Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Feedback for increased robustness of forwarding graphs in the cloud

Victor Millnert[*,a], Johan Eker[a,b], Enrico Bini[c]

[a] Lund University, Sweden
[b] Ericsson Research, Sweden
[c] University of Turin, 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 domain is a chain of connected virtual network functions. This allows the capacity of each function to be scaled up and down by adding or removing virtual resources. In this work, we develop a model of such service chain and pose the dynamic allocation of resources as an optimization problem. We design and present a set of strategies to allow virtual network nodes to be controlled in an optimal fashion subject to latency and buffer constraints. Furthermore, we derive a feedback-law for dynamically adjusting the amount of resources given to each functions in order to ensure that the system remains in the desired state even if there are modeling errors or for a stochastic input.

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 Fig. 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? The distilled problem is illustrated as the packet flows in Fig. 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.

This paper is an extension of the workshop paper [3], whose main contributions were the first 4 of the bullet points below. The main contribution of this paper is the final bullet point, namely the introduction of a feedback-law. Together the contributions are:

- a mathematical model of the virtual resources supporting the packet flows in Fig. 1,
- the set-up of an optimization problem for controlling the amount of resources needed by each function in the service chain,
- a solution of the optimization-problem under the assumption of a constant input flow,
- a feedback-law for dynamically changing the resources used by each function, allowing for a stochastic input and impulse disturbances.

* Corresponding author.
E-mail address: victor@control.lth.se (V. Millnert).

http://dx.doi.org/10.1016/j.sysarc.2017.09.005
Received 9 February 2017; Received in revised form 14 June 2017; Accepted 20 September 2017
Available online 22 September 2017

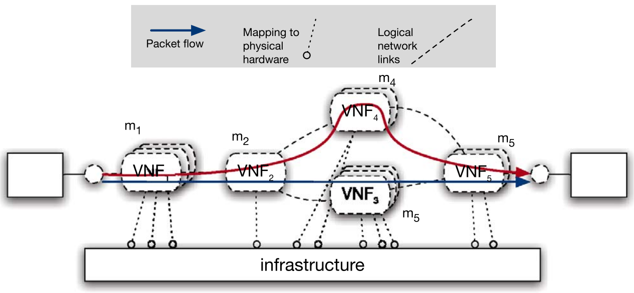

Fig. 1. Several virtual networking functions (VNF) are connected together to provide a set of services. Packet flow through a specific path the VNFs (a virtual forwarding graph). The VNFs consist of a set of virtual resources, e.g., VNF$_1$ consist of three virtual machines, that are mapped onto physical hardware referred to as the virtual network function virtualization infrastructure (NFVI). In the figure there are two packet flows: a blue {VNF$_1$, VNF$_2$, VNF$_3$, VNF$_5$}, and a red {VNF$_1$, VNF$_2$, VNF$_4$, VNF$_5$}. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Related work

In the area of virtual network functions and data center management there are a number of works considering the problem of controlling virtual resources. However many of them focus on orchestration, i.e. how the virtual resources should be mapped onto the physical hardware. Sparrow [4] presents an approach for scheduling a large number of parallel jobs with short deadlines. Cohen [5] and similarly Shen [6] address the issue of placement of VNFs within a physical network. A number of more works focusing on orchestration of VNFs, or scheduling of packet flows, can be found in [7–9], however they do not address the issue of elastically scaling the capacity of the VNFs nor do they allow for any end-to-end (E2E) constraints over the forwarding graphs.

The issue of elasticity and dynamic scaling of VNFs is studied by Kuo in [10], Mao et al. in [11], and recently by Wang et al. in [12] in which they developed a fast online algorithm for scaling and provisioning VNFs in a data center, however they still do not consider the issue of end-to-end constraints on the latency. This is done by Li et al. [13] who present a design and implementation of NFV-RT that aims at controlling NFVs with soft Real-Time guarantees, allowing packets to have soft end-to-end deadlines.

There has been many works outside the area of network function virtualization that has addressed the enforcement of an end-to-end deadline of a sequence of jobs that is to be executed through a sequence of computing elements. In the holistic analysis [14–16] 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 [17] proposed to split the E2E deadline proportionally to the local computation time or to divide equally the slack time. Later, Jiang [18] 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. [19,20] proposed to assign local deadlines to minimize a linear upper bound of the resulting local demand bound functions. More recently, Hong et al. [21] 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 [20,22].

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* [23]. In this area, Henriksson et al. [24] 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. [25], Cruz [26], and Parekh & Gallager [27]. These results built the foundation for the *network calculus* [28], later applied to real-time systems in the *real-time calculus* [29]. 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

We consider a *service-chain* consisting of *n functions*$F_1, \ldots, F_n$, as illustrated in Fig. 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 $r_i(t) \in \mathbb{R}^+$ packets per second (pps) entering the i'th function and the *cumulative arrived requests* for this function is

$$R_i(t) = \int_0^t r_i(\tau)\,d\tau. \tag{1}$$

In a recent benchmarking study it was shown that a typical virtual machine can process around 0.1–2.8 million packets per second, [30]. 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.

2.1. Service model

As illustrated in Fig. 3, the incoming requests to function F_i are stored in the buffer 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*$q_i(t) \in \mathbb{R}^+$:

$$q_i(t) = \int_0^t (r_i(\tau) - s_i(\tau))d\tau = R_i(t) - S_i(t). \tag{3}$$

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.

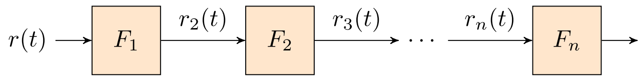

Fig. 2. Illustration of the service-chain of n connected functions where each box F_i represent a virtual network function.