Performance Tuning via Lean Measurements for Acceleration of Network Functions Virtualization

Qiang Wu, *Member, IEEE*, Xiangping Bryce Zhai, *Member, IEEE*, Xi Liu, Chun-Ming Wu, Fangliang Lou, and Hongke Zhang, *Fellow, IEEE*

***Abstract*— Network Functions Virtualization (NFV) replaces the specialized hardware with the software-based forwarding to promise the flexibility, scalability and automation benefits. With an increasing range of applications, NFV must ultimately forward packets at rates that are comparable to the native and specialized hardware-based approaches. However, the transition packet forwarding from specialized hardware to software-based has turned out to be more challenging than expected. Thus, NFV acceleration is desperately needed to play a crucial role in the development of NFV. It is an interesting issue how to address the persistent performance tuning in a way that provides far greater flexibility to meet the demands of power. The existing developments are very inefficient, since that the uncontrollable and unanticipated performance regressions frequently occur. Besides, the environments for full system simulations are traditionally expensive and time consuming to evaluate the system performance. In this paper, we propose the methodology named as "NFV Acceleration via Lean Measurements (NALM)" to tune the performance for the NFV acceleration. NALM provides a holistic measurement approach through combining individual measures to quickly identify the bottlenecks, which can help developers with a better understanding of the design tradeoffs. Moreover, the environments for large scale performance simulation are replaced by a debugger. Thus, the waste is eliminated in terms of time consumption and infrastructure costs of the full system simulation. The systematic analysis of the multi-cores speedup ratio highlights the potential optimization space and rules. We further propose the improvement recommendations on efficient practices. The experiments evaluate the specific effects, and the relationship between the metrics and forwarding performance.**

***Index Terms*— Packet forwarding, NFV acceleration, performance tuning, lean measurements, parallel computing.**

Manuscript received 22 December 2021; revised 24 May 2022; accepted 11 July 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Han. Date of publication 29 July 2022; date of current version 16 February 2023. This work was supported in part by the Industry-University-Research Cooperation Fund Project of ZTE under Grant 2022ZTE02-07, in part by the National Natural Science Foundation of China under Grant 61701231, and in part by the Foundation of Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology, under Grant NJ2020022. *(Corresponding author: Xiangping Bryce Zhai.)*

Qiang Wu and Xiangping Bryce Zhai are with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 210016, China, and also with the Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, Jiangsu 210023, China (e-mail: wu.qiang@nuaa.edu.cn; blueicezhaixp@nuaa.edu.cn).

Xi Liu is with the College of Communications Engineering, Army Engineering University of PLA, Nanjing 210012, China.

Chun-Ming Wu is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China.

Fangliang Lou is with the State Key Laboratory of Mobile Networks and Mobile Multimedia Technology, ZTE Corporation, Nanjing 210012, China.

Hongke Zhang is with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100091, China.

Digital Object Identifier 10.1109/TNET.2022.3193686

TABLE I
COMPARISON OF FORWARDING TECHNOLOGIES

Forwarding technology	Software-based	Specialized hardware	Programmable hardware
Efficiency	medium	high	high
Cost	low	medium	high
Openness	high	low	medium
Standardization	high	low	medium
Version cycle	daily builds, DevOps	>3 quarters	>3 quarters

I. INTRODUCTION

IN GENERAL, the three techniques shown in Table I are used to achieve the high-speed packet forwarding of telecom services. Traditionally, the specialized hardware-based approaches require purpose-built chips that are limited in flexibility, and incur high costs. Programmable hardware focuses on the processing of march-action type during the packet forwarding [1]. Although it is incapable of handling the entire forwarding process and can only address the partial performance of this process, the programmability and gained flexibility are enhanced in communication networks. The difficulty in programming, configuration, and firmware upgrades limits their application compared to the instruction-based architectures such as CPUs and GPUs. The software-based forwarding runs on the commercial off-the-shelf (COTS) in NFV. NFV integrating cloud and virtualization technologies replace the specialized hardware with software network functions implemented in COTS, to promise the flexibility, scalability and automation benefits [2]. The software-based packet forwarding has become a popular paradigm with an increasing range of applications [3].

However, the transitioning packet forwarding from specialized hardware to COTS has turned out to be more challenging than expected. It inevitably incurs the forwarding performance penalties due to the constraints in both forwarding software and COTS. Moreover, the uncontrollable and unanticipated performance regressions frequently occur [4]. NFV must ultimately forward packets at rates that are comparable to the native and specialized hardware-based approaches. Thus, there is a strong need for the NFV acceleration [5] to play a crucial role in the development of NFV. Specifically, the major bottlenecks of the software-based forwarding in NFV include:

- The introduction of the virtualization layer increases the overhead in both queue scheduling and packet copy, which leads a decrease of the forwarding performance.

- The low switching efficiency of the virtual switch (vSwitch) has become a bottleneck.
- Resource fragmentation due to the virtualization negatively affects the forwarding performance. A random I/O is generated among the criss-cross packet flows each other because of the complexity of packet forwarding, which will negatively affect the forwarding performance. Meanwhile, the shared resources are competed among the virtual machines (VMs), which can easily lead to the global fragmented I/Os. These fragmented I/Os and the original random I/Os in the process of forwarding overlap each other and are amplified. Then, the forwarding performance eventually degrades.

In addition to the drawback of technology itself such as inefficient performance and relevant high delay, the change in the development model of NFV acceleration is easily overlooked. The wide application of DevOps development model [6] has made the launched software version of NFV more frequent. DevOps aims to help extending the principles of lean development to production, based on the core lean concepts. Its characteristics, such as componentization, Continuous Integration (CI), Continuous Delivery (CD), and gray-scale release can make NFV an inexhaustible source of innovation [7]. With frequently launched software versions, the persistent performance tuning becomes a daily development link. To improve the forwarding capability of COTS servers, a frequent and large-scale optimization work is required at the level of virtualization technology and forwarding software. Thus, a systematic study on the issue of persistent performance tuning of software-based forwarding has to be introduced while providing far greater flexibility than the existing purpose-built hardware [8].

The issue of how to efficiently understand the performance and behavior of these large scale systems is further complicated by the new features after the network virtualization [9], such as the flexibility and sharing on-demand [10]. There are two main challenges that plague forwarding software performance tuning methodologies. First, the environments of full system simulation for system performance evaluation are traditionally expensive and time-consuming in a large scale forwarding systems with frequent changes. The flexibility is an advantage of NFV, and means the frequent changes in functionality of the user plane at the same time. It increases the adjustment frequency of a test environment and reduces the development efficiency, since an expensive test equipment has to turns around among developers [11]. As shown in Table I, the performance testing for software-based forwarding is performed almost daily in DevOps mode. Second, the existing approaches make development inefficient since the uncontrollable and unanticipated performance regressions frequently occur along with the diversity of application scenarios and technical measures [12]. Theoretical analysis is needed to guide the tuning direction and optimization space so that it is more efficient at determining which implementation can meet current requirements better.

In this paper, we propose the metrics of forwarding performance during the development process, that can be used as the target indicators for the improvement in lean measurements.

TABLE II
SYMBOL DESCRIPTION IN EQUATIONS

Symbol	Description
$T_{original}$	Original execution time.
$T_{enhanced}$	Execution time after enhanced performance.
$perf(r)$	Sequential performance of the resources with r BCEs.
w	Original workload.
w^*	Packet forwarding workload constrained by a memory space.
w_d	Packet dispatch and output workload.
w_p	Policy control and service processing workload.
w_c	Packet access work (interthread communication).
$g_d(pps, m)$	Workload function of packet dispatch and output.
$g_i(\alpha)$	Workload function that reflects the parallel processing.
α	Number of processing thread levels.
$g_f(pps)$	Workload function of the packets processing when $\alpha = 1$.
r_p	BCE resources when $\alpha = 1$.
$g_c(pps, \alpha)$	Workload function that reflects the packets processing with α during pps packet traffic.
r_c	BCE resources when α during pps packet traffic.
$Su(n)$	Function of system speedup ratio.
f_t	Variation factor of the total computational scale.
f_a	Proportional variation factor of nonparallel parts.
f_s	Fixed overhead factor.
f_i	New cumulative overhead factor for parallel computing.
$Freq(cpu)$	CPU frequency function.
CUR	CPU utilization rate.
pps	Packets per second.
CPP	Cycles per packet.
IPC	Instructions per cycle.
$eIPC$	Effective instruction per cycle.
IPP	Instructions per packet.

A quick feedback mechanism is established to help developers with a better understanding of the design tradeoffs. Then, we discuss a unified performance benchmark in the case of multiple versions with different feature sets. The main contributions are summarized as follows:

- We propose the novel NALM methodology combined with lean thinking for the software forwarding systems, to form a target-oriented system around the goal of efficient and sustainable performance tuning.
- We eliminate the waste in terms of time consumption and infrastructure costs of the full system simulation.
- We analyze the theoretical multi-cores speedup of NFV acceleration to guide the tuning. Moreover, we propose the new approaches, and recommend the improvements on efficient practices.

