i^{\text{on}} = t_{i-1}^{\text{on}}. \tag{28}$$

Note that the queue size for function F_i will therefore be zero when it switches on the additional machine,

$$q_i(t_i^{\text{on}}) = 0.$$

To compute $q_i^{\max}$, we examine both when $T_i^{\text{on}} \geq T_{i-1}^{\text{off}}$ (illustrated in Figure 7), as well as when $T_i^{\text{on}} < T_{i-1}^{\text{off}}$ (illustrated in Figure 8). By considering them both together, we find

$$q_i^{\max} = \max\big\{T_{i-1}^{\text{on}}\big(\bar{s}_{i-1}(1-\rho_{i-1}) - \bar{s}_i(1-\rho_i)\big), \\ T_{i-1}^{\text{off}}(\bar{s}_{i-1}\rho_{i-1} - \bar{s}_i\rho_i)\big\}$$

and, by considering the expressions of T_{i-1}^{on} and T_{i-1}^{off} of Eq. (10) it can be written as:

$$q_i^{\max} = T\max\big\{\rho_{i-1}\big(\bar{s}_{i-1}(1-\rho_{i-1}) - \bar{s}_i(1-\rho_i)\big), \\ (1-\rho_{i-1})(\bar{s}_{i-1}\rho_{i-1} - \bar{s}_i\rho_i)\big\}.$$

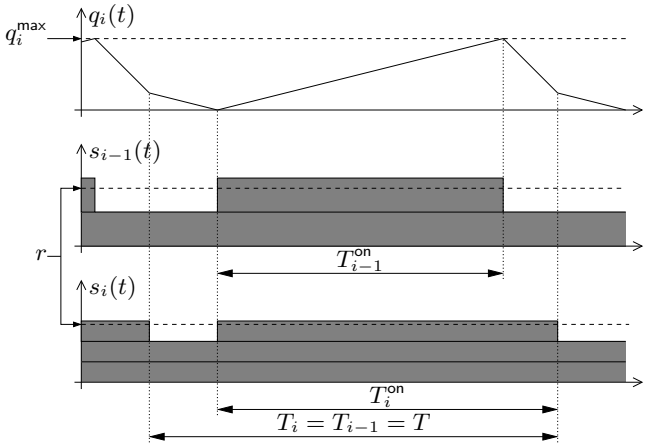

Figure 7: Case (1b), $T_i^{\text{on}} \geq T_{i-1}^{\text{on}}$. In this example: $r = 17$, $\bar{s}_{i-1} = 10$, $\bar{s}_i = 6$, $T = 120$, $T_{i-1}^{\text{on}} = 84$, $T_i^{\text{on}} = 100$, $q_i^{\max} = 168$.

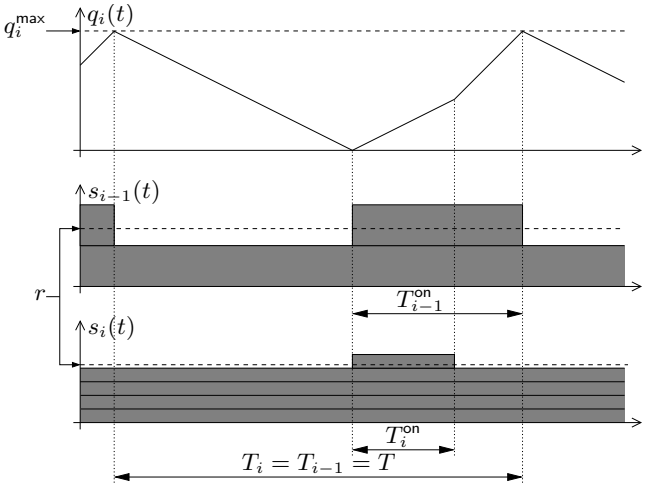

Figure 8: Case (1b), $T_i^{\text{on}} < T_{i-1}^{\text{on}}$. In this example: $r = 17$, $\bar{s}_{i-1} = 12$, $\bar{s}_i = 4$, $T = 120$, $T_{i-1}^{\text{on}} = 50$, $T_i^{\text{on}} = 30$, $q_i^{\max} = 280$.

