The Journal of Systems and Software 127 (2017) 12–27

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Profiling and accelerating commodity NFV service chains with SCC

Georgios P. Katsikas*, Gerald Q. Maguire Jr., Dejan Kostić

KTH Royal Institute of Technology, Stockholm, Sweden

ARTICLE INFO

Article history:
Received 3 September 2016
Revised 12 December 2016
Accepted 16 January 2017
Available online 23 January 2017

Keywords:
NFV
Service chains
Profiler
Scheduling
I/O multiplexing

ABSTRACT

Recent approaches to network functions virtualization (NFV) have shown that commodity network stacks and drivers struggle to keep up with increasing hardware speed. Despite this, popular cloud networking services still rely on commodity operating systems (OSs) and device drivers. Taking into account the hardware underlying of commodity servers, we built an NFV profiler that tracks the movement of packets across the system's memory hierarchy by collecting key hardware and OS-level performance counters. Leveraging the profiler's data, our Service Chain Coordinator's (SCC) run-time accelerates user-space NFV service chains, based on commodity drivers. To do so, SCC combines multiplexing of system calls with scheduling strategies, taking time, priority, and processing load into account. By granting longer time quanta to chained network functions (NFs), combined with I/O multiplexing, SCC reduces unnecessary scheduling and I/O overheads, resulting in three-fold latency reduction due to cache and main memory utilization improvements. More importantly, SCC reduces the latency variance of NFV service chains by up to 40x compared to standard FastClick chains by making the average case for an NFV chain to perform as well as the best case. These improvements are possible because of our profiler's accuracy.

1. Introduction

A cost effective means for network operators to increase quality of service (QoS) is through placing network functions (NFs) in the network. These functions provide either basic forwarding and routing capacities, or in the case of middleboxes, enrich the dataplane functionality by offering increased security, policy enforcement, performance improvements, etc. to the overlay services (Carpenter and Brim, 2002). However, to deploy and manage traditional middleboxes requires costly capital and operational expenditures (Sherry et al., 2012). As a result, network operators and cloud providers have shifted their focus towards network functions virtualization (NFV) by migrating middlebox functionality from hardware to software[1] running in commodity, off-the-shelf servers (European Telecommunications Standards Institute, 2012).

Making NFV-style packet processing (i.e., software-based) perform as well as its hardware equivalent (i.e., the hardware implementation of the middlebox device) is hard, mainly because of poor I/O performance. Therefore, researchers tailor the operating systems' (OSs) network stacks and device drivers to achieve line-rate forwarding and maximize throughput (Rizzo, 2012; Kim et al., 2012; DPDK, 2016; Bonelli et al., 2012). That is made possible by (*i*) enabling zero copy data transfers from the network interface (commonly abbreviated as NIC) to user-space (bypassing the kernel), (*ii*) pre-allocating memory resources, and (*iii*) batching packet processing to amortize system call overheads over multiple packets.

Although the aforementioned efforts improve the I/O performance of individual NFs, they diverted researchers' interest from the source of the problem. Indeed, no prior work *analyzed in-depth* the system's state when commodity NFV applications are executing, nor have the exact root causes of the observed performance been *quantified or explained*. NFV service providers could benefit from tools that can thoroughly analyze "hot" parts of NFV software stacks and draw attention to those functions that heavily utilize system resources, hence offer the greatest potential for acceleration. Such tools can also offer run-time support to allow automated tuning of the running NFV.

In response, the first contribution of this paper is our NFV profiler that collects data from low-level performance counters from the underlying NFV infrastructure to track packets as they move

* Corresponding author.
E-mail addresses: katsikas@kth.se (G.P. Katsikas), maguire@kth.se (G.Q. Maguire Jr.), dmk@kth.se (D. Kostić).

[1] The NFV Industry Specification Group of the European Telecommunications Standards Institute (ETSI) has defined NFV-based network functions as "virtual network functions (VNFs)" (ETSI, 2013). In this article we refer to such functions by using the generic definition "network functions (NFs)", that does not necessarily require these functions to be virtualized.

http://dx.doi.org/10.1016/j.jss.2017.01.005

from the NICs to the processors (and vice versa) through the different levels of the system's memory hierarchy. Our profiler decomposes the observed per packet latency into components mapped to the involved hardware components (e.g., caches, main memory) and associates these components with their cause(s) (i.e., the responsible pieces of code that cause this latency).

Today, modern services require combinations of NFs, known as service chains, to satisfy their QoS requirements (Quinn and Nadeau, 2015). For instance, Amazon offers services that allow tenants to build their own virtual infrastructure by combining functions such as filtering, routing, slicing, and load balancing (Amazon, 2016). In such an environment, even state of the art frameworks such as ClickOS (Martins et al., 2014) and NetVM (Hwang et al., 2014) cannot achieve high-performance, as there is a substantial throughput degradation when interconnecting multiple NFs.[2] Recent efforts, such as E2 (Palkar et al., 2015) and OpenNetVM (Zhang et al., 2016), overcome this problem by eliminating hypervisor and paravirtualization overheads via lightweight NFs (e.g., placed in containers) interconnected with fast, custom software switches.