The rest of paper is organized as follows. Section II proposes NALM methodology and analyzes the guiding role of lean measurements. In addition, the equation of the speedup ratio for multi-cores optimization indicates the direction of performance tuning. Then, several recommended improvements are illustrated in Section III. Afterwards, we quantitatively evaluate the specific effects of the recommended improvements based on the analyzed data in Section IV. The final Section V provides our conclusions. The symbols used in equations are shown in Table II. More detail related works can be found in Appendix.

II. NFV ACCELERATION VIA LEAN MEASUREMENTS

A. Correlations Between Lean Thinking and NFV Acceleration

Lean software development provides a management philosophy and enables us to select design solutions, methods, and

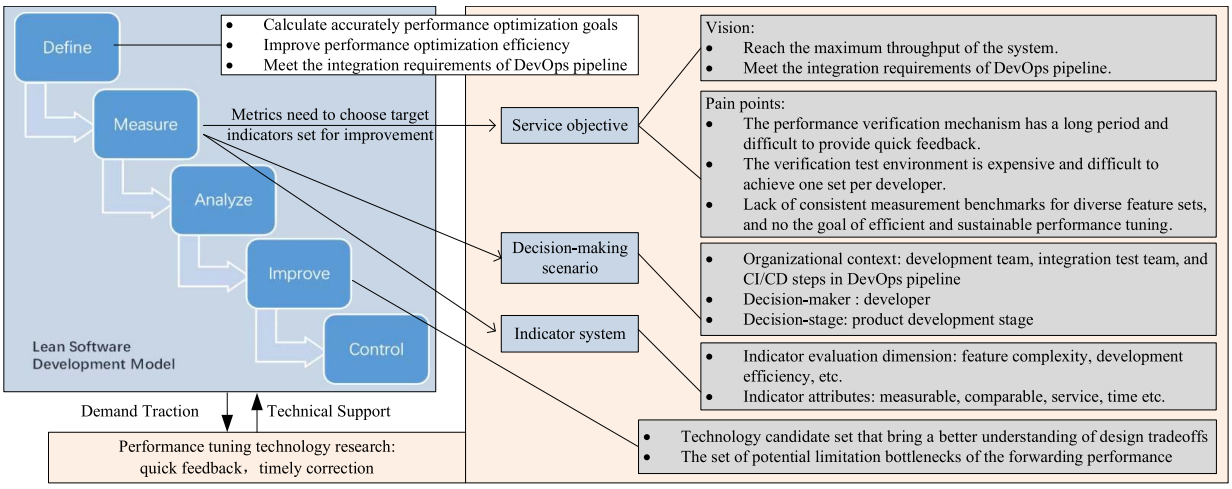


Fig. 1. The guiding roles of lean measurements in forwarding performance tuning.

design tools based on fitness for purpose, the main principles include eliminating waste, overall optimization and fast delivery. Its implementation has remarkably improved the efficiency of development and the quality of products [13], [14]. In some practices, the lean principles are applied to the operationalization of NFV for identification of lean wastes and optimizing them [15]. Even though lean principles are very promising for software development, the introduction of lean development in the domain of forwarding performance tuning is very difficult to achieve because it requires the combined analysis of different measures and technical research [16]. As shown in Figure 1, the research combined with lean thinking presents the gap between the ease of implementation and efficient implementation about performance tuning technology. Lean thinking puts forward the requirements for the improvement of the test environment and metric, which can more fully reflect the value of NALM. Specifically in the field of packet forwarding performance tuning, the guiding roles of lean measurements include:

- **Guide the establishment of reasonable forwarding performance metrics during development.** According to the principle of lean measurements, the measurement indicators need to select those target indicators for improvement. The traditional metric is mainly the forwarding throughput, which reflects the performance level and is also one of the final deliverables. Large-scale softwares are usually developed in parallel by dozens or hundreds of developers. A measurement indicator in generalities cannot indicate the direction of improvement, and is difficult to efficiently find the cause of performance regressions. In addition to final delivery metrics, the intermediate process metrics in detail are still needed. They can be measured quickly, and directly reflect which version or function integration has led to new performance regressions. Especially, each developer can easily obtain it during debugging.
- **Guide the establishment of a unified performance benchmark, in the case of multiple versions with different feature sets.** For large-scale R&D processes, an important meaning of measurement is to unify ideas and methods, so that different development nodes can communicate on a consistent benchmark and reduce the possibility of misunderstanding. Traditional measurement indicators only reflect the macroscopic situation of forwarding performance, and do not contain the complexity of the feature set. Compared with dedicated chips, the feature set of forwarding software changes frequently. The difference in feature set and implementation will cause great performance fluctuations. It is impossible to enumerate all the possibilities and try one by one because of a cost constraints. Therefore, it is difficult to determine the ceiling of forwarding performance tuning. More detailed measurement indicators are needed to objectively reflect the forwarding performance under a certain feature complexity, and to measure whether the optimization effects are within a reasonable range for a variety of approaches. A unified measurement benchmark needs to be established to reflect the optimize efficiency at the different stages to a certain extent.
- **Eliminate the waste in terms of time consumption and infrastructure costs of the full system simulation.** Based on the lean principles, we found that the time consumption and infrastructure costs of the performance simulation heavily waste in the R&D process. If the performance of a NFV network element is designed to be 100G, a test instrument with 100G line-speed have to be used to build a simulation environment that meets the functional requirements. A test instrument of this magnitude is usually very expensive, and function setting is time-consuming and labor-intensive [11]. Therefore, a high-performance simulation environment is usually shared by dozens of people, which is difficult to meet the efficiency requirements of large-scale team during parallel development. It is necessary to introduce a debugger to replace them based on the ideas of quick feedback and timely correction in lean measurements, so that each developer/tester can have one. Moreover, the

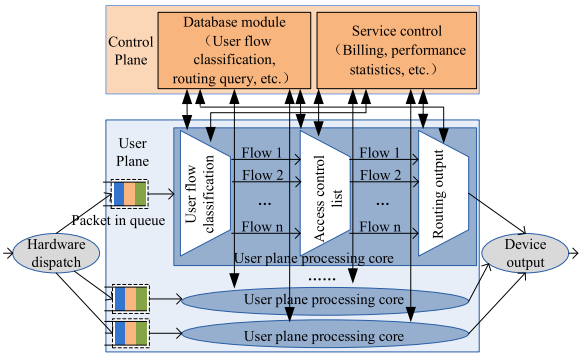


Fig. 2. User-plane software architecture based on a multi-cores processor.

measured indicators have real-time attributes, which can be conveniently observed during function debugging.

B. User-Plane Software Architecture Model

Figure 2 shows an abstract user-plane software architecture based on a multi-cores processor. It is a representative software-based packet forwarding system for large-scale dense model networks, and is also a typical parallel computing architecture. Dealing with large-scale network-intensive packet traffic, the general choice is to scale out nodes horizontally, and the user-plane processing core can be stacked up to extend forwarding capabilities. The data packets are dispatched to the user-plane processing core and the control-plane processing core through hardware. Under the centralized control of the control plane, various service enablers (e.g., DPI (deep packet inspection), billing, etc.) are arranged for the services. The policy decisions at the control plane can be injected into the user plane via the northbound interface. The northbound interface defines the unified operation interface of the forwarding abstraction layer, such as the forwarding table management, queue resources, and statistics reporting.

To address large-scale dense model networks, NFV applications require high throughput, consistent low latency, and high I/O rate processing capabilities to provide the required service level agreements [17]. It is necessary to guarantee that the average interval of packet forwarding processing is less than the average interval of packet arrival for high-speed software packet forwarding. The smaller the average interval between the arrival of small packets, the shorter the processing time. Refer to the technical manual of the corresponding CPU type, the average arrival interval of packets can be obtained in Tables III by simple calculations, a packet have to be processed within 16.8*ns* whereas the time required for a general 2GHz CPU to access DDR memory is 70*ns*, which imposes a very high requirement on software-based packet forwarding performance. A controlled systematic approach to performance tuning is required.

C. Theoretical Analysis for Multi-Cores Speedup Ratio

User-plane software can be theoretically abstracted into applications of high-throughput computing based on a multi-cores processor. Previous researches based on Amdahl's law focused on the characteristics of the multi-cores system as a research topic [18]. Multi-cores scalability was analyzed under fixed-time and memory-bound conditions from the data access perspective. The user-plane system is a dedicated system that is more complex and closer to instantiation than the idealized multi-cores architectures. As shown in Figure 2, it usually consists of multiple boards including multi-cores chips, hardware dispatch network cards, and memory hierarchy. The packet forwarding service has its own characteristics. We focus on packet forwarding service processing and propose a performance model of the user plane, which complements the existing studies.