Case (2a). This case is essentially the same as Case (1b). As shown by Table 3, the only difference is that $q_i(t)$ is reduced whenever F_i has its extra machine on, and grows whenever it is off. This then implies that the maximum queue size is attained when F_i switches on the extra machine. To minimize $q_i^{\max}$, the queue size of F_i should therefore be such that the queue is empty when it switches off the additional machine. Note that this corresponds to both F_i and F_{i-1} switching off their additional machine simultaneously (compare with Case (1b) where the two functions switches on their additional machine simultaneously). The time when F_i should switch on its additional machine is thus:

$$t_i^{\text{on}} = \underbrace{t_{i-1}^{\text{on}} + T_{i-1}^{\text{on}}}_{t_{i-1}^{\text{off}} = t_i^{\text{off}}} - T_i^{\text{on}} = t_{i-1}^{\text{on}} + T(\rho_{i-1} - \rho_i). \tag{29}$$

Note that for this case have to consider both $T_i^{\text{on}} \geq T_{i-1}^{\text{on}}$ and $T_i^{\text{on}} < T_{i-1}^{\text{on}}$ when computing $q_i(t_i^{\text{on}})$:

$$q_i(t_i^{\text{on}}) = \begin{cases} T(1-\rho_i)(\bar{s}_i\rho_i - \bar{s}_{i-1}\rho_{i-1}), & T_i^{\text{on}} \geq T_{i-1}^{\text{on}} \\ T\rho_i\big(\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})\big), & T_i^{\text{on}} < T_{i-1}^{\text{on}} \end{cases} \tag{30}$$

The maximum queue size is, as stated earlier, found when F_i switches on its extra machine. By considering $T_i^{\text{on}} \geq T_{i-1}^{\text{on}}$ and $T_i^{\text{on}} < T_{i-1}^{\text{on}}$ together, the expression for $q_i^{\max}$ can be combined into:

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

Case (2b). Table 3 show the similarity between this case and Case (1a), with the difference being that for this case $q_i(t)$ only shrinks when $m_i(t) = \bar{m}_i + 1$ and $m_{i-1}(t) = \bar{m}_{i-1}$. Therefore, $q_i(t)$ will always grow when $m_{i-1}(t) = \bar{m}_{i-1}+1$. To reduce the rate of this growth F_i should therefore have its extra machine on whenever F_{i-1} has its extra machine on. Furthermore, to reduce the local maximum attained at the end of this growth, F_i should switch on its additional machine such that $q_i(t)$ is empty at the start of it, i.e. $q_i(t_{i-1}^{\text{on}}) = 0$. Furthermore, since $q_i(t)$ grows when both F_i and F_{i-1} has its additional machine off, there is also a local maximum for $q_i(t)$ attained when F_i switches on its additional machine. To minimize this local maximum, F_i should ensure that $q_i(t)$ is empty when

it switches off its additional machine, i.e. $q_i(t_i^{\mathsf{off}}) = 0$. The on-switching time should thus be:

$$t_i^{\mathsf{on}} = t_{i-1}^{\mathsf{on}} - \frac{q_i(t_i^{\mathsf{on}})}{\bar{s}_i(1-\rho_i) + \bar{s}_{i-1}\rho_{i-1}} \tag{31}$$

where

$$\begin{aligned} q_i(t_i^{\mathsf{on}}) &= T_i^{\mathsf{off}}(\bar{m}_{i-1}\bar{s}_{i-1} - \bar{m}_i\bar{s}_i) \\ &= T(1-\rho_i)(\bar{s}_i\rho_i - \bar{s}_{i-1}\rho_{i-1}). \end{aligned} \tag{32}$$

The other local maximum, occurring when F_{i-1} switches off its additional machine is therefore:

$$\begin{aligned} q_i(t_{i-1}^{\mathsf{off}}) &= T_{i-1}^{\mathsf{on}}\big((\bar{m}_{i-1}+1)\bar{s}_{i-1} - (\bar{m}_i+1)\bar{s}_i\big) \\ &= T\rho_{i-1}\big(\bar{s}_{i-1}(1-\rho_{i-1}) - \bar{s}_i(1-\rho_i)\big) \end{aligned}$$

The maximum queue-size for this case thus given by:

$$\begin{aligned} q_i^{\mathsf{max}} = T\max\big\{&\rho_{i-1}\big(\bar{s}_{i-1}(1-\rho_{i-1}) - \bar{s}_i(1-\rho_i)\big), \\ &(1-\rho_i)(\bar{s}_i\rho_i - \bar{s}_{i-1}\rho_{i-1})\big\}. \end{aligned}$$

Conclusion. By taking the maximum among all four cases, Equation (25) is found and the Lemma is proved. □

The expression of q_i^{max} of Eq. (25) 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 of Eq. (9), it always holds that $\rho_i \in [0,1)$. Hence, from the expression of (25), it follows that q_i^{max} is always bounded. □

The second ingredient needed to solve the optimal design problem is the expression of the end-to-end delay.

