NUMASFP: NUMA-Aware Dynamic Service Function Chain Placement in Multi-Core Servers

Venkatarami Reddy Chintapalli, Sai Balaram Korrapati, Bheemarjuna Reddy Tamma, Antony Franklin A
Indian Institute of Technology Hyderabad - INDIA
Email: {cs17resch01007, es18btech11011, tbr, antony.franklin}@iith.ac.in

***Abstract*—Many prior works investigated service function chain (SFC) placement in multi-core NFV servers with an assumption of treating all CPU cores equal. However, sophisticated servers follow non-uniform memory access (NUMA) architecture in which CPU cores are distributed across different NUMA nodes to enhance scalability. Our preliminary experiments show that placing VNFs in arbitrarily selected cores can degrade SFC performance, owing to contention for system resources at intra-node and inter-node levels. To handle the aforementioned problem, we study the impact of placing SFCs in a server by jointly considering the discrepancy of cores in different NUMA nodes and carefully allocating shared resources such as last level cache and memory bandwidth. We first propose a mathematical formulation, to optimally choose a node to place each SFC and allocate required shared system resources, in the form of a non-linear integer programming (NLIP) problem. Then we develop *NUMASFP*, a NUMA-aware dynamic SFC placement mechanism that places SFCs inside a server and migrates SFCs among NUMA nodes based on their current traffic rates to maximize the aggregate throughput of all SFCs. Furthermore, it ensures performance isolation for SFCs that are running inside the server. We evaluate the performance of *NUMASFP* by implementing it on a prototype system and also conducting large-scale simulation studies. The performance results demonstrate that the proposed mechanism can effectively handle SFCs over state-of-the-art and baseline approaches.**

I. Introduction

Network Functions Virtualization (NFV) is introduced to replace the conventional middleboxes (e.g., firewall, deep packet inspector, etc.) by their software counterparts known as Virtual Network Functions (VNFs) which run on general-purpose hardware platforms and promise several benefits such as reduced cost, ease of deployment, and flexibility. Despite several benefits, the extra overhead introduced by the virtualization layer causes performance degradation in terms of latency and throughput, which is not admissible for various network services [1]–[3]. The packets of a network service are processed by a sequence of VNFs, which forms a Service Function Chain (SFC) [4].

Fig. 1 shows the architecture of a modern multi-core system composed of several multi-core processors called nodes, in which one or multiple SFCs can be deployed. Most of the related works on SFC placement deploy multiple VNF instances on the same server to save energy and improve resource utilization [5]–[7]. However, when placing the SFCs inside a server, these works overlooked two factors that have a greater impact on the performance in terms of throughput. The *first* factor is the effect of inter-node resource contention due to cross-node memory access bottleneck. Aforementioned works

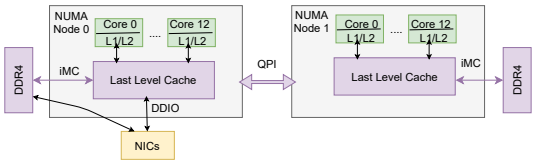


Figure 1: Multi-core NUMA server architecture.

treat all CPU cores as equal and place VNFs randomly inside nodes. However, CPU cores in multi-core systems are not equal due to NUMA (Non-Uniform Memory Access) architecture. When VNFs from the same SFC are placed in cores of different NUMA nodes in a server, the performance deteriorates due to cross-node memory access bottleneck. A cross-node memory access bottleneck occurs when CPU cores in one node access memory in another node via Quick Path Interconnect (QPI). The *second* factor is the effect of intra-node resource contention. When all the VNFs of the same SFC are placed in the same node to avoid cross-node memory access bottleneck, the performance degrades significantly due to intra-node shared resource contention, primarily due to Last Level Cache (LLC), integrated Memory Controller (iMC) (which connects LLC and memory with high speed Memory Bandwidth (MB)), and QPI.

Recently, studies in [8]–[11] demonstrated that the inter-node resource contention has a significant impact on the performance degradation, and the authors of [8] addressed it by placing all VNFs of an SFC in the same NUMA node. However, these works ignored guaranteeing performance isolation [1] (which helps to SFCs to meet their SLAs (e.g., throughput, latency)) among the running SFCs. Furthermore, the authors did not consider the impact of dynamic variation in SFC traffic. Experimental results show that arbitrarily selecting cores to place VNFs of an SFC can result in up to 52% lower throughput when compared to the optimal placement (i.e., placing the entire SFC in the same node). The performance of an SFC is also affected by the input traffic rate. Furthermore, we find that migration of SFCs among NUMA nodes in a server based on their traffic rates provides a significant improvement in the aggregate throughput of all SFCs with negligible overhead.

On the other hand, several recent works [12]–[16] presented that VNFs running in the same NUMA node experience intra-node resource contention, particularly due to LLC contentions, resulting in significant performance degradation for various VNFs. Our experimental study shows that intra-node resource contention results in up to 45% throughput degradation com-

pared to running that VNF alone. To address this intra-node contention, aforementioned works used the Intel's Cache Allocation Technology (CAT) tool [17] to partition the LLC cache ways and assign it to VNFs running in a node. However, these studies ignored the impact of Memory Bandwidth (MB) contention for VNFs due to LLC allocations, and we attempt to demonstrate that it causes up to 20% reduction in throughput. We also demonstrate how it can be addressed by jointly allocating MB and LLC ways to VNFs using Intel's recently introduced resource partitioning tools i.e., Memory Bandwidth Allocation (MBA) [18] and CAT.