TABLE III
GENERAL 2Ghz CPU MAIN PARAMETERS

Type	Size	Access scope	Access latency (cyc)	Access time (ns)
Instruction cache	32 k	dedicated	5/7	2.5
L1 cache	32 k	dedicated	5/7	2.5
L2 cache	256 k	dedicated	12	6
L3 cache	20 M	shared	40	20
Non-local L3 cache	20 M	shared	88	44
Local DDR	128 G	shared	140	70
Non-local DDR	128 G	shared	250	125

The packet forwarding service usually does not involve complex data processing. According to Tables III, it can be concluded that the processing bottleneck mainly results in the data access. Therefore, we refer to the memory-bounded speedup model in [19] and analyze the speedup ratio of user-plane software within the whole architecture design of a computing system. Let us define the speedup radio as:

$$Speedup = \frac{Enhanced\ \ performance}{Original\ \ performance} = \frac{T_{original}}{T_{enhanced}}. \quad (1)$$

Following the simple hardware model for multi-cores chips in [18], a multi-cores system as the research object can contain at most n base core equivalents (BCEs), where each individual BCE implements the baseline core. Software architectures can expend the resources of r BCEs to create a powerful core with greater sequential performance $perf(r)$, where the performance of a single BCE is set to be 1. The model allows $perf(r)$ to be an arbitrary function that depends on the actual hardware technique and implementation. Assume the original workload is w. Thus, the original execution time is $T_{original} = w/perf(1) = w$.

For the data access scalability analysis of a packet forwarding service, the task can be divided into three parts: 1) packet dispatch and output work w_d, 2) policy control and service processing work w_p, as well as 3) packet access work (inter thread communication) w_c. Then, $w^* = w_d + w_c + w_p$ is regarded as the packet forwarding workload under a memory space constraint.

Each working node of the user plane can be viewed as a processor-memory pair. In this way, the number of processors and the memory capacity will be expanded simultaneously. The pps (packets per second), which means the transmission rate in units of network packets, is generally used to evaluate the network forwarding capability. Let $w_d = g_d(pps, m)$ be the function that reflects the variable factor of packet dispatch and output workload as the memory capacity increases m

times during the pps packet traffic, and The corresponding BCE resources are r_d.

For large-scale and multilevel packet forwarding services, a pipeline design is necessary. To ensure that the intermediate calculation results can be shared by the related processing threads, more memory queues must be added among processing thread levels and the packets are cached. Let $g_i(\alpha)$ be the function that reflects the parallel processing workload increase factor, where the number of processing thread levels is α. Most of previous studies based on Amdahl's law are special cases with $\alpha = 1$ [19]. $g_f(pps)$ is the function that reflects the packets processing workload. Thus, the policy control and service processing workload is $w_p = g_f(pps) + g_i(\alpha)$, and the corresponding BCE resources are r_p. Let $w_c = g_c(pps, \alpha)$ be the function that reflects the packet access workload increase factor where the number of processing thread levels is α during pps packet traffic, and the corresponding BCE resources are r_c. When the number of cores is scaled by m times, we define the speedup under the memory-bounded model as:

$$\begin{aligned} Speedup_{mb} &= \frac{f_t \times w}{w_d + w_c + w_p} \\ &= \frac{f_t \times w}{\frac{g_d(pps,m)}{m \times perf(r_d)} + \frac{g_f(pps)+g_i(\alpha)}{m \times perf(r_p)} + \frac{g_c(pps,\alpha)}{m \times perf(r_c)}}, \end{aligned} \tag{2}$$

where $r_d + r_p + r_c \leq n$ and f_t is a variation factor of the total computational scale. The total computational scale is related to a variety of factors, such as software architecture, service types, hardware and software infrastructure platforms, and the number of tasks. Routers and firewalls are two different typical forwarding devices. The firewall involves table lookup processing of quintuples for packets, and its service types are more complicated. In the processing of software-based forwarding, it can be considered that the service of firewall forwarding has a higher total computational scale.

Implication 1. The performance tuning issue of software-based packet forwarding involves the whole architecture design of a user-plane system, such as the functional distribution of software and hardware, the service process flow, and the used algorithms.

Implication 2. From the perspective of the entire system, there is no innate limitation to performance scalability. It can be obtained at the cost of an increase in problem size on top of the original work, but the need for technical improvements is mainly to meet the power-performance constraints.

Implication 3. Performance tuning is closely related to the service model. The system design must reasonably allocate the corresponding BCE resources on the basis of predicting the w_d, w_c and w_p workload ratios according to the service model. The memory performance is often an inherent and critical obstacle, since the fixed data-access time is still a technical issue in today's technology, and memory queues must be added among processing thread levels for packet caching.

To more clearly show the relevant factors that affect the system acceleration ratio, we abstract the system speedup ratio $Su(n)$ from another more macro perspective as:

$$S_u(n) = \frac{f_t \times n}{1 + (n-1) \times f_a + n \times f_s + n \times \sum_{i=1}^{n} f_i}, \tag{3}$$

where f_a is a proportional variation factor of nonparallel parts in the theoretical ideal model according to Amdahl's law. For example, a packet needs to be sent to the corresponding port according to the querying result of the routing table. The routing table querying and the packet forwarding have a sequential order, regarded as a serial processing process. f_s is a fixed overhead factor, which represents the proportion of fixed serial execution time that is newly introduced by parallel computing, and the fixed overhead does not change as the user-plane processing core increases. For example, the system overhead introduced by the system runtime environment and supporting platform is closely related to the factor f_s. In the NFV scenario, the virtualization layer hypervisor will introduce a new fixed system overhead, thereby increasing the factor f_s. f_i is a new cumulative overhead factor introduced by parallel computing, which increases with the increase in the number of user-plane processing cores n. For each specific user-plane processing core, f_i is not always the same. In the run-to-completion (RTC) model, there is basically the same cumulative overhead factor f_i, because each user-plane processing core has the same service type. In the pipeline mode, the user-plane processing cores perform computations simultaneously without any competition, i.e., the different stages of the packet forwarding process are parallelized. The service type at each user-plane processing core is not necessarily the same, so its cumulative overhead factor f_i is not always the same.

It can be seen from (3) that simply increasing the user-plane processing core does not always achieve the effect of forwarding performance improvement. In the process of programming even in a multi-cores environment, the interaction processing load between the cores will become a bottleneck when the cluster processing cores reach a certain amount. Simply increasing the number of processors will not linearly increase the performance of the program. Instead, the overall performance will become increasingly poor. Therefore, it is equally important to improve the performance of the software-based packet forwarding performance itself. The increase in the number of processing cores will increase the equipment costs and technical risks indirectly. It is transformed into a comprehensive systemic issue that requires optimization in multiple dimensions. To achieve better forwarding performance, the optimization space that can be used include:

$$S_u(n) := (n, f_t, f_a, f_s, f_i), \quad f_x \in (high, low). \tag{4}$$

A reduction in the factor of total computational scale f_t obtains a better speedup ratio. The changes in the functions distribution of software and hardware and the differences in software algorithms and service process flows can all lead to the changes in the factor of total computational scale f_t. A reduction in the nonparallel parts corresponding to the factors f_a, f_s, and f_i gives a better speedup ratio. Therefore, eliminating serialization caused by similar lock competition becomes an issue that programmers need to solve. The number

of user-plane processing cores n has nothing to do with factors f_a and f_s, and has a definite correlation relationship with f_i. The two relationships need to be carefully balanced according to the service goals. Within a certain range, the forwarding performance can be quickly expanded through core expansion without refined tuning. When the maximum forwarding performance of the entire system is pursued, it is necessary to adjust f_i to the minimum value so that most user-plane processing cores participate in the forwarding processing to improve the overall performance. Due to the influence of the cumulative overhead factor f_i, there is a theoretical upper limit to increase the forwarding performance of the system by increasing the number of user-plane processing cores. A good speedup ratio always presents the following goal status:

$$S_u(n) := (n, f_t, f_a, f_s, f_i) := (n, low, low, low, low). \quad (5)$$

(5) shows the performance tuning direction of software-based packet forwarding performance. The equation can also be used to calculate the execution time after program optimization and to assess whether the software architecture meets the performance requirements.

D. Process Models and Measurements

We aim to enable sustainable performance tuning leading to a lean software process. To improve the performance in terms of the key indicator level, it is necessary to determine the reasons for the high processing power consumption, which is important for initiating the right tuning. The instruction flow indicator is the basis for performance optimization. Thus, we calculate the performance metrics of software-based packet forwarding performance as:

$$pps = \frac{Freq(cpu) \times CUR}{CPP} = \frac{Freq(cpu) \times CUR \times eIPC}{IPP}, \quad (6)$$

where $Freq(cpu)$ is the CPU frequency, and CUR is the CPU utilization rate. Effective CPU utilization for the program must be calculated to set the affinity of a process/thread with a CPU core, and does not include the part of the idle task for the thread loops. The pps is the number of data packets per second that can be processed before dropping data, and is always measured using the smallest packet size. The CPP is the average number of instruction cycles for each packet, which can be converted to pps by using the CPU frequency. The IPC is the average number of instructions executed for each clock cycle. The $eIPC$ refers to the part of IPC for effective processing flow, and does not include the instructions for idle processing. The IPP is the average number of instructions executed for each packet. The goal of creating leaner forwarding performance tuning is to obtain better pps. Moreover, the following deduction is given:

Observation. The relationship between pps and CUR appears close to linear when the $eIPC$ and IPP can be considered invariable during low CUR. Because a larger packet throughput leads to more concurrent processing which affects the $eIPC$ and IPP indicators. In addition, the relationship between pps and CUR appears nonlinear when $eIPC$ and IPP change during high CUR.