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*

$$\hat{D}_{1,n} = T \times \sum_{i=1}^{n} \delta_i. \tag{33}$$

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

Proof: With a constant input $r_0(t) = r$ to the service chain, the maximum E2E delay for function F_i is given by

$$\hat{D}_{1,i} = \max_t \frac{R_0(t) - S_i(t)}{r} = \max_t \left(t - \frac{S_i(t)}{r}\right),$$

with $S_i(t)$ being the cumulative served request by F_i, as in Eq. (2), and $R_0(t)$ is the cumulative arrived requests of (1). Since $S_i(t)$ is piecewise linear function, growing with rates $\bar{s}_i\bar{m}_i$ or $\bar{s}_i(\bar{m}_i+1)$ as illustrated in in Figure 9, it follows that the maximum end-to-end delay up to the i-th function, $\hat{D}_{1,i}$, is attained when F_i switches on the additional machine (denoted by t_i^{on}), that is

$$\hat{D}_{1,i} = \max_t \left(t - \frac{S_i(t)}{r}\right) = t_i^{\mathsf{on}} - \frac{S_i(t_i^{\mathsf{on}})}{r}. \tag{34}$$

As illustrated in Figure 9, function F_i will add D_i^* to the maximum E2E delay up to function F_{i-1}. Therefore, it is possible to write the maximum E2E delay up to the i-th function as

$$\hat{D}_{1,i} = \hat{D}_{1,i-1} + D_i^*. \tag{35}$$

Equation (34) then implies that

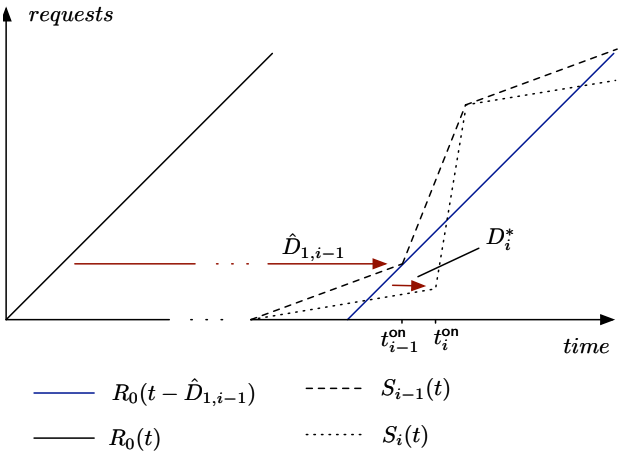

Figure 9: Illustration of how function F_i adds D_i^* to the maximum E2E delay. $R_0(t)$ is the cumulative arrived requests into the service chain, $S_i(t)$ and $S_{i-1}(t)$ are the cumulative served requests by function F_i and F_{i-1} respectively. $R_0(t-\hat{D}_{1,i-1})$ is the linear lower-bound approximation for $S_{i-1}(t)$. The maximum E2E when adding function F_i to the service-chain is then given as $\hat{D}_{1,i} = \hat{D}_{1,i-1} + D_i^*$.

