JARS: A Joint Allocation of Radio and System Resources for Virtualized Radio Access Networks

Keval A Malde, Venkatarami Reddy Chintapalli, Bhavishya Sharma, Bheemarjuna Reddy Tamma, Antony Franklin A
Indian Institute of Technology Hyderabad, India
Email: {cs20mtech01003, cs17resch01007, cs20mtech12006, tbr, antony.franklin}@iith.ac.in

NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium | 978-1-6654-7716-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/NOMS56928.2023.10154407

***Abstract*—Mobile operators are widely adopting Network Functions Virtualization (NFV) to get the benefits of virtualization, including ease of deployment, flexibility, and cost savings. NFV allows multiple virtualized Radio Access Networks (vRANs) to run on commodity hardware enabling joint signal processing and efficient interference management. In addition, mobile operators can run general-purpose workloads alongside vRANs to utilize spare system resources in the NFV infrastructure. In such a consolidated scenario, it is necessary to ensure that the Key Performance Indicators (KPIs) of vRAN workloads are always met. But the workload consolidation could cause high variability and unpredictability in the performance of the deployed vRANs due to contentions for shared system resources like CPU cores, Last Level Cache (LLC), etc. In order to address this problem, we present *JARS* – a joint allocation of radio and system resources for the NFV infrastructure – that dynamically adjusts system resources such as CPU cores and LLC-ways, and radio resources such as Physical Resource Blocks (PRBs) to ensure KPIs for vRANs and improve overall resource utilization by workload consolidation. We profile srsLTE to determine the minimal CPU core and LLC resource requirements to satisfy KPIs during different traffic loads, which is used in the JARS. Experimental studies on a prototype system show that the proposed JARS outperforms a state-of-the-art scheme by 23%.**

I. Introduction

Network Functions Virtualization (NFV) and cloud computing have attracted tremendous interest from mobile operators. They allow on-demand allocation of resources to Virtual Network Functions (VNFs), achieving better flexibility, scalability, and cost savings. VNFs are usually deployed in Virtual Machines (VMs) or containers using commodity hardware as shown in Fig. 1. A forthcoming trend is extending NFV to the Radio Access Network (RAN) enabling virtual RAN (vRAN) a.k.a. Cloud RAN (C-RAN) to deploy baseband functions of 4G/5G RANs on NFV Infrastructure (NFVI) [1].

For efficient utilization of NFVI, a standard approach is RAN pooling which involves sharing underlying system resources among several cells for achieving the so-called statistical multiplexing gain [2]. The authors of [3] considered spatio-temporal traffic fluctuations at cell sites for efficient allocation of computing resources. It is observed that even during peak hours, more than 50% of CPU cores are not being fully utilized due to variations in the computing demands of uplink and downlink signal processing [4]. The operators can use these spare resources to run non-critical, general-purpose workloads as shown in Fig. 1, which is known as *workload consolidation*. General-purpose workloads can be Machine Learning (ML) workloads [5], cellular-specific workloads such as local content caching and delivery workloads [6] that help the operators reduce user latency and traffic on back-haul networks, or even core network related workloads [7].

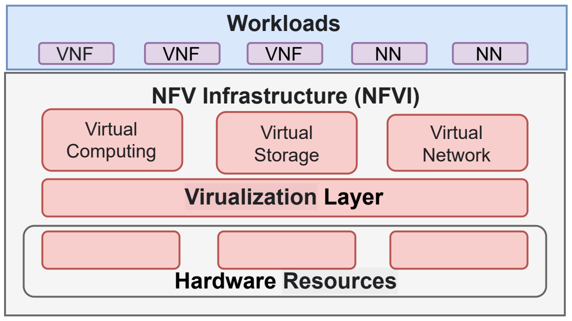


Fig. 1: An example of workload consolidation on NFVI: VNFs (vRANs) with general-purpose workloads (NNs).