The built-in hardware performance monitoring unit of mainstream CPUs provides a lot of hardware event counters. The execution of CPU instructions can be monitored in detail. The above-mentioned measurement indicators can be collected conveniently with the help of these tools. It is necessary to collect these measurement indicators in a planned way, and establish correlations with service features, operating platforms, and resource distribution to perform the statistical analysis. Which can grasp the process nature is conducive to improvement, thereby establishing possible causality, and analyzing the key factors that affect the performance changes.

In the case of NALM, the performance metrics must be continuously monitored in the software continuous integration process for the rapid detection of limitation bottlenecks to judge the effectiveness of the tuning direction in time. Functions and instructions that run most of the time and call high frequency processes need to be specially focused on. Thereafter, continuous improvements to avoid problems caused by the cumulative cross-effects of limitation bottlenecks and aid in obtaining leaner software performance are performed.

III. Improvement Recommendations on Efficient Practices

Telco and NFV application workloads tend to be any combination of latency sensitive, jitter sensitive, or demanding high packet rate throughput or aggregate bandwidth. The exact benefits and effects of each of optimization technique choices will be highly dependent upon the specific applications and workloads, and therefore need to be continuously tuned for best performance. In this section, we summarize our findings and recommend best practices to tune the different layers of an application's environment for Telco and NFV workloads.

A. Push a Virtualization-Based Platform to Its Full Potential

1) Offloading Service Traffic by Smart Network Interface Card (NIC): Network-related workloads are particularly expensive in terms of computing resources. The workloads for virtual switching functions alone can consume more than 90% of a server's available CPU resources. Offloading network tasks can return these important resources to the application layer. In a large forwarding device, a smart NIC and a CPU jointly schedule and cooperate to realize the processing of the packet forwarding function. A OVS table and other related processing functions can be mostly removed from a CPU. A smart NIC handles the functions and offloads the packet processing workloads from the CPU. Moreover, a CPU plays the role of workload scheduling. Through a OVS table, a CPU can determine which service types and how much workloads should be performed by a smart NIC [20]. Corresponding to the aforementioned theoretical analysis for multi-cores speedup ratio, the practice can effectively reduce f_t (a variation factor of the total computational scale), thereby obtain better system speedup ratio $Su(n)$. A lower IPP is presented in the performance metrics, which can be continuously and visually monitored. The comparison results are shown in Figure 8 of Section IV.

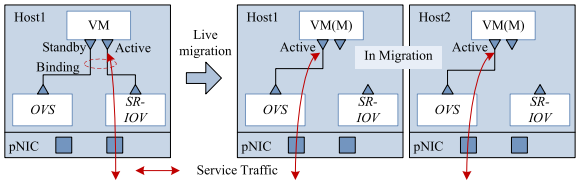


Fig. 3. Live migration of VMs with the SR-IOV network card.

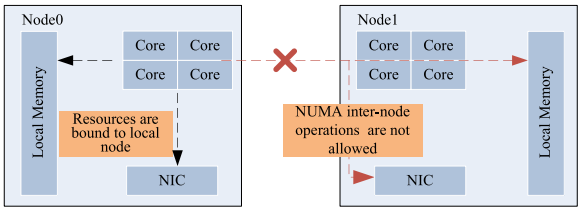


Fig. 4. Resources are bound to local node.

SRIOV and OVS table synching are often used in combination. OVS table synching is responsible for east-west horizontal traffic forwarding, and can solve complex networking issues. However, VM migration may become a problem. Migrating one VM directly to another SRIOV server (even if this server supports live migration) may fail at the start of migration unless that the load of the host is migrated from a SRIOV server to a non-SRIOV server. This is a necessary function that needs to be researched before SRIOV is deployed in a production environment. As shown in Figure 3, the VM configures the SRIOV VF (virtual functions) interface and Virtio interface, and then binds the two interfaces to implement live migration among VMs. In the normal operating mode, the VF network card is used as the main network card. When VM live migration occurs, the VM management function will delete the VF network card temporarily by replacing it with the Virtio network as the main network and then perform VM migration. When the VM live migration is completed, the VM management function will re-add the VF network card to the VM and bind the Virtio network card and VF network card for the next migration. Then, the VF network card becomes the main network card again.

2) Enhanced NUMA (Nonuniform Memory Access Architecture) Deployment: NUMA is a distributed memory access method in which processors can simultaneously access different memory addresses and greatly increase parallelism [21], [22]. A large number of NUMA internode operations can severely affect virtualized forwarding performance, called as the NUMA trap. Now, we propose the specific deployment method under a virtualization-based platform. As shown in Figure 4, the CPU, the memory, and the PCI-I/O should be bound on the same NUMA node and properly match the capacity according to the service model to avoid inter-node access to resources. With memory affinity, the cores and the accessed memory are attributed to the same NUMA node to avoid the performance penalty of cross-node access. In the NUMA mode, the processor is divided into multiple nodes, each of which is allocated the local memory space. The processors in all nodes have access to all of the physical memory of the system, but accessing the local memory in same node takes considerably less time than accessing the memory in other nodes. So does the other local resources (network cards, etc.). The practice can effectively reduce the factor f_a of nonparallel parts and obtain better system speedup ratio $Su(n)$.

3) Compiling Chain and Runtime Environments: The performance of the runtime environment is one of the most important factors that constrain the transaction throughput in software-based packet forwarding performance. Compilation optimization can provide a considerable performance boost, but hundreds of continuously growing available design and optimization choices make this job quite difficult [23]. The practice can effectively reduce the fixed overhead factor f_s, thus obtain the better system speedup ratio $Su(n)$. A higher $eIPC$ is presented in the performance metrics when a more reasonable configuration, which can be continuously and visually monitored in the tuning process of compilation optimization. This point can be studied on a special issue, and this paper will not discuss it.

B. Construction of a Software Architecture According to the Service Characteristics

In general, an optimization at service architecture level is the first choice, according to the service characteristics, to make full use of the hardware resources and to improve the mechanisms and processing flows. The optimization goal is to reduce the nonparallel parts factor f_a and the cumulative overhead factor f_i. If a breakthrough in this aspect can be found, there are usually relatively large gains. To judge whether a service architecture optimization is effective, the intuitive and real-time feedback can be obtained by monitoring IPP and $eIPC$ indicators.

1) Lock-Free Parallel Computing Architecture: When more than two threads need to operate in the same data area, there are issues of read synchronization and contention writing. For example, when two threads receive two packets from the same flow, they both need to update the count of the flow table. Then, the system will encounter an issue of contention writing at this time. The situation of the read operation is relatively simple. If it is not locked, the read process of thread A will be interrupted by thread B to write the updated data. At this time, the data read by thread A are dirty as part of the data is updated by thread B. There is also a case where a read thread reads the data when write thread A is writing the data. In both cases, there is an order issue for read and write.

One solution is to add locks. Read and write threads all need to acquire locks. During a data operation, the read and write operations by other threads need to wait for the locks. The lock instruction itself is simple, but the lock contention impacts the performance. Only one thread can access the critical section when multiple threads require access, which decreases the transaction concurrency. Lock-based programming needs to be accessed in an orderly manner when sharing data, and all operations changing the shared data must show atomic semantics. Even a simple code such as $++i$ also requires the use of locks. Lock-based programming faces declining

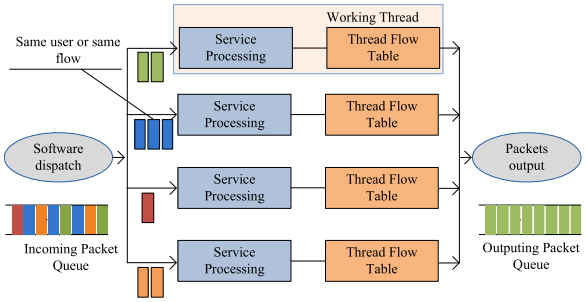


Fig. 5. The lock-free parallel computing architecture.

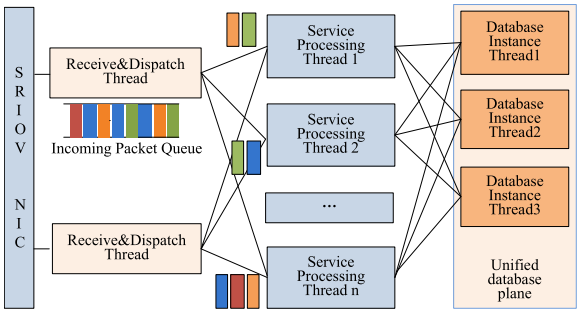


Fig. 6. Service processing and user data separation.

efficiency, deadlock, priority inversion and other issues. Lock-based programming requires designers to carefully optimize and solve these issues. An important effect of lock-free programming is that a series of threads accessing lock-free operations. If a thread is suspended, it will not prevent other threads from running (non-blocking). The significance of lock-free programming is not absolute high performance, but is that its use can avoid deadlock/livelock, priority inversion and other issues. Atomic operations can be considered indivisible when operating on memory, and other threads do not interrupt the operation. In lock-free programming, not all operations are atomic. Indeed, only a limited set of operations are atomic, which means that lock-free programming is a difficult task.