$$\begin{aligned} D_i^* &= \hat{D}_{1,i} - \hat{D}_{1,i-1} \\ &= t_i^{\mathsf{on}} - \frac{S_i(t_i^{\mathsf{on}})}{r} - t_{i-1}^{\mathsf{on}} + \frac{S_{i-1}(t_{i-1}^{\mathsf{on}})}{r} \\ &= t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}} + \frac{S_{i-1}(t_{i-1}^{\mathsf{on}}) - S_i(t_i^{\mathsf{on}})}{r} \\ &= t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}} + \frac{\overbrace{S_{i-1}(t_i^{\mathsf{on}}) - S_i(t_i^{\mathsf{on}})}^{q_i(t_i^{\mathsf{on}})} + \int_{t_i^{\mathsf{on}}}^{t_{i-1}^{\mathsf{on}}} s_{i-1}(x)dx}{r} \\ &= t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}} + \frac{q_i(t_i^{\mathsf{on}})}{r} + \frac{1}{r}\int_{t_i^{\mathsf{on}}}^{t_{i-1}^{\mathsf{on}}} s_{i-1}(x)dx \\ &= t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}} + \frac{q_i(t_i^{\mathsf{on}})}{r} + (t_{i-1}^{\mathsf{on}} - t_i^{\mathsf{on}})\frac{s_{i-1}^*}{r} \\ &= \frac{q_i(t_i^{\mathsf{on}})}{r} + (t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})(1 - \frac{s_{i-1}^*}{r}), \end{aligned}$$

where $\int_{t_i^{\mathsf{on}}}^{t_{i-1}^{\mathsf{on}}} s_{i-1}(x)dx = (t_{i-1}^{\mathsf{on}} - t_i^{\mathsf{on}}) \times s_{i-1}^*$ since $s_{i-1}(t)$ is a piecewise constant function, changing value only in t_{i-1}^{on}. The values of s_{i-1}^* depend on whether F_{i-1} has its additional machine on or off during this time-interval. It should be noted that when $t_i^{\mathsf{on}} \geq t_{i-1}^{\mathsf{on}}$, function F_{i-1} will start its additional machine before F_i does so, and F_{i-1} will therefore have $(\bar{m}_{i-1}+1)$ machines on during the time-interval $[t_{i-1}^{\mathsf{on}}, t_i^{\mathsf{on}}]$. On the other hand, if $t_i^{\mathsf{on}} < t_{i-1}^{\mathsf{on}}$, it follows that F_i will start its additional machine before F_{i-1} does so, and F_{i-1} will only have $\bar{m}_{i-1}$ machines on during the time-interval $[t_{i-1}^{\mathsf{on}}, t_i^{\mathsf{on}}]$. Hence, s_{i-1}^* can be written as:

$$s_{i-1}^* = \begin{cases} \bar{s}_{i-1}(\bar{m}_{i-1}+1), & t_i^{\mathsf{on}} \geq t_{i-1}^{\mathsf{on}} \\ \bar{s}_{i-1}\bar{m}_{i-1}, & t_i^{\mathsf{on}} < t_{i-1}^{\mathsf{on}} \end{cases}.$$

It should also be noted that t_i^{on} and t_{i-1}^{on} are such that the time between them is the smallest possible. Hence, $(t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})$ might be positive or negative, and corresponds to the expressions derived in Lemma 1, Eqs. (27), (28), (29), and (31) for Case (1a)–Case (2b) respectively.

When $t_i^{\mathsf{on}} \geq t_{i-1}^{\mathsf{on}}$ we can therefore write D_i^* as

$$\begin{aligned} D_i^* &= (t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})\Big(1 - \frac{\bar{s}_{i-1}(\bar{m}_{i-1}+1)}{r}\Big) + \frac{q_i(t_i^{\mathsf{on}})}{r} \\ &= \frac{\bar{s}_{i-1}}{r}(t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})\Big(\underbrace{\frac{r}{\bar{s}_{i-1}} - \bar{m}_i}_{=\rho_{i-1}} - 1\Big) + \frac{q_i(t_i^{\mathsf{on}})}{r} \\ &= \frac{\bar{s}_{i-1}}{r}(t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})(\rho_{i-1} - 1) + \frac{q_i(t_i^{\mathsf{on}})}{r}. \end{aligned} \tag{36}$$

For the opposite case, when $t_i^{\mathsf{on}} < t_{i-1}^{\mathsf{on}}$ we instead get

$$\begin{aligned} D_i^* &= (t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})\Big(1 - \frac{\bar{s}_{i-1}\bar{m}_{i-1}}{r}\Big) + \frac{q_i(t_i^{\mathsf{on}})}{r} \\ &= \frac{\bar{s}_{i-1}}{r}(t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})\Big(\frac{r}{\bar{s}_{i-1}} - \bar{m}_i\Big) + \frac{q_i(t_i^{\mathsf{on}})}{r} \\ &= \frac{\bar{s}_{i-1}}{r}(t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})\rho_{i-1} + \frac{q_i(t_i^{\mathsf{on}})}{r}. \end{aligned} \tag{37}$$