Workload consolidation involves the mapping of VNFs (vRANs) and general-purpose workloads to the underlying compute servers present in the NFVI by the virtualization layer. The VNFs and other workloads that are deployed on any given server end up contending for the shared system resources (such as CPU, Last Level Cache (LLC), and memory bandwidth), which results in possible violation of Key Performance Indicators (KPIs) (e.g., throughput, latency) of vRANs deployed [8]. In [4], the authors proposed *concordia*, a mechanism for dynamically allocating the required number of CPU cores of the server to the vRAN and the remaining cores to general-purpose workloads to achieve effective CPU utilization. However, it overlooked the impact of LLC contentions on the performance of vRAN when it is co-located with other workloads on the same server.

Through experiments conducted over the srsLTE platform [9], we study how the performance of vRAN is impacted by varying amounts of system resources (such as CPU cores and LLC) allocated for different ingress traffic loads. It is observed that the combination of bare minimum CPU cores and LLC resources that need to be allocated to achieve the guaranteed KPIs is different for different ingress traffic loads of the cell. So, instead of over-provisioning for its peak resource demand, an efficient solution could be to dynamically adjust the amount of system resources allocated to a vRAN based on its current traffic demand. Towards this, in this paper, we present *JARS* – a joint resource allocation strategy – that dynamically allocates radio resources (i.e., PRBs) and system resources (i.e., CPU cores and LLC-ways) to a vRAN when it is co-located with other workloads on a compute server in the NFVI. The key contributions of this paper are as follows.

- Experimental study showing how vRAN performs with varying amounts of system resources like CPU cores and LLC-ways, and radio resources like PRBs.
- A novel resource allocator *JARS* that helps vRAN to maintain its KPIs even when it is co-located with other workloads on the same server for improving overall resource utilization of the NFVI.
- Experimental results on a prototype system show that the proposed *JARS* improves the performance of vRAN by 23% as compared to state-of-the-art approaches.

II. Related Work

Allocating compute resources in cellular networks has been the subject of extensive research. The authors of [10] created a model using a game-theory bargaining strategy that dynamically assigns computing resources to RAN. The authors of [11] employed traffic prediction to dynamically adjust computing resources for improving energy efficiency. The authors of [4] proposed *Concordia*, a user space deadline scheduling framework for vRAN that constructs a prediction model to forecast the worst-case execution time of vRAN signal processing tasks and reserves a minimal number of CPU cores for vRAN while freeing up the remaining cores for other workloads. If no care is taken while choosing general-purpose workloads for deployment along with vRAN, they could end up consuming more shared resources (e.g., LLC-ways) and cause performance degradation to the vRAN. Co-located workloads that negatively impact the performance of vRAN and other high-priority workloads deployed on a compute server are called as noisy-neighbors (NN).

There has been a significant amount of research that looked into the impact of NN in virtualized mobile environments. The authors of [12] developed an analysis methodology to evaluate how NNs affect the KPIs of 5G Core. To find NNs, they employed a variety of ML models. To find the underlying cause of NN using supervised learning, the authors of [13] developed an anomalous detection model. All of these works centered on detecting NNs in a general context and did not offer a solution to the NN problem.

A few works looked into the NN problem from an LLC point of view. For isolation and performance guarantees, the authors of [14] employed Intel Cache Allocation Technology (CAT) technology on the 5G core. The authors of [15], [16] looked at how different Class of Services (CLOS) performed in the presence of NN, including virtual firewalls and routers. They looked at how a NN affects throughput and latency for each COS and how to use CAT technology to overcome performance degradation and achieve considerable performance gains. The performance of a variety of VNFs in a NN environment was thoroughly studied in [17]. The authors of [18] created a framework which differentiates workloads into high priority and best-effort and then using deep reinforcement learning it allocates the required LLC-ways to the high priority workload and the remaining to the best-effort workload.