For the packet forwarding services, the traditional multi-cores database locks have high overhead. The lock-free parallel computing architecture shown in Figure 5 is used to achieve more efficient forwarding processing performance. Parallelism is provided by multiple threads processing packet delivery at the same time. An incoming packet queue is dispatched to the working threads by a software dispatch module according to the service policies. To ensure the lock-free programming in the service processing architecture, service policies are set so that the packets with mutex are dispatched to the same thread. Considering an example of 5th-Generation core network (5GC), a user session corresponding to the user is established in a software dispatch module when a user's initial packet reaches an 5GC. The thread number registered for this user is recorded in the user session and generates the user's forwarding flow table in the private flow table of the thread. Subsequently, the packets from the same user will be dispatched to the same thread by the flow table match. Each user data area is distributed in the private flow table of each service thread to achieve the lock-free access. The packets are distributed precisely to the thread where the user data area is distributed in the dispatch cores. This avoids the issues of read synchronization and contention writing. Guaranteed from the logical service architecture, there is no lock contention conflict which eliminates the waiting time of lock. The design architecture is suitable for the devices of forwarding policy per-user (or per-flow), such as 5G UPF (User Plane Function), vFW, SDN switch.

2) Service Processing and User Data Separation Architecture: The lock-free parallel computing architecture is certainly more efficient because there is no lock contention conflict. Though it eliminates lock wait time, there are applicability issues in some scenarios.

The types of service that the single-user or single-flow forwarding throughput is limited by the single core/thread processing capability. The currently known processing core of 2.4 GHz frequency and the single thread processing capability are in the range of 500 K - 2 Mpps, usually calculated as 500 Kpps. 2 Mpps corresponds to a specific traffic model and are calculated according to the packet size of 512 bytes, which is approximately 2.5 G - 10 Gbps. Due to the limitation of single core forwarding performance, the lock-free parallel computing architecture is not applicable when the throughput of a single-user or single-flow exceeds the upper limit of a single core's forwarding performance.

The types of service that the packets cannot be dispatched according to a single user or single flow classification. For example, the five-tuple sets will change when the packets are forwarded in firewall, because the traffic is processed through network address translation (NAT). Although the downlink and uplink traffic belong to the same session connection, the five-tuple index is not the same. The dispatch cores distribute packets to each service processing thread according to the flow table established by the five-tuple sets. The upstream and downstream processes of the same session connection may be distributed in different service processing threads. When more than two service processing threads synchronously access the same data area, there are issues of read synchronization and contention writing. Because it is difficult to dispatch packets according to a single-user or single-flow policy, the lock-free parallel computing architecture is not applicable.

For these types of service, the service processing and user data separation architecture shown in Figure 6 is an alternative solution. A unified database plane is constructed with computation and data separation. Service processing threads are completely homogeneous, no longer store user data, and become pure computing modules. The user data or flow data are managed by the dedicated database instance processing cores. A semi globalization mechanism is used, and each user's data or flow data is maintained by a unique database instance core. Service processing threads focus on the packet forwarding processing and query a hash table to locate the corresponding database instance thread. In the lock-free parallel computing architecture, the packet forwarding of the same user or the same flow is processed by the same cores. Because of the accurate dispatch, the packets of the same user or the same flow are delivered to the specific core in

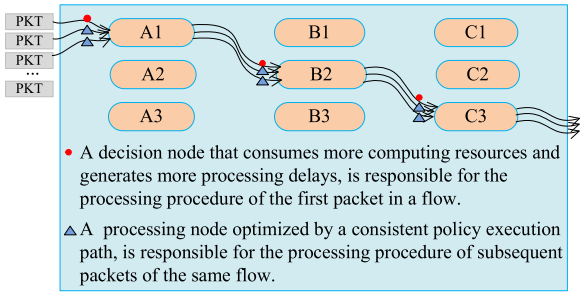


Fig. 7. Consistent policy execution path for the same packet flow.

sequence, and there is no packet disorder problem. In this architecture model, the packet forwarding of the same user or the same flow is processed by multiple cores, and it is necessary to sequence packets of the same user or same flow. The packet sequence is guaranteed by a database instance thread, and single user data are centrally processed by one database instance thread. For example, a packet fragmentation is integrated by the same database instance thread. Theoretically, the throughput of a single-user or single-flow can achieve the limit of the processing capability of multiple service cores forwarding performance accumulations. The dispatch cores evenly distribute the packets to multiple service processing threads with a single-core load balancing algorithm. The ratio of the database instance threads to the service processing threads can be flexibly configured according to the different service models.

3) Design Service Process Flow Based on Traffic Model Characteristics: Continuous and stable requirements in development continue to increase service complexity, causing service process flow to become more sophisticated and less efficient. It is very important to keep the simplified process flow. Once it becomes complicated, the corruption speed and complexity of the system increase exponentially, and the programming skills cannot compensate for it. A good design of the service process flow is the key to performance tuning. Service process flows are related to the specific service characteristics, and the different service types lack commonalities. For example, there is a packet flow $\mathbf{p} = \{p_1, p_2, \ldots, p_N\}$ with N packets, and the n-th packet consumes the computing resource r_n leading to the processing delay d_n. In general, we aim to optimize the cost for the packet flow, i.e.:

$$\text{minimize} \quad \omega \sum_{n=1}^{N} r_n + (1-\omega) \sum_{n=1}^{N} d_n, \tag{7}$$

where ω is the coefficient. Specially, our goal reduces to optimize the computing source $\min \sum_{n=1}^{N} r_n$ when $\omega = 1$. Besides, our goal reduces to optimize the total processing delay $\min \sum_{n=1}^{N} d_n$ when $\omega = 0$.

As shown in Figure 7, the consistent policy execution path is a more efficient service process flow. It is necessary to determine the forwarding path while receiving the first packet of a flow and a consistent policy execution path for the same packet flow. Intuitively, it can eliminate many processing delays and conserve decision-making overhead in this situation as $r_1 \gg r_2 \approx \cdots \approx r_N$ and $d_1 \gg d_2 \approx \cdots \approx d_N$. Because, the system needs to perform analysis and judgment to determine the action while processing the first packet of a flow. Then, the forwarding flow table is modified, corresponding to the decision of the forwarding path. According to matching the forwarding flow table, the subsequent packets of the same flow directly perform the same actions of the first packet to avoid making repeat decisions.

C. Polishing and Perfecting Details of the Software

Software-based packet forwarding is a kind of high performance computing. If the coding method is not suitable or the algorithm is not refined enough, it will often become a bottleneck of performance. Previous research proposed some general analysis, such as ILP (instruction-level parallelism), TLB (translation lookaside buffer), etc. [25]. But for the specific scenario of virtualized packet forwarding, there are still issues of method selection and adaptability that need to be resolved.

1) Algorithm Optimization: The algorithm optimization mainly aim to reduce the total computational scale factor f_t, thus obtain better system speedup ratio $Su(n)$.

High-performance index: The HASH algorithm is a frequently used technology for fast index queries in high-performance packet forwarding. If the selection of the HASH algorithm is not suitable or the implementation is not precise enough, it often becomes a bottleneck for performance. The traditional zipper method for HASH conflicts is not suitable for linear speed forwarding scenarios. The performance of the existing HASH algorithm itself is not a bottleneck. The average collision depth of the HASH barrel is also approximately between 2 and 3. The main problem is that a query requires multiple memory accesses, and a large amount of CPU time is spent on waiting for memory access. The key issue is how to reduce the number of index queries accessing memory for HASH conflicts.

The table index array and its corresponding signature array are added to the original data structure HASH table node. The keyword is a 4-byte signature computed using the JHASH algorithm [26] and stored in the signature array. The structure is aligned to 64 bytes, i.e., within the cache line. The original HASH algorithm queries the Level 1 HASH table node. Then, a Level 2 JHASH is performed on keywords, while the JHASH signature is calculated. Then, the JHASH signatures of each element in the signature array of the Level 1 HASH nodes are compared in turn. If the signature matches, the table resource is accessed according to the corresponding table index, and the full keyword matching is performed. The HASH algorithm optimization can improve the performance due to the following points:

- Reduced number of memory jumps. The HASH collision chain is transformed into an array, and the JHASH signature is compared when searching for resource entries first (4-byte comparison efficiency is high). The above two points ensure that the memory jump count is controlled within 2 iterations.
- Full use of the CPU cache line is made to increase CPU utilization.

- The advantage of the JHASH algorithm. The JHASH algorithm is simple and has better discreteness. The DPDK's experimental result shows that after 2 levels of HASH, the collision array of the HASH barrel is enough to store in a cache line.

There are also many algorithms used during packet forwarding, such as the access control list (ACL) algorithm and the meter algorithm. According to the characteristics of the service traffic model, the forwarding performance in real time can be reflected with the aid of an observation indicator of pps, CPP, $eIPC$, and IPP in the case of NALM, which can be chosen as a better algorithm.