In Lemma 1, both $(t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})$ and $q_i(t_i^{\mathsf{on}})$ were derived for Case (1a)–(2b) in Eqs. (26)–(32). For each of the four cases, D_i^* is given by:

Case (1a). For this case, it always holds that $t_i^{\mathsf{on}} \geq t_{i-1}^{\mathsf{on}}$. Hence by inserting $q_i(t_i^{\mathsf{on}})$ of Eq. (26) and $(t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})$ of Eq. (27) into Eq. (36) we can write D_i^* as

$$D_i^* = T \times \frac{1}{r}\bar{s}_i\rho_i^2\frac{\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})}{\bar{s}_{i-1}(1-\rho_{i-1}) + \bar{s}_i\rho_i}.$$

Case (1b). For this case Eq. (28) imply that $t_i^{\mathsf{on}} = t_{i-1}^{\mathsf{on}}$ and that $q_i(t_i^{\mathsf{on}})$ is always 0. Hence, D_i^* will always be 0, implying that $\delta_i = 0$.

Case (2a). Here one must distinguish between two cases: $T_i^{\mathsf{on}} \geq T_{i-1}^{\mathsf{on}}$ and $T_i^{\mathsf{on}} < T_{i-1}^{\mathsf{on}}$. When $T_i^{\mathsf{on}} \geq T_{i-1}^{\mathsf{on}}$ it always hold that $t_i^{\mathsf{on}} \leq t_{i-1}^{\mathsf{on}}$. Hence, by inserting $(t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})$ and $q_i(t_i^{\mathsf{on}})$ given by Eqs. (29)–(30) into Eq. (37) we can write D_i^* as

$$D_i^* = T \times \frac{1}{r}\big(\bar{s}_{i-1}\rho_{i-1}(\rho_{i-1} - \rho_i) + (1-\rho_i)(\bar{s}_i\rho_i - \bar{s}_{i-1}\rho_{i-1})\big).$$

When $T_i^{\mathsf{on}} < T_{i-1}^{\mathsf{on}}$, it instead holds that $t_i^{\mathsf{on}} \geq t_{i-1}^{\mathsf{on}}$. Therefore, by inserting Eqs. (29)–(30) into Eq. (36) we can write D_i^* as

$$D_i^* = T \times \frac{1}{r}\Big(\rho_i\big(\bar{s}_i(1-\rho_i) - \bar{s}_{i-1}(1-\rho_{i-1})\big) + \bar{s}_{i-1}(\rho_{i-1}-1)(\rho_{i-1}-\rho_i)\Big).$$

Case (2b). For this case, it always holds that $t_i^{\mathsf{on}} \leq t_{i-1}^{\mathsf{on}}$. Therefore, by inserting $(t_i^{\mathsf{on}} - t_{i-1}^{\mathsf{on}})$ and $q_i(t_i^{\mathsf{on}})$ given by Eq. (31)–(32) into Eq. (37) we can write D_i^* as

$$D_i^* = T \times \frac{1}{r}\bar{s}_i(1-\rho_i)^2\frac{\bar{s}_i\rho_i - \bar{s}_{i-1}\rho_{i-1}}{\bar{s}_i(1-\rho_i) + \bar{s}_{i-1}\rho_{i-1}}.$$

Conclusion. It therefore follows that for all four cases it is possible to write $D_i^* = T \times \delta_i$, with δ_i being an opportune constant depending only on r, $\bar{s}_i$, and $\bar{s}_{i-1}$. Note that Eqs. (8)–(9) imply that ρ_i (and ρ_{i-1}) depend on r and $\bar{s}_i$ (and $\bar{s}_{i-1}$). Equation (35) then implies that the maximum queueing delay is $\hat{D}_{1,n} = \sum_{i=1}^n D_i^* = T \times \sum_{i=1}^n \delta_i$, and the lemma is proved. □