Unfortunately, these latest advancements have not yet been adopted by cloud providers and it is unlikely that this will happen soon, as cloud providers continue to rely on commodity OSs, I/O drivers, and switching fabrics. Although techniques such as single root I/O virtualization (SR-IOV) can bypass the hypervisor and pass packets from the NICs to the virtual machines (VMs) (Amazon, 2016), cloud applications still use costly system calls to interact with the NICs. These interactions are frequent and consume a large fraction of the execution time of an NFV instance.

In the context of chained services, according to our profiler, I/O is not the only problem, as the length of a service chain imposes serious scheduling overheads. Part of this problem has been recently addressed by Sivaraman et al. (2016) with their programmable packet scheduling techniques in switches and by Mittal et al. (2016) introduction of packet scheduling algorithms that roughly meet the requirements of a universal packet scheduler. These approaches can affect the order and timing of packet departures from a queue in a switch or NF, however we suggest a promising alternative direction that is inline with Amazon's attempts to integrate custom schedulers in their cloud services (Amazon, 2016).

In contrast to Sivaraman et al. (2016); Mittal et al. (2016), our research findings show that *a chain of NFs requires a global scheduler to make chain-level decisions*, rather than an internal scheduler that executes local switch policies. To address this gap, we designed and implemented the Service Chain Coordinator (SCC). SCC adjusts the frequency of I/O operations in tandem with adjusting the priority and time quanta allotted to each NF by the scheduler, to maximize the effective run-time of the service chain. In short, we make the following contributions:

1. We introduce an NFV profiler that collects and analyzes low-level performance counters in close collaboration with the hardware and OS. To the best of our knowledge, this is the first NFV profiler; a key tool for uncovering the underlying performance problems of NFV service chains.
2. By exploiting the output of the profiler, our run-time component automatically combines multiplexing of system calls and scheduling re-configurations to accelerate NFV service chains running on Linux OSs.

We implemented SCC on top of the FastClick NFV framework (Barbette et al., 2015). SCC's accelerations realize long chains of user-space NFV service chains, based on commodity device drivers, with 3x lower latency and 3x better cache, and main memory utilization compared to standard FastClick chains. SCC chains also achieve *multiple orders of magnitude lower latency variance* compared to FastClick; a crucial performance indicator for highly-interactive services.

In Section 2, we formulate the research problem and provide a quantitative summary of our contributions.

[2] Figs. 10 and 12 of the ClickOS and NetVM papers respectively.

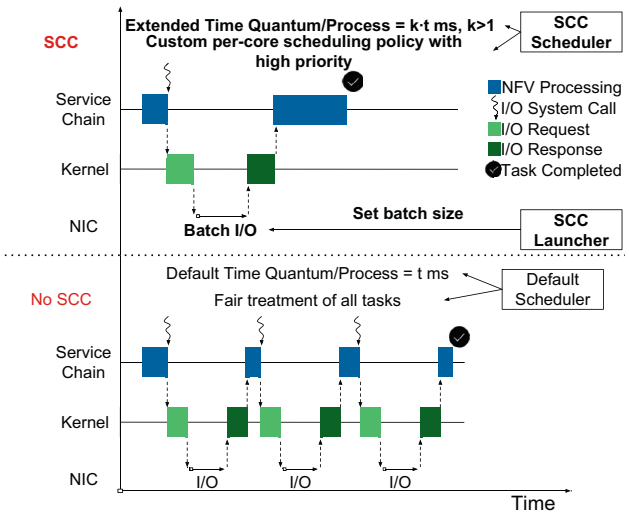

Fig. 1. The SCC run-time combines (*i*) tailored scheduling for NFV service chains via the SCC Scheduler with (*ii*) fewer (but longer) user to/from kernel-space interactions by multiplexing I/O-related system calls via the SCC Launcher. SCC achieves faster completion time, hence lower latency, than the "No-SCC" case.

2. Problem statement

First, we state our research question and the way to address this question.

Key Question: What are the reasons that cause user-space NFV service chains, using commodity OSs and network drivers, to exhibit low performance?

Methodology: In Section 3 we describe an NFV profiler that (*i*) utilizes low-level hardware and software performance counters to track packets as they move across the system's memory hierarchy, (*ii*) measures the per packet latency of the involved hardware components (e.g., caches and main memory), and (*iii*) associates this latency with the cause(s) (i.e., the responsible pieces of code).

We leverage the profiler's power to reveal problems in NFV service chains and quantify their effects (see Section 4). We accelerate NFV service chains by solving those problems identified by the profiler via an automated run-time called SCC (see Section 5).

We illustrate the problems and the solutions realized by SCC in Fig. 1. The bottom part of this figure, labeled as "No SCC", shows a typical way user-space NFV applications based on standard network drivers interact with the NICs via the OS's kernel. As we show in Section 4, this causes two major problems related to the key question stated above:

Problem 1. The service chain at the bottom part of Fig. 1 requires frequent, usually per packet, system calls that cause the service chain to yield the CPU to the OS in order that the latter can perform the necessary I/O operations.

Problem 2. The default Linux scheduler is inappropriate for NFV service chains because it grants short time quanta to the NFV processes and treats them as any other process in the system. As a result, the default Linux scheduler imposes excessive scheduling contention, the latency of which is greater than the actual run-time of a service chain.