2) Reasonable Planning for Memory and Caching: Memory performance is often a key limiting factor in software-based forwarding performance. For architects, a very important task is to coordinate the relationships between the CPU core computing speed and the Level 1, 2 and 3 caches. The optimization goal is to reduce the nonparallel parts factor f_a and the cumulative overhead factor f_i. The following methods can more efficiently use the cache: The codes with high frequency reside in the instruction cache; reduce the inter-core conflicts in resource access; limit the memory range of thread access; reduce the page switching consumption; and increase the hit rates of TLB and caches.

IV. Experiments and Discussions

NALM provides a holistic measurement approach combining individual measures to achieve a comprehensive analysis. In this section, we evaluate NALM with the goals: 1) demonstrate the relationships between the metrics and forwarding performance in the system measurement method proposed by NALM, and 2) quantitatively evaluate the specific effects of the recommended improvements. The network forwarding performance is traditionally tested with RFC2544 or a similar performance test.

A. Two Typical Execution Models for SFCs

To illustrate the key role of NALM in the performance tuning process, two simple yet representative SFC examples are implemented as simulation models using DPDK libraries 18.11, including five network functions:

1) Network address translation (NAT), mainly completes the mutual conversion between public IP address and private IP address. NAT performs a 5-tuple querying to decide how to translate the IP address header for a packet flow according to a set of configured rules.

2) Deep packet inspection (DPI), performs in-depth inspections for application layer loads of packets. DPI filters and controls the traffic according to a pre-defined strategy for the fine-grained identification of services, the statistics on service traffic proportions, and the service proportion shaping. DPI in this experiment is mainly for content charging.

3) Layer 3 forwarder (L3FWD), performs longest matching rule by querying a routing table to find the output interface according to a destination IP address.

4) Access control (ACL), performs a 5-tuple querying to decide whether to block a flow according to a set of configured rules.

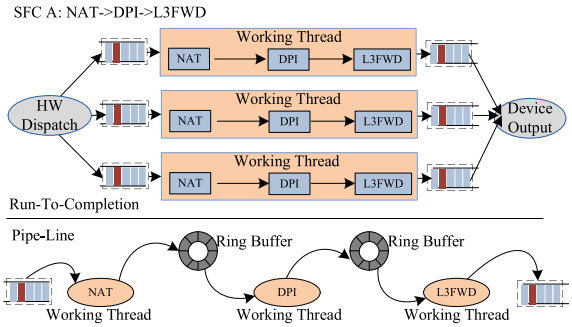


Fig. 8. Two typical execution models of SFCs.

5) Basic stateful load balancing (LB) of Layer 4, establishes a connection with one of the several destination servers based on the 5-tuple for a packet flow. It includes frequent flow table query, address rewriting and other processing.

The traditional packet forwarding model include two types, i.e., RTC and Pipe-Line. They are used in the SFCs forwarding processing in this experiment [29]. Under the Pipe-Line model, the forwarding function of the entire SFC is split into multiple independent stages, and the products are delivered through queues in different stages. Through the filter, different working threads can be allocated for different operations, and the two speeds can be matched through the queue to achieve the best concurrency. The delivery of packets in inter-cores will cause performance penalty. Under RTC model, a SFC forwarding processing contains several different logic functions, and they will all run on a single CPU core, thereby avoiding delivering packets inter-cores. As shown in Figure 8, the SFC A implements the sequential processing of NAT→DPI→L3FWD. As a comparative use case, the SFC B implements the sequential processing of ACL→LB→L3FWD. Two typical execution models for SFC B is similar to that of SFC A. The example SFC A shown in Figure 8 is run on 18 cores for the two typical models, so as does SFC B.

The Intel VTune Amplifier tool [30] is used to measure the number of CPU clock cycles when fetching instructions and data to quantify the transmission overhead inter-core. Then, the in-depth performance analysis is performed. Table IV lists the number of CPU clock cycles of SFCs processing in two typical models. These measurements can be quickly obtained from a debugging PC at any stage of code debugging. The long-term accumulation of similar measurement data in processes will become an important reference for design tradeoffs. Figure 9 presents the forwarding performance of SFCs in two typical models. The measurements need to use a high throughout tester, such as IXIA BPS which is one of the few software-based traffic generators capable of generating 100Gbps traffic. It is very expensive in terms of cost. In terms of timeliness, this performance test must be performed after that all the functional features are developed and delivered. Therefore, it cannot be used frequently and conveniently as VTune. The key observations are summarized as follows:

1) The CPP measurement in Table IV always shows a negative correlation with the forwarding performance in Figure 9. There is sufficient and detailed measurements to conduct in-depth performance analysis in terms of the key indicator

TABLE IV
PROCESSING CYCLES OF SFCS IN TWO TYPICAL MODELS

SFC A	NAT	DPI	L3FWD	CPP
Run-To-Completion (cyc)	210.3	308.5	160.6	679.4
Pipe-Line (cyc)	69.3	112.2	26.1	207.6
SFC B	ACL	LB	L3FWD	CPP
Run-To-Completion (cyc)	10.3	15.5	6.6	32.4
Pipe-Line (cyc)	36.5	75.8	23.3	135.6

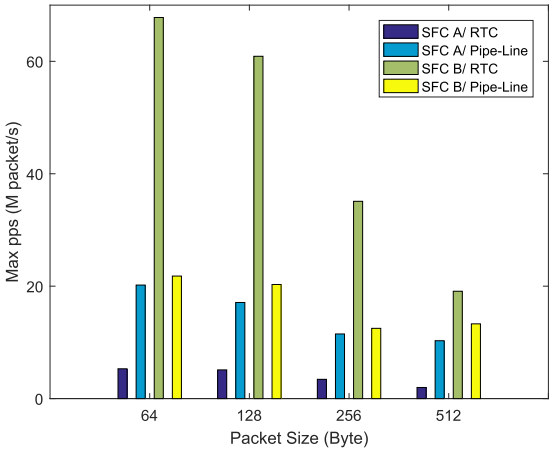


Fig. 9. Forwarding performance of SFCs in RTC and Pipe-Line.

level, which can help determine the cause of performance penalty.

2) The design of the performance forwarding scheme is closely related to the service characteristics. The accumulation of experience data is required to bring a better understanding of design tradeoffs. Compared with the Pipe-Line model, SFC B has better performance due to avoiding transferring packets inter-core under the RTC model. Whereas, SFC A has better performance under the Pipe-Line model rather than RTC model. With increasingly complex behavior of SFC B, the size of the state maintained by SFC B becomes very large. Whereas, the L1/L2 cache resources is fixed and limited in each core. SFC B requires more resources than the local cache sizes. It is difficult for RTC to support a large number of flows with high-throughput due to poor cache locality (both i-cache and d-cache). Even for the same network functions, there will be significant differences in performance when accessing the local and non-local caches. Decision-making for the optimization of forwarding performance requires reference to meticulous measurements.

3) In this case, the CPP is used as one of consistent metric for different feature sets. It reflects the complexity attribute and development capability attribute of the feature set.

B. Experiments on a Large-Scale Commercial Platform

The above two experiments use simulation models to verify the metrics proposed by NALM and its improved practice. Now, we conduct experiments on a large-scale commercial platform to illustrate the actual effect of the proposed tuning

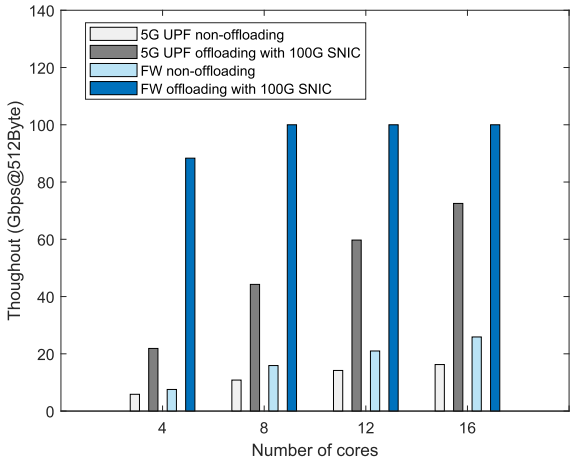


Fig. 10. The performance comparison with 100G smart NIC between firewall and 5GC UPF when offloading.

model from a macro perspective. A typical traffic model of 5G subscriber is set as: 1)the traffic per subscriber is 50kbps during busy hours, 2) the bearer per attached subscriber is 1.6, and 3) attach and detach attempt times per subscriber are all 1 during busy hours. Obviously, the number of subscribers, flow entries and throughput will increase correspondingly. In the test-bed setup of typical application scenarios, each compute node runs on ZXTECS, which is a carrier-grade cloud network platform based on OpenStack and is also a software platform, different from the hardware in NFVI. The 5GC UPF and firewall are selected as the VNF in the test bed because its conditions are similar to commercial test conditions.

Offloading service traffic by smart NIC can reduce CPU resource consumption under the same traffic and effectively solve the high system overhead caused by I/O processing. The ZXCLOUD R5300-G3 server, 2CPU OF Intel Xeon E5-2620 and the 100 GE smart NIC of H3PCX VU9P are used for the system under test. Different number of working threads are enabled to send and receive packets. Figure 10 shows the comparison of performance when offloading with smart network card for two different service types: the firewall (FW) and the 5GC UPF. According to the throughput test in Figure 10, the speedup ratio $S_u(n)$ relative to the 4-cores system can be calculated, and the corresponding speedup ratio curve is fitted as shown in Figure 11.