6. Disturbances

Until now, the analysis of Sections 3, 4, and 5 addressed the case with a constant input rate $r_0(t) = r$. However, variations from such an ideal condition can easily be are modeled by adding disturbances. An impulse disturbance of mass d_i will affect both the maximum queue-size q_i^{max} and the on-time T_i^{on} needed by the addition machine of F_i to process the extra work. If we denote by $\hat{q}_i$ the *largest queue-size for a system without disturbances*, then the maximum queue size that can avoid an overflow is

$$q_i^{\mathsf{max}} = \hat{q}_i + d_i. \tag{38}$$

The additional time needed by F_i to process the disturbances is $d_i/\bar{s}_i$. The only time that the function can find "free time" to process work this extra work, is when it normally would be off, i.e. during T_i^{off} in a period. Depending on how big this disturbance is, it might need several periods worth of T_i^{off}-time in order to process the extra work. The total on-time needed to handle the disturbance, along with the usual incoming request is therefore:

$$\tilde{T}_i^{\mathsf{on}} = T\underbrace{\Big\lfloor \frac{d_i/\bar{s}_i}{T_i^{\mathsf{off}}} \Big\rfloor}_{\text{number of full periods needed}} + T_i^{\mathsf{on}} + T_i^{\mathsf{off}}\overbrace{\Big(\frac{d_i/\bar{s}_i}{T_i^{\mathsf{off}}} - \Big\lfloor \frac{d_i/\bar{s}_i}{T_i^{\mathsf{off}}} \Big\rfloor\Big)}^{\text{fraction of final } T_i^{\mathsf{off}} \text{ needed} \in [0,1)} \tag{39}$$

It will therefore take $\lfloor (d_i/\bar{s}_i)/T_i^{\mathsf{off}} \rfloor + 1$ periods before the extra work is processed and the schedule can return to normal. This assumes that the function does not get any extra disturbances while processing the first one. Note that $\tilde{T}_i^{\mathsf{on}} \to \infty$ as $T_i^{\mathsf{off}} \to 0$, therefore, should $\tilde{T}_i^{\mathsf{on}}$ grow very large it would be necessary to switch on yet another machine. If such a thing would happen in the i-th function, it would thus need to switch between using $\bar{m}_i + 1$ and $\bar{m}_i + 2$ machines, which is the problem studied in this technical report.

The extra on-time needed changes the desired stop-time for the additional machine, and if F_i is switching between $\bar{m}_i$ and $\bar{m}_i + 1$ machines, this would be computed as:

$$t_i^{\mathsf{off}} = t_i^{\mathsf{on}} + \tilde{T}_i^{\mathsf{on}}, \tag{40}$$

where $\tilde{T}_i^{\mathsf{on}}$ is given by (39). Note that this assumes that d_i is known. Note that it could be measured indirectly by taking the difference of the *real queue-size* when switching on the extra machine, denoted by $\tilde{q}_i(t_i^{\mathsf{on}})$, and the *expected queue size* $q_i(t_i^{\mathsf{on}})$:

$$d_i = \tilde{q}_i(t_i^{\mathsf{on}}) - q_i(t_i^{\mathsf{on}}).$$

Note that Eqs. (38) and (40) imply that handling a disturbance will yield a cost increase due to the extra on-time needed and due to the extra queue-size needed. However, it will not affect the solution of the optimization problem, since this added cost is constant and does not depend on the variable of the optimization problem. In Figure 10 we illustrate how the modeling errors can be modeled as a disturbance and how one can stay on for a longer time in order to process the extra load and "catch up".

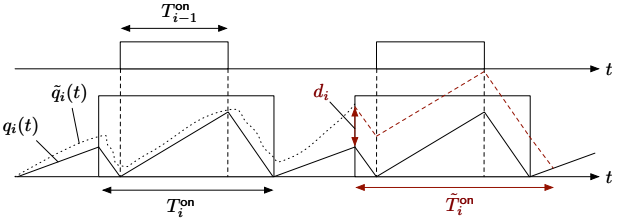

Figure 10: Illustration of how a disturbance can capture modeling errors and how one can process the disturbance in order to "catch up" with the model.

7. Summary

In this technical report 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. It would also be very natural to extend the model to account for uncertainties in the service-rate. With this uncertainty it would be beneficial to close a feedback loop around the service rate in order to guarantee a desired service rate.

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

Source code. The source code used to compute the solution of the examples in Section 4 and 5 can be found on Github at https://github.com/vmillnert/REACTION-source-code.

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] 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.