To the best of our knowledge, none of the existing works explored how LLC resource contentions due to the presence of NNs affect the performance of vRAN. It is crucial to develop new strategies for efficiently distributing CPU cores and limited system resources like LLC-ways across the co-located workloads while also taking into account their KPIs and the expense of the servers used.

III. Background and Motivation

This section provides background information on Intel's CAT tool, followed by motivational results demonstrating the impact of radio and system resources on the performance of vRAN using the srsLTE platform.

A. Cache Allocation Technology (CAT): An Overview

Intel's CAT tool provides software control over the number of LLC-ways that can be utilized by a CPU core. By means of a specialized resource tag called Class of Service (CLOS), CAT can assign partitions of the LLC to a particular core, thereby limiting the amount of LLC a core can use. The available cache ways/partitions can be dynamically assigned to each CLOS using Capacity Bit Mask (CBM), which indicates how much cache can be used by them. Bits within the CBMs indicate relative amount of cache space allocated and priority level of the CLOS. It is possible to create disjoint or overlapping cache ways using CLOS. But, the number of cache partitions is architecture-dependent.

Consider a commodity server with 20 CPU cores and 20 LLC-ways (each LLC-way is of 1 MB) for deploying vRAN and NN workloads. Table I shows a sample allocation of LLC-ways to these workloads in two different scenarios. In the first scenario, the vRAN is configured with 16 LLC-ways (80% of cache) over 18 cores, whereas the NN workload is configured with the remaining four LLC-ways (20% cache) across two cores. Using the same core assignment as the first scenario, the second scenario distributes cache ways (50%) evenly between the vRAN and the NN workloads.

TABLE I: CLOS using Capacity Bit Masks (CBMs)

Scenario	**Capacity Bitmask in Hex**	**Cache Capacity**	**CPU Assignment**
1	0x0ffff 0xf0000	16 MB 4 MB	2-19 (vRAN) 0,1 (NN)
2	0x003ff 0xffc00	10 MB 10 MB	2-19 (vRAN) 0,1 (NN)

B. Motivation

To clearly understand the effect of system resources and radio resources provisioned on the performance of vRAN, experiments are conducted by using an open-source LTE platform, srsLTE [9]. Fig. 2 shows the experimental setup using three commodity servers. One server runs the core network i.e., Evolved Packet Core (EPC) (srsEPC), while another server runs the base station (srsENB) which acts as

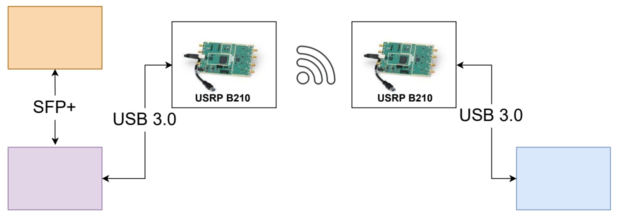


Fig. 2: Experimental setup using srsLTE and USRP SDRs.

the vRAN, and the third server acts as virtual User Equipment (UE) (srsUE). The UE and eNodeB communicate using USRP B210 SDRs (software-defined radios), which are connected via USB 3.0 to the respective servers. All the experiments are done on band 7 with 1x1 SISO antenna configuration in a controlled indoor environment. The system configurations and software instances used for conducting experiments are presented in Table II. To ensure the study conducted on vRAN is not affected by any other system-related factors, the features are set as shown in Table III.

TABLE II: System configurations

Component	Description
Processor	Intel Xeon CPU E5-2630 @ 2.30 GHz 20 Core Dual NUMA socket
OS and Kernel	Ubuntu 18.04.1; 5.4.0-122-generic
Cache	32 KB L1; 256 KB L2; 25.6 MB LLC
LLC-ways	20
Main Memory	62 GB

TABLE III: System parameters

Feature	Status
Hardware prefetching, L1 data prefetching, adjacent cache line prefetching, and IP prefetching	Disabled
C-States	Off
Turbo mode	Disabled
CPU governor	Performance

Tools Used: We used the following tools to profile vRAN, to make system-level changes, and to create a NN environment.