In this paper, we propose *NUMASFP*, a mechanism for NUMA-aware dynamic SFC placement with the objective of maximizing the aggregate throughput of all SFCs deployed on a general purpose processor (GPP) server. It guarantees performance isolation among the SFCs and thereby avoids performance degradation. It performs three main tasks: *first*, it chooses a node to place SFC; *second*, it allocates resources (i.e., LLC and MB) to each SFC to ensure performance isolation; and *finally*, it dynamically migrates SFCs between nodes based on their traffic rates. We profile each VNF to determine the required amount of resources (i.e., LLC and MB) based on its traffic rate. To migrate VNFs between cores on a server, we use the new docker feature *cpuset* [19]. The main contributions of this work are as follows:

- We showcase the impact of inter-node and intra-node resource contentions in a multi-core system on SFC performance and also highlight the importance of allocating MB along with LLC to guarantee performance isolation.
- We formulate the Dynamic SFCs Placement Problem (DSPP) as a Non-Linear Integer Programming (NLIP) problem with the objective of maximizing aggregate throughput of deployed SFCs in a server.
- We propose a NUMA-aware dynamic SFC placement mechanism (*NUMASFP*) to solve the DSPP problem and implement it on OpenNetVM [20], a high-performance container-based platform for NFV.
- Through extensive evaluations, we show how the proposed mechanism outperforms state of the art and baseline approaches.

II. Background and Motivation

In this section, first, we provide background details on the NUMA architecture and Intel RDT tools. Next, we present results highlighting the importance of addressing intra-node and inter-node contentions in order to ensure performance isolation for the SFCs deployed in a server.

A. NUMA Architecture

Most of the GPP servers available in the market are based on NUMA architecture in which multiple CPU cores are grouped into nodes. NUMA architecture has many advantages such as reduced cost and increased scalability [21]. Each node has its own memory known as local memory and an advanced memory controller that allows it to access memory on all other nodes. For a node, local memory of another node is remote and accessing the remote memory is slower than accessing the local memory, as data must be transferred from the remote memory to the local memory through NUMA interconnect. A *cross-node memory access bottleneck* occurs when CPU cores in one node access memory in another node via QPI.

Each node uses iMC to connect to the local channels of DDR4 memory as shown in Fig. 1. Accessing the physical memory connected to a remote iMC is called remote memory access. The QPI interfaces are responsible for transferring data between two nodes.

B. Intel RDT Tools

Intel RDT [22] provides the capability to partition system resources like LLC and MB in a way that their usage can be restricted to co-located applications, containers, and Virtual Machines (VMs). Cache partitioning capability is known as Cache Allocation Technology (CAT) [17] and bandwidth limiting capability is called as Memory Bandwidth Allocation (MBA) [18].

Cache Allocation Technology (CAT): CAT enables partitioning of the LLC and allocates the LLC partitions to specific cores of the node. CAT introduces 16 Class of Services (CLOS) to entitle cache partitioning, and a core or a set of cores can be associated with a specific CLOS. The available cache ways/partitions can be dynamically allocated to each CLOS using Capacity Bit Mask (CBM), which indicates how much cache can be used by each CLOS. Cache partitioning based on CAT can be configured with the help of *pqos* [23] program or by using *libpqos* libraries. Note that, the number of cache partitions is architecture-dependent.

Memory Bandwidth Allocation (MBA): The MBA feature provides indirect and approximate control over MB that is available per core. It actually throttles the outgoing traffic from the L2 cache to the LLC by injecting a delay to each outgoing request. The MB throttling value is defined as a percentage and the upper-bound and granularity of MBA are machine dependent. It provides a method to control VNFs which may be over-utilizing bandwidth relative to their priorities and thus improves the performance of high priority VNFs. Like CAT, CLOS is used here too for throttling the MB for different cores. MBA levels in terms of percentage can be assigned to the required CLOS dynamically up to 90% in steps of 10%.

C. Motivation

Fig. 2 shows the setup used for conducting experiments with two main components: System Under Test (SUT) and packet generator. These components are running on similar type of servers with two Intel Xeon Gold 6126 48-core CPUs at 2.60GHz, 98 GB memory, with Ubuntu 18.04 with kernel 4.4.0-131-generic. Each CPU has 19.7 MB size LLC with eleven LLC ways that can be partitioned among services through CAT. Servers are connected in back-to-back manner with dual-port 10 Gbps (20 Gbps total) DPDK compatible NICs to reduce switch overheads. These two NICs are installed on the NUMA $Node\ 0$, which we refer to as local node, and the other node as remote ($Node\ 1$). We use Pktgen [24], a DPDK-based

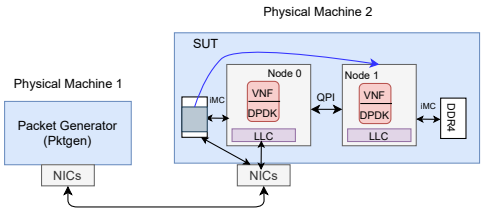


Figure 2: The experimental setup used for measuring the impact of interference.

high-speed traffic generator, for generating traffic rates. On the SUT, we installed OpenNetVM platform on the local node and deployed specific DPDK enabled VNF (called target VNF) or sequence of connected VNFs (called SFC) which receives input traffic, processes it, and sends back to the sender. Each VNF runs on a dedicated core and uses memory attached to the local node. Our study involved different experimental scenarios, where for each scenario, we measure the throughput of the target VNF or SFC by varying traffic rates ($64B$ packet size) from 1 $Gbps$ to 10 $Gbps$. Experiments for each case is repeated 10 times and the average results are presented. Moreover, it needs to be stated that each experiment is run for 60 $secs$. In these experiments, performance results are presented in the absence of intra-node contention.

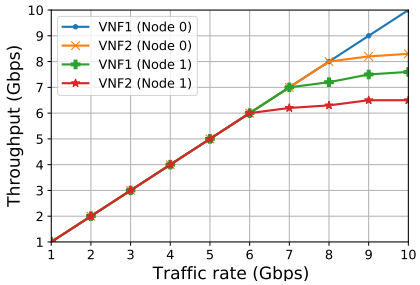


Figure 3: Variation in VNF throughput vs. input traffic rate for different placements of the target VNF.

1) Performance Impact of SFC Placement in a Server: **NUMA Impact - Minimal VNF:** In the first scenario, VNF is placed on $Node$ 0, whereas in the second scenario, VNF is placed on $Node$ 1. Fig. 3 depicts the observed throughput of the target VNFs (VNF1: Basic Monitor (BM); VNF2: Firewall (FW)) [25] for traffic rates ranging from 1 $Gbps$ to 10 $Gbps$. The throughput of these VNFs has reduced significantly for higher input traffic rates when running in $Node$ 1 than when they are running in $Node$ 0. This is due to the overhead involved in remote memory access. Furthermore, we also observe the performance of a target VNF depends on its functionality. **Observation 1:** It is important to select a right node when deploying a VNF instance in a server. A VNF performance is affected by its functionality, incoming traffic, and the node on which it is running. One may need to migrate VNFs between nodes based on traffic rates to increase their throughput and thereby meeting the SLAs.