It can be seen that the bottleneck for the performance of 5GC UPF lies in the total computational scale of the software, and has a higher variation factor f_t of the total computational scale. The corresponding BCE resource need to be allocated more to the processing for the 5GC UPF. Whereas, the bottleneck for the performance of firewall lies in the forwarding capability of the smart network card, more CPU cores beyond 5 cores does nothing to improve throughput. According to Implication 3, performance tuning is closely related to the service model, and the total computational scale is related to the service type. The system design must reasonably allocate the corresponding BCE resource on the basis of predicting the workload ratios according to the service model.

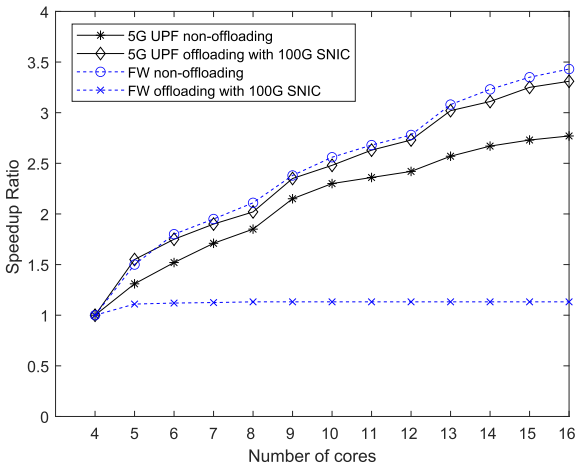


Fig. 11. Fitting curve of speedup ratio relative to the number of cores.

The central issue of how improvement actions lead to a better state is studied in terms of multidimensional optimization techniques in this paper. According to Section II-B, the speedup ratio is closely related to four types of factors. Among these theoretical tuning directions of performance, the recommended improvements comprise: the architecture optimization corresponding to the factor f_t of total calculation scale and the factor f_a of nonparallel parts, the algorithm optimization corresponding to the total calculation scale factor f_t, and the processing flow optimization corresponding to the factor f_i of cumulative overhead.

A systematic benchmarking experiment is used to quantify the comparative NFV acceleration effects reflected in the architecture, processing flow, and algorithms. The test bed uses NFVI ZTETECS as the cloud management platform, and 5G UPF is deployed two R5300-G4X servers (one as the control node and the other as the computing node). There are two configurations available: 1) 2CPU of Intel Xeon Gold 6330N on server with E810-CQDA2*4 NIC, and 2) 2CPU Intel Xeon Platinum 8380 on server with E810-CQD2*6 NIC. The IXIA tester IxNetworks-XGS2 is used to simulate the control plane and user plane services of 5G mobile users. A mobile operator's standard service test model is adopted, as shown in Table V. The performance benchmarking verifies the basic forwarding capability of UPF and the forwarding capability with content billing DPI service processing respectively. When the average CPU utilization of the worker thread is 85%, the benchmarking results are shown in Figure 12. Further, the quantitative comparisons of different dimension optimization techniques are roughly calculated as shown in Figure 13. The width represents the scope of application of the optimization technology. The height represents the contribution to the NFV acceleration ratio, and the higher is better. It can be seen that these tuning directions have a significant improvement on performance, and the architecture and algorithms have a more prominent impact on the NFV acceleration. The sNIC technology has good applicability, while the other technologies have their own corresponding scope of application.

TABLE V
THE STANDARD SERVICE TEST MODEL FOR 5G UPF

Main parameter	Indicators
Number of access subscribers	600,000
Traffic per subscriber	50 kbps during busy hours
Packet size range	64 through 1518 bytes; average 690 bytes
Number of content billing (DPI) rules	L7 protocol: 40,000; L3 and L4: 10,000
Number of control and charging policy	static: 40; dynamic: 10
Traffic ratio	HTTP: 80%, UDP: 20%

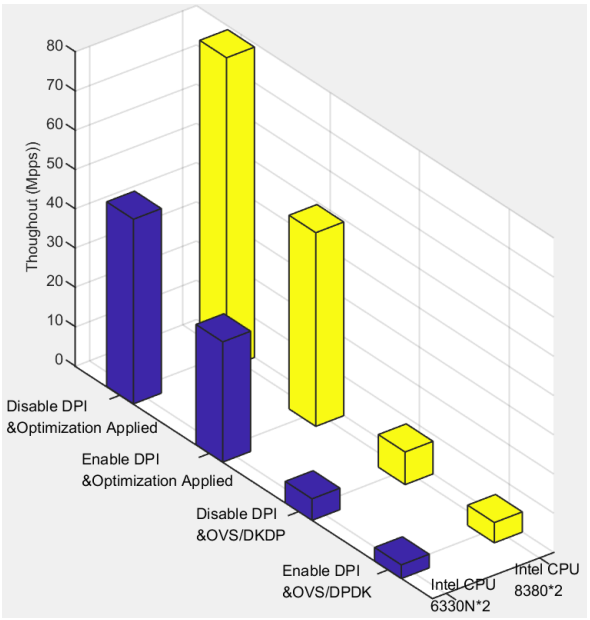


Fig. 12. Performance comparison of 5G UPF.

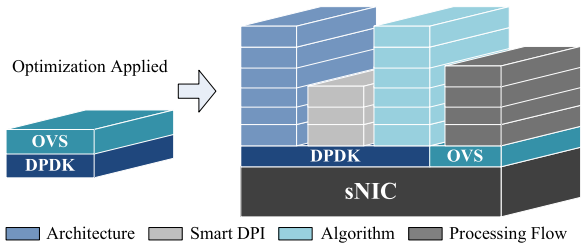


Fig. 13. Effect comparison of key factors for NFV acceleration.

C. The Impact of Memory Hierarchy on the Inter-Core Transfer Overheads and Scalability

The experimental results of Figure 11 show that increasing the user-plane processing core does not linearly increase the forwarding performance improvement. As the number of cores increases, the speedup ratio becomes lower and lower. The interaction processing load inter-cores will become a bottleneck when the cluster processing cores reach a certain amount. A processor usually contains multiple cores, integrated cache, and memory [31]. The software design in the forwarding system should ensure the performance matching between them

TABLE VI
IMPACT OF MEMORY HIERARCHY FOR INTER-CORE TRANSFER OVERHEADS

Data block location	Latency distribution	Transfer overhead (cyc)
Local-node L3-clean	d_{L3}	50
Local-node L3-dirty	d_{L3} + p_c	76
Inter-node L3	d_{L3} + p_c + p_n	98
Local-node DDR	d_{DDR}	155
Inter-node DDR	d_{DDR} + p_n	280

to obtain the optimal forwarding performance and scalability. The north-bridge and south-bridge are two commonly used components in classic CPUs. They are the communication channel among the processor, memory, and other peripherals.

First, by simply increasing the number of processors, the full advantages of multi-cores parallelism are not realized because of the coding quality. The inter-core transfer overheads are evaluated under different experiment settings for both single node and multi-nodes. In the ZXCLOUD R5300 server of the test bed, the general 2Ghz CPU main parameters are shown in TABLE III. The access delays of L3 and DDR are respectively denoted as d_{L3}, d_{DDR}. As shown in Figure 4, a CPU can access the data block faster from components that are within local node. There is a latency penalty when the objects of data block travel across inter-node which we refer to as inter-node penalty, denoted as p_n. Moreover, the cache coherence penalty, denoted as p_c, is a concern in a multi-cores environment. Since each core has its own cache, a copy of the data in that cache may not always be the most up-to-date version. Specifically, the packet inter-core transfer overheads are measured by the waiting time that a CPU core fetches instructions or data. Table VI shows the evaluation. The multi-cores memory hierarchy has significant impact on the forwarding performance and scalability.

Since all CPU cores read L3/DDR memory by sharing a north-bridge. When the north-bridge is congested, all devices and processors are paralyzed. As computing power increases, so must memory bandwidth. If the processor cannot be provided enough data, even more processing cores on a chip will be of no benefit, especially that a performance penalty incurs in the processing of packet forwarding. Packets processed inter-cores must be transferred via the L3 or DDR cache. As the number of cores increases, the data exchange overheads of inter-core or inter-node will increase accordingly, and the performance bottleneck of the north-bridge in response time will become more and more obvious, thus incurring the inter-core transfer performance penalty. For certain CPUs, the careful trade-offs should be made during software coding to avoid the north-bridge congestion.

Second, some features of the software and hardware in server become new bottlenecks, which affect the parallel expansion of performance such as the bus contention and share memory. Whether it is the main processor or the accelerator card, or the hard disk or NIC of the south bridge, it is necessary to frequently access the memory. When these units all scramble to access memory, the competition for north-bridge bandwidth intensifies, and there is only one bus between the north-bridge and the memory. In a COTS for packet forwarding, the access overhead of the NIC or the accelerator card to the memory far exceeds that of a server for general computing purposes, which has a significant impact on the bus communication capability between the north-bridge and the memory. The access mechanism of memory becomes another bottleneck of the system design.