- *isolcpus* [19]: Removes the user specified CPUs from the kernel's Symmetrical Multiprocessing (SMP) balancing and scheduler algorithms.
- *perfstat* [20]: Provides system-level information such as cache hit/miss, memory bandwidth consumption, and Instructions Per Cycle (IPC).
- *pqos* [21]: Allows the user to configure LLC-ways by creating CLOS using Intel's CAT tool.
- *iperf3* [22]: To generate user traffic between UE and EPC.
- *taskset* [23]: To set or retrieve the CPU affinity of a workload.
- *Redis [24]*: An open-source, in-memory data store used to simulate a NN workload.

1) Impact of CPU core allocation: This study investigates the effect of CPU core allocations on the performance of vRAN. We identify the minimum number of CPU cores required by the vRAN to achieve maximum throughput for different PRB configurations. In general, all workloads running on a server are scheduled by the kernel, which considers all cores equivalent and can schedule a workload on any core or combination of cores based on the workload's requirements. The core(s) assigned to a specific workload can change during its execution for numerous reasons, including load balancing. If the number of workloads exceeds the number of available cores, then some of the workloads wait in a queue until the cores become available. This typically results in a delay in execution time, which in the case of vRAN can result in a poor connection for UEs or loss of connectivity in the worst-case scenario. To ensure that the vRAN gets sufficient CPU cores to meet the tighter deadline, we leverage processor affinity or CPU pinning to bind/pin it to a specific core(s). The kernel then allocates the requested cores exclusively to the vRAN, thereby preventing other tasks/workloads from being scheduled on them.

We have conducted a series of experiments to identify the minimum number of CPU cores required by the vRAN to achieve the same throughput as when it operates under ideal conditions for varying traffic loads. The ideal condition is when all CPU cores are allocated to the vRAN and no other workloads (NNs) are deployed on the same server. In the first set of experiments, the vRAN is running under ideal conditions. For each experiment in the set, we changed only one configuration parameter i.e, PRBs and observed the maximum achieved throughput for that particular PRB configuration. We generated downlink TCP traffic from EPC to UE using *iperf3* tool for 120 *secs* and measured the average throughput achieved.

The second set of experiments aims to find the bare minimum number of CPU cores required by the vRAN to achieve the same throughput as that under ideal conditions for various PRB configurations. To determine the minimum number of cores required for vRAN, we began by allocating a single core using *taskset* tool and then increased the number of cores until the throughput matched that of the ideal conditions.

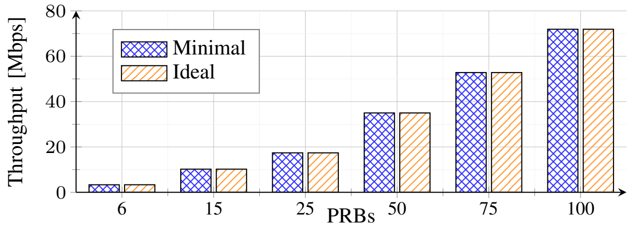


Fig. 3: Throughput trends for vRAN in case of CPU ideal and minimal.

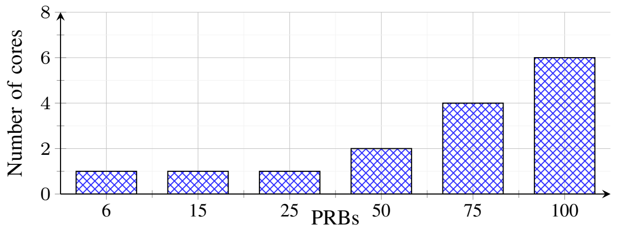


Fig. 4: Minimum number of cores required for vRAN to achieve maximum throughput for different PRBs.

Fig. 3 shows the achieved throughput values under ideal CPU allocation (first set of experiments) and minimal CPU allocation (2nd set of experiments). It is clear that the corresponding throughput values are same in both sets of experiments for all different PRB configurations. This indicates that we do not have to allocate all of the CPU cores of the server to the vRAN to guarantee its KPIs.