We conduct another experiment to see how VNF migration among nodes affects the performance (refer Fig. 4). We start

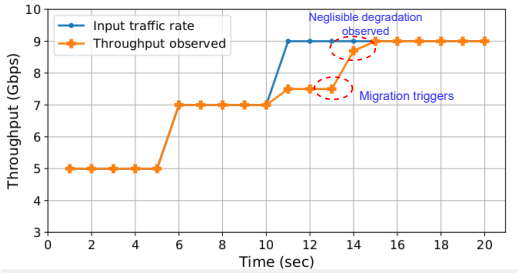


Figure 4: The effect of VNF migration between NUMA nodes on VNF performance in a server.

by running a BM VNF alone in the remote node. The target VNF achieves the maximum throughput for the first 10 $secs$ when the offered load is below 7 $Gbps$; however, its throughput degrades as its input traffic rate increases to 9 $Gbps$ at 11 $secs$. We migrate[1] the target VNF to the local node at 13 $secs$ and notice seamless migration with a minimal performance impact in the immediate second. This is due to the availability of high speed bandwidth between NUMA nodes (19.2 GB/s in the SUT). From this, we recommend migrating VNFs among NUMA nodes based on traffic rates to provide a significant improvement in the aggregate throughput of all VNFs with a negligible overhead.

NUMA Impact - Service Function Chain: The impact of NUMA placement on SFC performance is depicted in the following scenario. We place an SFC of three VNFs (VNF1: BM, VNF2: Router (RT), and VNF3: Simple Forward (SF)) [25] at two possible locations ($Node$ 0 and/or $Node$ 1), resulting in a total of nine possible placements. For example, combination $0-1-0$ represents a placement in which BM is placed on $Node$ 0, RT is placed on $Node$ 1, and SF is placed on $Node$ 0. Since no packet copying is required between the nodes in case of $0-0-0$, it is expected to be the best-case scenario. Placement $1-0-1$, on the other hand, is expected to be the worst case, as each packet must traverse through cross-node four times, adding a lot of packet copying overhead and resulting in the lowest throughput.

Fig. 5 shows the throughput observed for four different SFC placements. It is observed that arbitrarily selecting cores to place VNFs in an SFC can result in 52% lower throughput compared to the optimal placement combination ($0-0-0$). Best performance can be obtained either by placing entire SFC in remote or local node. When SFC is placed in the remote node ($1-1-1$), the performance degrades by 22% due to remote memory access overhead. On the other hand, when we place some VNFs on the local and others on the remote node ($1-0-1$), we see 52% throughput degradation when compared to the optimal. This effect can be seen due to both excessive packet copying between nodes and remote memory access overhead. By this, we can say that cross-node memory bottleneck causes significant performance degradation in addition to remote memory access overhead.

Observation 2: It is always recommended to place all VNFs

[1]We used *cpuset* feature in docker to migrate the target VNF between cores.

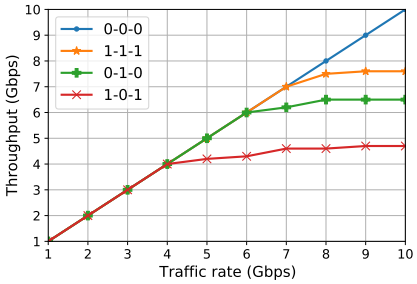
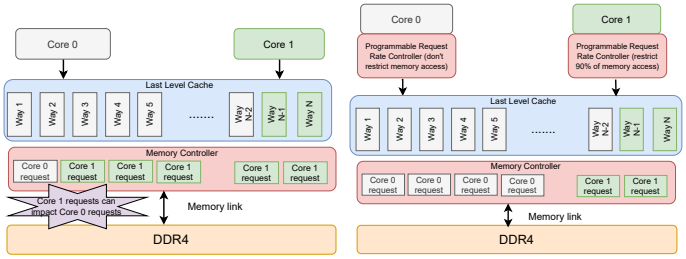


Figure 5: Variation in SFC throughput vs. input traffic rate for different VNF placement combinations.

of an SFC in the local node if it has enough cores; otherwise, the entire SFC should be placed in the remote node rather than splitting VNFs of the SFC among nodes of the server.

It is challenging to guarantee performance isolation to SFCs in the above scenario, regardless of the node used for SFC placement. It all depends on how aggressively the co-located services are accessing shared resources such as LLC and MB.



((a)) Controlling LLC only (using CAT). More cache ways are given to VNF service running in $Core$ 0 as it is the high priority one. The other service running in $Core$ 1 gets fewer cache ways, resulting in highly frequent cache misses and saturating the memory link.

((b)) Controlling LLC as well as memory bandwidth. Memory link access for the VNF service running in $Core$ 1 is restricted by using MBA support to ensure that the high priority NF service indeed gets high priority.

Figure 6: Importance of Intel RDT tools (CAT and MBA).

2) Towards Guaranteeing Performance Isolation : **Importance of MB along with LLC**: Here, we demonstrate the impact of intra-node resource contention when a target VNF is co-located with other workloads in the same node. We experimented with Firewall VNF [25] with Stress-ng [26] as its noisy neighbor, which causes a lot of LLC misses and MB usage to induce sufficient stress on the memory subsystem. To avoid contention for CPU cycles, both Firewall and the noisy neighbor are configured to run on dedicated cores.