V. CONCLUSION

The software-based packet forwarding performance faces a long-term challenges. NFV acceleration plays a crucial role in the development of NFV. We believe that the effect of performance optimization is ultimately reflected in two aspects at the CPU instruction execution level.

- Increase in eIPC indicators: the code should make full use of the CPU pipelines to increase the number of instructions within a clock cycle that can be executed, e.g., the VPP architecture, loop unrolling, dependence reduction between instructions, and hardware and software prefetching. Then the parallelism can be increased significantly, though the total number of instructions executed at a node is constant.
- Decrease in IPP indicators: the number of instructions required for forwarding processing can be reduced as much as possible by using the efficient instructions, algorithms, or architectural decomposition, e.g., calculating ahead is used instead of calculating every iteration in the loop, node split, traffic offloading, etc. Then, the number of instructions executed by a node is significantly reduced.

The NFV acceleration requires a wealth of experience to quickly identify the bottlenecks, and seek the best measure from numerous technology options. Future works will focus on the enrichment and improvement on the knowledge graph of NFV acceleration, so that the software-based NFV acceleration become controllable and expected with higher efficiency for development.

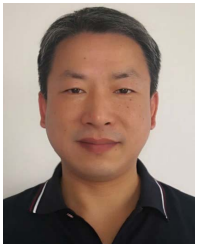
REFERENCES

[1] A. M. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.
[2] V.-G. Nguyen *et al.*, "SDN/NFV-based mobile packet core network architectures: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1567–1602, 3rd Quart., 2017.
[3] M. Satyanarayanan *et al.*, "An open ecosystem for mobile-cloud convergence," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 63–70, Mar. 2015.
[4] L. Linguaglossa *et al.*, "Survey of performance acceleration techniques for network function virtualization," *Proc. IEEE*, vol. 107, no. 4, pp. 746–764, Apr. 2019.
[5] X. Fei *et al.*, "Paving the way for NFV acceleration: A taxonomy, survey and future directions," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–42, Sep. 2020.
[6] A. Brunnert *et al.*, "Performance-oriented DevOps: A research agenda," 2015, *arXiv:1508.04752*.
[7] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May /Jun. 2016.
[8] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 1, pp. 34–47, Mar. 2015.
[9] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[10] S. G. Kulkarni *et al.*, "NFVnice: Dynamic backpressure and scheduling for NFV service chains," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 639–652, Feb. 2020.
[11] R. Kundel, F. Siegmund, R. Hark, A. Rizk, and B. Koldehofe, "Network testing utilizing programmable network hardware," *IEEE Commun. Mag.*, vol. 60, no. 2, pp. 12–17, Feb. 2022.
[12] F. Lou, "Network acceleration and performance improvement," DPDK Summit China, 2017. [Online]. Available: https://www.dpdk.org/wp-content/uploads/sites/35/2018/06/DPDK-China2017-Lou-Network-Performance-Tuning.pdf
[13] J. Pernstål, R. Feldt, and T. Gorschek, "The lean gap: A review of lean approaches to large-scale software systems development," *J. Syst. Softw.*, vol. 86, no. 11, pp. 2797–2821, Nov. 2013.
[14] P. Rodríguez *et al.*, "Building lean thinking in a telecom software development organization: Strengths and challenges," in *Proc. Int. Conf. Softw. and Syst. Process*, 2013, pp. 98–107.
[15] M. Wagner. *Lean NFV Aims to Reignite Virtualization*. Accessed: 2019. [Online]. Available: https://www.lightreading.com/nfv/nfv-specs-open-source/lean-nfv-aims-to-reignite-virtualizatio n/d/d-id/750646
[16] K. Petersen and C. Wohlin, "Software process improvement through the lean measurement (SPI-LEAM) method," *J. Syst. Softw.*, vol. 83, no. 7, pp. 1275–1287, Jul. 2010.
[17] M. Condoluci and T. Mahmoodi, "Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges," *Comput. Netw.*, vol. 146, pp. 65–84, Dec. 2018.
[18] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.
[19] X. H. Sun and L. M. Ni, "Scalable problems and memory-bounded speedup," *J. Parallel Distrib. Comput.*, vol. 19, no. 1, pp. 27–37, Sep. 1993.
[20] Y. Le *et al.*, "UNO: Uniflying host and smart NIC offload for flexible packet processing," in *Proc. Symp. Cloud Comput.*, 2017, pp. 506–519.
[21] P. Stenstrom, T. Joe, and A. Gupta, "Comparative performance evaluation of cache-coherent NUMA and COMA architectures," in *Proc. 19th Annu. Int. Symp. Comput. Archit.*, 1995, pp. 315–326.
[22] P. Caheny, L. Alvarez, S. Derradji, M. Valero, M. Moreto, and M. Casas, "Reducing cache coherence traffic with a NUMA-aware runtime approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 5, pp. 1174–1187, May 2018.
[23] G. Fursin *et al.*, "Collective mind: Towards practical and collaborative auto-tuning," *Sci. Program.*, vol. 22, no. 4, pp. 309–329, 2014.
[24] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi, "High-speed software data plane via vectorized packet processing," *IEEE Commun. Mag.*, vol. 56, no. 12, pp. 97–103, Dec. 2018.
[25] B. Calder and G. Reinman, "A comparative survey of load speculation architectures," *J. Instruct., Level Parallelism*, vol. 2, pp. 1–39, 2000.
[26] B. Jenkins, "Hash functions," *Dr. Dobb's J.*, Sep. 1997. [Online]. Available: http://www.burtleburtle.net/bob/hash/doobs.html
[27] M. B. Rutzig *et al.*, "TLP and ILP exploitation through a reconfigurable multiprocessor system," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum (IPDPSW)*, Apr. 2010, pp. 1–8.
[28] J. L. Lo, J. S. Emer, H. M. Levy, R. L. Stamm, D. M. Tullsen, and S. J. Eggers, "Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading," *ACM Trans. Comput. Syst.*, vol. 15, no. 3, pp. 322–354, Aug. 1997.
[29] P. Zheng, A. Narayanan, and Z.-L. Zhang, "A closer look at NFV execution models," in *Proc. 3rd Asia–Pacific Workshop Netw.*, Aug. 2019, pp. 85–91.
[30] (2019). *Intel VTune Amplifier*. [Online]. Available: https://software.intel.com/en-us/vtune
[31] R. Balasubramonian, N. P. Jouppi, and N. Muralimanohar, *Multi-Core Cache Hierarchies*. Williston, VT, USA: Morgan & Claypool, Nov. 2011.

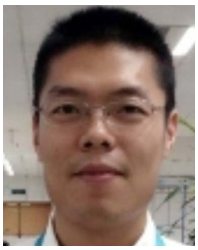
Qiang Wu (Member, IEEE) is currently a Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. Before that, he worked as a member of the Academic Committee of State Key Laboratory of Mobile Networks and Mobile Multimedia Technology, ZTE Corporation, China. He has more than 100 authorized patents, of which nearly 20 patents correspond to international standards. His research interests include mobile networks, industrial internet, integration of satellite-terrestrial networks, and cyber security. He is a Fellow of CICC. The Chinese government honored him with the Second-Class National Science and Technology Progress Award in 2009 and the Second-Class National Technology Innovation Award in 2014.

Xiangping Bryce Zhai (Member, IEEE) received the B.Eng. degree in computer science and technology from Shandong University in 2006 and the Ph.D. degree in computer science from the City University of Hong Kong in 2013. He is currently an Associate Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. He is also with the Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, China. Previously, he was a Post-Doctoral Fellow with the City University of Hong Kong. His research interests include the Internet of Things, power control, edge computing, resource optimization, and spatial analytics. He has been actively involved in organizing and chairing several international conferences, and has served as reviewer for several journals.



Xi Liu received the B.E. and Ph.D. degrees from the Army Engineering University of PLA, Nanjing, China, in 1995 and 2013, respectively. He is currently an Associate Professor with the College of Communications Engineering, Army Engineering University of PLA. The main research interests include computer networking, SDN, and data communications.

Chun-Ming Wu received the Ph.D. degree in computer science from Zhejiang University in 1995. He is currently a Professor with the College of Computer Science and Technology, Zhejiang University. He is also the Associate Director of the Research Institute of Computer System Architecture and Network Security, Zhejiang University; and the Director of the NGNT Laboratory. His research fields include reconfigurable networks, networks security, and next-generation networks infrastructures.



Fangliang Lou is a Systems Architecture Expert with the State Key Laboratory of Mobile Networks and Mobile Multimedia Technology, ZTE Corporation, Nanjing, China. He focuses on the cut edge research and product implementation of packet forwarding technology in packet communication equipment, and has over 20 years of experience in the research and development field of telecommunication.

Hongke Zhang (Fellow, IEEE) received the Ph.D. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 1992. He was elected as an academician of the Chinese Academy of Engineering in November 2021. He is currently a Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China, where he currently directs the National Engineering Laboratory on Internet Technology for Next-Generation Internet, China. His research has resulted in many research papers, books, patents, systems, and equipment in the areas of communications and computer networks.