Most recent works [13]–[16] in NFV have shown the impact of LLC contention and overcome it by using Intel's CAT tool. In this work, we show the importance of controlling MB along with LLC in order to achieve performance isolation. The problem of allocating LLC ways only to the co-located VNFs is demonstrated in Fig. 6(a). If we do not control MB along with LLC, the workload which is given less LLC ways will contend more for MB and leads to performance degradation for the VNF which is given more LLC ways. It means, sometimes

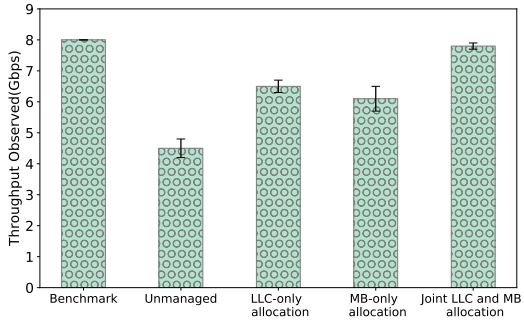


Figure 7: Throughput observed for Firewall VNF for various experimental scenarios (VNF is running in $Node$ 0).

even when we allocate more LLC ways, it does not ensure performance isolation instead it causes another problem. In order to achieve performance isolation, we can use CAT and MBA to partition LLC and MB, respectively. They together help to perform resource allocation to the co-located VNFs and other services, and thereby improve the performance of high priority VNF as shown in Fig. 6(b).

Fig. 7 shows the importance of jointly allocating LLC and MB in order to meet performance requirements when a target runs alone in the server compared to *no resource allocation scenario* (*Unmanaged*: octans [8]) and *single resource allocating scenarios*. The amount of resources allocated is based on information obtained from the VNF profiling which is explained in the following section. We observe that Firewall VNF achieves maximum throughput up to 8 $Gbps$ of input traffic rate. From profiling, we find that with the minimum combination of five LLC-ways and 50% MB allocation in the server, this VNF exhibits maximum performance (for 8 $Gbps$ input traffic rate) as that of when it is deployed alone in the server (*benchmark*). We allocate five LLC-ways and 50% of MB for the scenarios *LLC-only* and *MB-only*, respectively and leave other resource unallocated. When the resources are not allocated to the target VNF, its performance drops by 45% due to intra-node resource contention compared to when the target VNF is deployed alone. We observe that allocating LLC only does not always result in performance isolation due to bandwidth saturation and it could result in 20% reduction in throughput. We repeated each experiment 10 times and plotted the results with 95% confidence.

Observation 3: The performance of a VNF is highly dependent on both LLC and MB partitioning. In order to ensure performance isolation, the allocation of MB along with LLC is of paramount importance in GPP servers.

III. VNF Profiling

In this section, we briefly describe the adopted VNF profiling procedure and provide insights on observed performance. We profile a variety of VNFs available from OpenNetVM. The main goal of the VNF profiling is to efficiently profile each of the VNFs, understand its performance characteristics, and build a lookup table, thereby helping the NFV orchestrator effectively allocating various system resources.

Table I: Minimum resource combinations for two VNFs to achieve required throughput at local and remote nodes

VNF	Throughput (Gbps)	1-4	5	6-7	8	9	10
Router	Local	(2,10)	(2,20)	(3,20)	(3,30)	(4,30)	(5,40)
	Remote	(3,20)	(3,30)	(5,30)	—	—	—
Basic Monitor	Local	(2,10)	(2,20)	(2,30)	(4,30)	(4,30)	(4,50)
	Remote	(3,20)	(3,30)	(4,40)	—	—	—

The VNF profiler begins by constructing throughput vs traffic rate curves as a function of allocated LLC ways and MB. To build this curve for a VNF, the profiler runs the VNF alone on the SUT by pinning it on a dedicated core(s) and, at each step, increasing the allocated LLC ways and MB. From the constructed curves, the maximum achievable throughput for a combination of LLC ways and MB can be identified.

Table I shows the lookup table built for Router (RT) and Basic Monitor (BM) VNFs. The VNF resource requirement to achieve corresponding throughput is defined as a tuple (l_i, m_i), where l_i and m_i denote the LLC ways and the MB level allocated to the VNF, respectively. The experiments exhibit that the combination of resources needed to achieve the guaranteed performance varies among VNFs. Moreover, it also depends upon the incoming traffic rate at the target VNF.

Observation 4: As the VNFs face varying traffic conditions during their lifespan, it is critical to address the dynamic allocation of LLC and MB to maximize the aggregate throughput of SFCs deployed on a GPP server.

In addition, Table I shows that the amount of resources required is lower when a VNF is deployed in the local node than when it is on the remote node. This is due to Data Direct I/O (DDIO) feature [27]. The DDIO enables NICs to copy packets to the LLC of the local NUMA node rather than going all the way to the main memory and back when the application reads the packets. This is only possible on the NUMA node to which the NIC is connected, not on the remote NUMA node. As a result, running a VNF in a remote node consumes more resources such as LLC and MB. Furthermore, due to remote memory access, VNF achieves less throughput when running at a remote node compared to when running at the local node.

Observation 5: VNF performs better when it is deployed on the local node compared to the deployment on a remote node. When all VNFs of an SFC are placed in the same node, its resource requirement is approximately equal to the maximum of resource requirements of its constituent VNFs. This is because all VNFs of the SFC share the same LLC on the node.

To illustrate how this works, consider an SFC of length two (VNF1: RT; VNF2: BM) with a traffic rate of 7 *Gbps* that is to be placed in the server. We notice that if it is placed on the local node, the required resources are *max((3,20), (2,30)) = (3,30)*. *max* function returns the maximum value of each resource in the list of entries. The number of resources required, if it is placed on the remote node, is *max((5,30),(4,40)) = (5,40)*. This is because of all VNFs share the same LLC. This approach is known as *Max_alloc*. When resources are allocated to each VNF independently, as in work [13], SFC uses the sum of resources of all VNFs and hence consumes so many

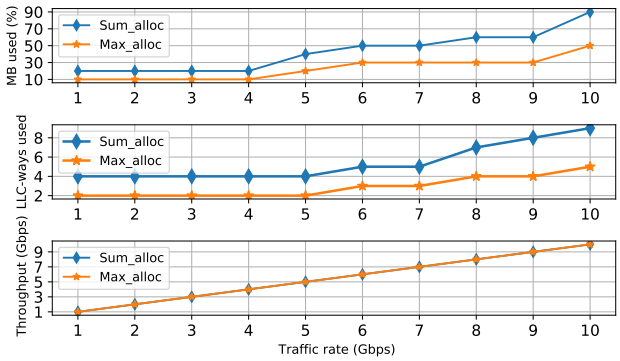


Figure 8: Variations in LLC and MB using *Sum_alloc* and *Max_alloc* approaches (SFC is deployed on the local node(0)).

resources of the node. We refer this approach as *Sum_alloc*. In Fig. 8, the bottom, middle, and top plots depict the throughput achieved, LLC ways allocated, and MB consumed using these two approaches at input traffic rates ranging from 1 $Gbps$ to 10 $Gbps$. Indeed, placing the entire SFC in the same node saves many system resources while not affecting its performance.

To the best of our knowledge, no existing work considered the joint allocation of system resources such as LLC and MB to the VNFs of SFCs along with choosing the right NUMA node inside a GPP server. In short, to design NUMA-aware dynamic SFC placement mechanisms, we have to take into account the impact of NUMA and traffic rate, and figure out which SFC needs to be migrated in order to improve aggregate throughput of SFCs deployed on GPP servers.

IV. NUMASFP Architecture and Problem Formulation

In this section, we present *NUMASFP* architecture followed by problem formulation.

A. NUMASFP Architecture

NUMASFP is built on OpenNetVM platform and main components of its system architecture are depicted in Fig. 9. When a packet arrives at the NIC, the RX thread reads the quintuple, which includes the IP protocol, IP addresses, and ports of both source and destination. The hash of it is then determined to look up the **Flow Table** (which maintains all VNF Ids of each SFC) to steer the traffic to the corresponding VNF. Based on the position of the VNF in the SFC, the packet

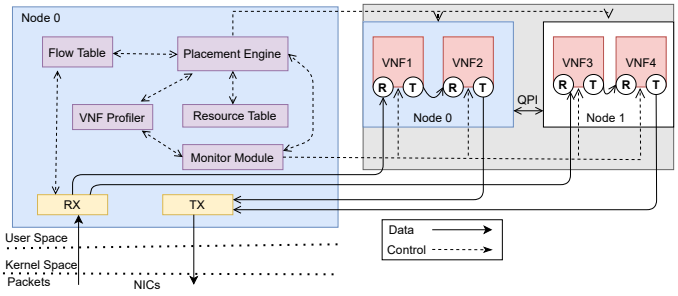


Figure 9: NUMASFP Architecture.

can be sent to the next VNF or out through a port after the VNF has processed it. The main module of *NUMASFP* is **Placement Engine**, which collects the traffic rate of each SFC from the **Monitor Module** as well as the resource requirements from the **VNF Profiler** on a continuous basis and dynamically allocates them system resources. While the Monitor Module keeps track of the input traffic rate of each SFC, the VNF Profiler efficiently profiles each of the VNFs to understand their performance characteristics and builds the lookup table as described in Section III. The lookup table built by the VNF Profiler helps the Placement Engine to allocate various system resources efficiently. The Placement Engine also migrates SFCs (i.e., all VNFs of a chosen SFC) among NUMA nodes to maximize the aggregate throughput of all SFCs deployed on the server. **Resource Table** maintains the resources information (in terms of cores, number of LLC-ways, and percentage of MB) that the Placement Engine has allocated to each SFC. As the SFCs face varying traffic conditions during their lifespan, it is necessary to adaptively allocate LLC and MB to them and migrate them among nodes to maximize the aggregate throughput. We call this problem a Dynamic SFCs Placement Problem (DSPP).

B. Problem Formulation

In this section, we formulate the DSPP problem as a Non-Linear Integer Programming (NLIP) model. We assume NUMA architecture described earlier where there are K nodes in the GPP server with C cores per node. A set of N SFCs needs to be deployed, where each SFC is composed of an array of chained VNFs. We assume that each VNF runs in a dedicated core to avoid CPU contention, and all VNFs belonging to a given SFC are placed in the same node. Each NUMA node ($k \in K$) has the same number of resources R_{kj}, where j represents CPU Cores (C), LLC (L), or MB (M). Besides, all NICs are assumed to be connected to the local node. Let x_{ik} be a binary variable representing if SFC $i \in N$ placed in node $k \in K$. Resources required by SFC i when it is placed in node k based on its traffic rate is defined as a^k_{ij}, where $j \in \{C, L, M\}$. Resource information is obtained by profiling which is done in offline as described in Section III. Let P_i be the throughput of SFC i when it is placed in the local node, which is the optimal achievable value. Let θ_{ik} be the performance drop rate when SFC i is placed in a remote node k. It is obvious that the performance drop is a referred value of optimal performance. Let ϕ_i be the throughput of SFC i at a particular time instance and it should be noted that the objective is to maximize the aggregate throughput of all SFCs deployed on different nodes of the GPP server. Hence, it is formulated as:

$$\max \sum_{i \in N} \phi_i \tag{1}$$

where,

$$\phi_i = x_{ik} * P_i * (1 - \theta_{ik}); \quad \forall k \in K, \ i \in N \tag{2}$$

subject to the following constraints:

$$\sum_{i \in N} a^k_{ij} * x_{ik} \leq R_{kj}; \quad j \in \{C, L, M\}, \ \forall k \in K, \ i \in N \tag{3}$$

$$\sum_{k \in K} x_{ik} \leq 1; \quad \forall \ k \in K, i \in N \tag{4}$$

$$x_{ik} \in \{0, 1\}; \quad \forall \ k \in K, i \in N \tag{5}$$

Eqn. (1) defines the optimization objective which is maximising the aggregate throughput of all the SFCs running in the server. Eqn. (2) defines the relationship between optimal performance and performance drop of SFCs when run in remote nodes in the server. Eqn. (3) states that the resource consumed should not exceed the capacity of many-core system during deployment. Eqn. (4) specifies that the SFC should be placed only once in the server. Eqn. (5) indicates whether SFC i is placed in node k or not. Since the optimal placement of SFCs is known to be NP-hard [8], the NLIP-based approach has a limitation of scalability due to its high computational complexity.

V. Heuristic Placement Algorithm

In this section, we present the proposed heuristic algorithm followed by its complexity analysis.

A. Heuristic Algorithm

Since DSPP is NP-hard problem, it cannot scale well, especially when the problem size increases. Algorithm 1 describes the proposed heuristic algorithm for DSPP, which is implemented in the placement engine module. Table II shows the definitions of all the variables used in the algorithms. The proposed algorithm has two major steps:
(1) *Initial placement:* It starts by locating the resources needed for each SFC in both local and remote nodes and sorts SFCs in descending order based on their traffic rates and begins placing SFCs in the local node until all of its resources are depleted. Then it attempts to place on a remote node (steps 3-7). It is based on the important empirical observations that the performance drops drastically as the number of cross-node packet copies in the SFC chain increases.
(2) *Dynamic SFC placement:* Since the input traffic rate of SFC changes over time, the required resources must be reconfigured rapidly to meet the performance guarantees of SFCs.

Based on the ingress traffic rates of SFCs at time interval t, step 11 computes the required resources for each SFC from the profiled data and stores them in matrix G. Firstly, resources are adjusted based on their availability to the SFCs where they are running, using procedure $resource\_adjustment()$. Because of the resource constraints at each node, even after adjusting the resources, some SFCs may still fail to meet the performance guarantees; thus, we do the SFC migrations between NUMA nodes. Steps 13-24 check for performance violations for each SFC and move an SFC to the list L_mig if it can be migrated to the remote node else to the list R_mig if it can be migrated to the local node. The number of consecutive performance violations is used as a threshold parameter to determine if an SFC needs to be migrated. In our experiments, we set this value to 3. If the R_mig list is not empty, the placement engine invokes the dynamic migration algorithm, in which SFCs from the remote node are migrated to the local node and vice versa.

Algorithm 1: Heuristic Placement Algorithm

Input : Ingress traffic rates of SFC's: λ_i, $\forall i \in N$
Output: Aggregate throughput of all SFC's

begin
  // Initial placement
  Find the required resources for each SFC in local and remote nodes
  Sort SFCs in descending order based on the their traffic rates
  for *each SFC* **do**
    Place in the local node by allocating resources needed. If it is not feasible, place it on the remote node
  end
  // Dynamic placement
  $violation_i \leftarrow 0,\ \forall i = \{1, 2, \ldots, N\}$
  for $t = 1, 2, \ldots, T$ **do**
    G= Compute required resources using profiled data based on $\lambda^{(t)} = \{\lambda_1^{(t)}, \lambda_2^{(t)}, \ldots, \lambda_{|N|}^{(t)}\}$
    $resource\_adjustment(G)$
    for $i = 1, 2, \ldots, N$ **do**
      if $(\phi_i \; != \lambda_i)$ **then**
        $violation_i \leftarrow violation_i + 1$
      else
        $violation_i \leftarrow 0$
      end
      if $(violation_i > max\_limit \; and \; Remote_i)$ **then**
        $R\_mig.add(i)$
      else if $(\phi_i < max\_remote \; and \; Local_i)$ **then**
        $L\_mig.add(i)$
      end
    end
    if $(R\_mig.size() != 0)$ **then**
      dynamic_placement(R_mig, L_mig)
    end
  end
end

Table II: Description of variables used in Algorithms

Variable	Variable Description
max_limit	Maximum limit on number of violations
max_remote	Maximum throughput for SFC when placed in remote node
$Local_i$	Boolean vector which indicates SFC_i is placed in local or not
$Remote_i$	Boolean vector which indicates SFC_i placed in remote or not
$violation_i$	Vector that accumulates count for number of violations of SFC_i
$swap_i$	Boolean which indicates if SFC_i has already been migrated from local to remote node
L_mig	List of SFCs running in local node and which can be migrated to remote node
R_mig	List of SFCs running in remote node and which can be migrated to local node

The procedure used for the dynamic migration of SFCs among nodes is given in Algorithm 2.

In dynamic placement, R_mig is sorted in descending order based on the difference in traffic rates between the previous and current intervals to achieve the highest aggregate throughput. L_mig is sorted in ascending order based on the current traffic rates of those SFCs in the list. Next, for each SFC in R_mig, we must determine which SFC in L_mig can be migrated by checking the corresponding resources by calling $feasible()$ routine. Considering the current traffic rates for the SFCs, the resource requirement is compared accordingly along with the available resources to check feasibility of migration from remote to local and vice versa.

B. Complexity Analysis

In Algorithm 1, we first sort SFCs based on their traffic rate and place them in nodes. Thus, the time complexity of initial placement is $\mathcal{O}(N \times \log N)$, where N is the number of SFCs.

Algorithm 2: dynamic_placement(R_mig, L_mig)

begin
  Sort SFCs in R_mig in descending order based on the maximum traffic rate difference between consecutive time intervals
  Sort SFCs in L_mig in ascending order of traffic rates
  $swap_k \leftarrow false, \forall\ k = \{1, 2, \ldots, |L\_mig|\}$
  for $i = 1, 2, \ldots, |R\_mig|$ **do**
    for $j = 1, 2, \ldots, |L\_mig|$ **do**
      if $(feasible(R\_mig_i, L\_mig_j, swap_j))$ **then**
        Migrate R_mig_i to local and L_mig_j to remote and allocate resources accordingly
        $violation_{R\_mig_i} \leftarrow 0$
        $swap_{L\_mig_j} \leftarrow true$
        break
      end
    end
  end
end

Algorithm 2 addresses dynamic placement of SFCs by trying to migrate SFCs among nodes and allocating resources available at each time interval based on traffic rates. As a result, the time complexity is determined by the number of SFCs and the number of nodes in the server. If K is the number of nodes in a server, then the overall time complexity of the proposed heuristic algorithm is $\mathcal{O}(N \times \log N + N \times K)$.

VI. Performance Evaluation

We compare the proposed mechanism with three alternative placement mechanisms: (1) *Node-balance:* it places SFCs by dividing them into all nodes to balance each node's core utilization; (2) *Node-first:* it tries to place SFCs on the local node first until all resources are consumed and then places the SFCs on the remote node; (3) *Octans [8]:* places SFCs with high traffic rates in the local node until all of its resources are depleted. Then it attempts to place in the remote node. All of these mechanisms assume that the placement of SFCs is static and one-time activity. We also allocate required system resources to SFCs in all these mechanisms to guarantee performance isolation. We evaluate the performance of *NUMASFP* by implementing it on a prototype system on OpenNetVM, a high-performance container-based platform for NFV, and the required changes are made to it. Since the testbed configuration used to perform real-time experiments is limited to 20 $Gbps$ traffic, we also conducted simulation experiments to demonstrate the efficacy of *NUMASFP* when running large number of SFCs. To migrate VNFs between cores on a server, we use the new docker feature *cpuset*.

A. Simulation Results

We consider 20 different SFCs of length 3 where VNFs are randomly picked from [20]. Five SFCs are randomly selected and placed in each of the four homogeneous servers, each of which has two NUMA nodes. Each SFC receives input traffic for 120 $secs$. To simulate a time-varying traffic demand, the traffic rate of each SFC is taken from the real traces available from [28] and generalized to 10 $Gbps$. Normalized Aggregate Throughput (NAT) is considered as the performance metric,

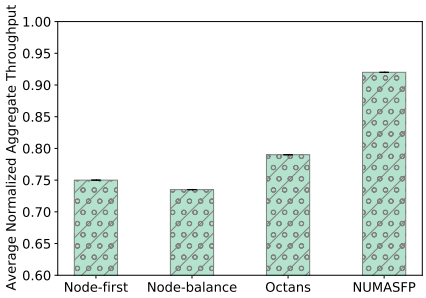


Figure 10: ANAT of four servers for different mechanisms.

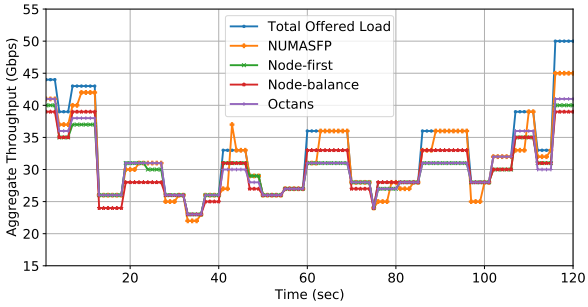


Figure 11: Aggregate throughput for various mechanisms over time for one of the four servers.

which is defined as the ratio of aggregate throughput achieved for all SFCs over the total offered load. NAT is calculated for each time interval. The average of NAT for all time instances is calculated and it is named as Average NAT (ANAT).

Fig. 10 shows ANAT for different mechanisms. *NUMASFP* outperforms the other mechanisms. As plotted, *NUMASFP* achieves 16%, 25%, and 23% more ANAT than *Octans*, *Node-balance*, and *Node-first* mechanisms, respectively. This is due to dynamically migrating SFCs among NUMA nodes. To get a clear understanding, we present in Fig. 11 the aggregate throughput of SFCs running in one of the servers using different placement mechanisms over time. *NUMASFP* outperforms other mechanisms for most of the time instances or it is comparable in some time instances, because it dynamically migrates SFCs across nodes based on the traffic rates of the SFCs. Other mechanisms presented do not account for the migration, and SFCs running in the remote node experience lower throughput at higher traffic rates, whereas SFCs running in the local node do not efficiently utilise the available resources. Based on these results, Fig. 12 represents achieved ANAT of the same server where *NUMASFP* performs better than other mechanisms.

From these results, we conclude that proposed *NUMASFP* mechanism is more effective in improving resource utilization while maximizing aggregate throughput of all SFCs. It also guarantees performance isolation among the co-located services deployed in NFV-based systems.

B. Testbed Results

Since the testbed configuration used to perform real-time experiments is limited to 20 *Gbps* traffic, we run two SFCs of length two, these two SFCs receive traffic ranging from 1 *Gbps* to 10 *Gbps* independently. SFC1 is run in the local node and

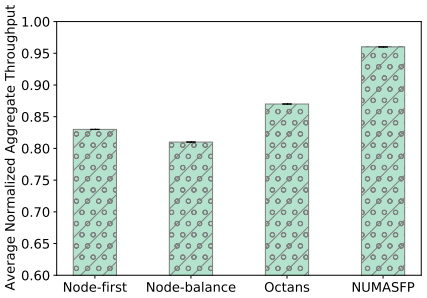


Figure 12: ANAT of the single server for different mechanisms.

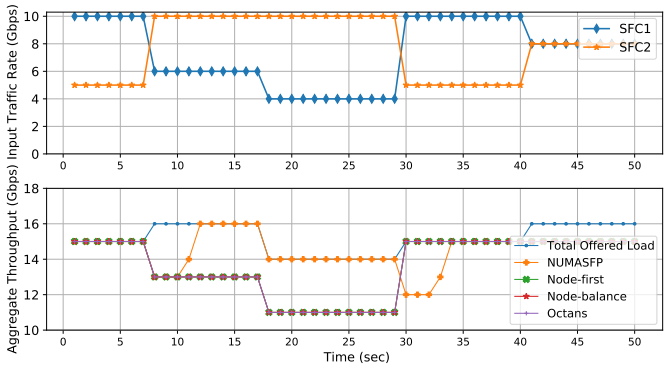


Figure 13: Experimental results of *NUMASFP*.

SFC2 in the remote node, with initial traffic rates of 10 *Gbps* and 5 *Gbps*, respectively. At time instance 8, the traffic rate of SFC1 is decreased to 6 *Gbps* and that of SFC2 is increased to 10 *Gbps*. As a result, the total throughput of all mechanisms drops. After the violation limit is exceeded at time instance 11, *NUMASFP* migrates SFC1 to the remote node and SFC2 to the local node. This results in an increase in throughput, as seen in Fig. 13.

Setting threshold for the number of violations before making a migration decision may cause ping-pong effects that affect overall system performance as each migration causes some amount of physical resources to be provisioned or released. This can be addressed by leveraging machine learning (ML) techniques that can help in predicting the future traffic which can help in minimizing the number of migrations.

VII. Conclusions and Future Work

In this paper we presented *NUMASFP*, an NFV orchestrator that places SFCs in a many-core server and dynamically migrates SFCs among NUMA nodes based on their traffic rates while maximizing the aggregate throughput of all SFCs. We built a prototype of *NUMASFP* on OpenNetVM, a high performance NFV platform for the performance study. The evaluations of *NUMASFP* reveal that it provides performance isolation while significantly improving aggregate system performance. As future work, we plan to work on integrating ML mechanisms for deciding when and where to migrate SFCs and how many resources to allocate to VNFs.

Acknowledgement

The authors would like thank Mr. Maruthi S Inukonda for extending his support in using the OpenNetVM platform for migration experiments.

REFERENCES

[1] Chaobing Zeng, Fangming Liu, Shutong Chen, Weixiang Jiang, and Miao Li. Demystifying the performance interference of co-located virtual network functions. In *Proc. of IEEE INFOCOM*, pages 765–773, 2018.

[2] Dejene Boru Oljira, Anna Brunstrom, Javid Taheri, and Karl-Johan Grinnemo. Analysis of network latency in virtualized environments. In *Proc. of IEEE GLOBECOM*, pages 1–6. IEEE, 2016.

[3] Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, and Justine Sherry. Contention-aware performance prediction for virtualized network functions. In *Proc. of ACM SIGCOMM*, pages 270–282, 2020.

[4] Joel Halpern, Carlos Pignataro, et al. Service function chaining (sfc) architecture. In *RFC 7665*. 2015.

[5] Minghao Chen, Yi Sun, Hailin Hu, Liangrui Tang, and Bing Fan. Energy-saving and resource-efficient algorithm for virtual network function placement with network scaling. *IEEE Transactions on Green Communications and Networking*, 5(1):29–40, 2021.

[6] Amir Varasteh, Marilet De Andrade, Carmen Mas Machuca, Lena Wosinska, and Wolfgang Kellerer. Power-aware virtual network function placement and routing using an abstraction technique. In *Proc. of IEEE GLOBECOM*, pages 1–7, 2018.

[7] Ke Yang, Hong Zhang, and Peilin Hong. Energy-aware service function placement for service function chaining in data centers. In *Proc. of IEEE GLOBECOM*, pages 1–6, 2016.

[8] Zhilong Zheng, Jun Bi, Heng Yu, Haiping Wang, Chen Sun, Hongxin Hu, and Jianping Wu. Octans: Optimal placement of service function chains in many-core systems. In *Proc. of IEEE INFOCOM*, pages 307–315, 2019.

[9] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. Understanding host network stack overheads. In *Proc. of ACM SIGCOMM*, pages 65–77, 2021.

[10] Christian Sieber, Raphael Durner, Maximilian Ehm, Wolfgang Kellerer, and Puneet Sharma. Towards optimal adaptation of nfv packet processing to modern cpu memory architectures. In *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*, pages 7–12, 2017.

[11] Youbing Zhong, Zhou Zhou, Xuan Liu, Da Li, Meijun Guo, Shuai Zhang, Qingyun Liu, and Li Guo. Bpa: The optimal placement of interdependent vnfs in many-core system. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 305–319. Springer, 2020.

[12] Mihai Dobrescu, Katerina Argyraki, and Sylvia Ratnasamy. Toward predictable performance in software packet-processing platforms. In *Proc. of ACM NSDI*, pages 141–154, 2012.

[13] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy, and Scott Shenker. Resq: Enabling slos in network function virtualization. In *Proc. of NSDI*, pages 283–297, 2018.

[14] Paul Veitch, Edel Curley, and Tomasz Kantecki. Performance evaluation of cache allocation technology for nfv noisy neighbor mitigation. In *Proc. of IEEE NetSoft*, pages 1–5, 2017.

[15] Norbert Schramm, Torsten M Runge, and Bernd E Wolfinger. The impact of cache partitioning on software-based packet processing. In *Proc. of IEEE NetSys*, pages 1–6, 2019.

[16] Bin Li, Yipeng Wang, Ren Wang, Charlie Tai, Ravi Iyer, Zhu Zhou, Andrew Herdrich, Tong Zhang, Ameer Haj-Ali, Ion Stoica, et al. Rldrm: closed loop dynamic cache allocation with deep reinforcement learning for network function virtualization. In *Proc. of IEEE NetSoft*, pages 335–343, 2020.

[17] Intel. Introduction to Cache Allocation Technology in the Intel Xeon Processor E5 v4 Family. Intel Corporation, 2016.

[18] Intel. Resource Director Technology in Linux. Intel Corporation, 2017.

[19] http://man7.org/linux/man-pages/man7/cpuset.7.html.

[20] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, KK Ramakrishnan, and Timothy Wood. Opennetvm: A platform for high performance network service chains. In *Proc. of ACM HotMiddlebox*, page 26–31, 2016.

[21] Yuxia Cheng and Wenzhi Chen. Evaluation of virtual machine performance on numa multicore systems. In *Proc. of IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 136–143, 2013.

[22] Andrew Herdrich, Edwin Verplanke, Priya Autee, Ramesh Illikkal, Chris Gianos, Ronak Singhal, and Ravi Iyer. Cache qos: From concept to reality in the intel® xeon® processor e5-2600 v3 product family. In *Proc. of IEEE Symposium HPCA*, pages 657–668, 2016.

[23] User space software for intel resource director technology. https://github.com/ 01org/intel-cmt-cat/tree/master/pqos.

[24] Inc. GitHub. Pktgen - traffic generator powered by dpdk. https://github.com/Pktgen/Pktgen-DPDK/, 2019.

[25] https://github.com/sdnfv/opennetvm/tree/master/examples.

[26] http://kernel.ubuntu.com/ cking/stress-ng.

[27] Intel. DDIO: Intel Data Direct I/O Technology Overview. Intel White Paper, 2012.

[28] Venkatarami Reddy Chintapalli, Koteswararao Kondepu, Andrea Sgambelluri, Bheemarjuna Reddy Tamma, Piero Castoldi, Luca Valcarenghi, et al. Orchestrating edge- and cloud-based predictive analytics services. In *Proc. of IEEE EuCNC*, pages 214–218, 2020.