Fig. 4 shows the minimum number of CPU cores required to achieve the same performance as the ideal scenario of different PRB configurations. It shows that the required number of CPU cores increases with the number of PRBs, however, up to 25 PRBs, a single core is sufficient to achieve the maximum throughput. By allocating the required number of CPU cores to vRAN without affecting its performance, other workloads can be deployed on the same server to utilize the remaining CPU cores. CPU core pinning ensures that no other workload is scheduled onto the CPU cores allocated to vRAN, thereby creating isolation among the co-located workloads.

Observation 1: The number of CPU cores assigned to vRAN significantly impacts its performance. Furthermore, the number of cores required is a function of PRBs. By profiling the vRAN, we can find the minimal number of cores required for vRAN and allocate the remaining cores to other co-located workloads for effective utilization of the system resources.

2) Impact of Noisy Neighbors (NNs): This study aims to determine the impact of NNs on vRAN when they are deployed on the same server. NN workloads demand a significant amount of shared resources and cause resource contentions, resulting in performance degradation to high priority, co-located workloads like vRAN. Here, we focus on an often overlooked shared resource i.e., LLC. Along with vRAN, we deploy *redis* server as the NNs, which is cache intensive in nature. We set the number of PRBs to 100, allocate six CPU cores to vRAN (as shown in Fig. 4) and the remaining to the NNs (by running multiple instances of *redis*). We again generated a downlink TCP traffic for 120 secs to vRAN and measured its achieved throughput.

Fig. 5 shows the achieved throughput of vRAN when it runs in isolation (as shown earlier in Fig. 3) vis-a-vis when it is co-located with the NNs. Even with a sufficient number of cores assigned to the vRAN, it is clear from the plot that its performance degrades by around 25% when it is co-located with the NN. In order to determine the cause of this degradation, we also measured the number of LLC misses for the vRAN using *perfstat* tool and plotted them on Y2-axis. The number of LLC misses experienced by the vRAN is significantly higher when it is co-located with the NN. It indicates that the majority of LLC-ways are occupied by the NN, which substantially degraded the performance of the vRAN. This study indicates that vRAN's performance is also sensitive to LLC-ways.

Observation 2: Core isolation alone is not sufficient to guarantee KPIs of vRAN when it is co-located with NNs. Contentions at LLC, due to NNs, can cause significant performance degradation for vRAN.

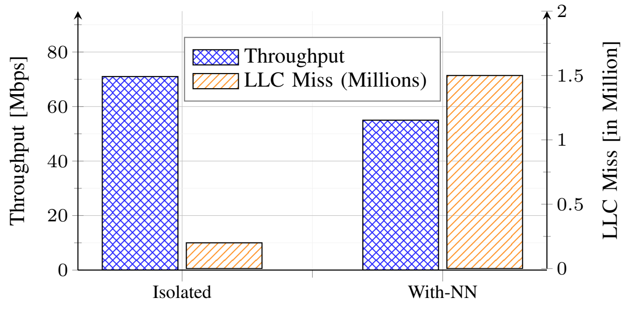


Fig. 5: Impact of NNs on vRAN throughput and corresponding LLC miss rate.

3) Impact of LLC allocation: This study aims to determine the effect of LLC allocations on vRAN performance. We continue to use the same experimental setup as that used in the previous subsection to study the impact of NNs. The LLC allocations for vRAN are controlled by using *CAT* tool. We start with allocating one LLC-way to vRAN and repeat the experiments by adding more LLC-ways incrementally until the desired maximum throughput (as shown earlier in Fig. 3) was achieved.

Fig. 6 shows how the number of LLC-ways allocated for 100 PRBs affects the throughput of vRAN. It is clear that the throughput has increased with the number of LLC-ways. The throughput achieved when all (20) of the LLC-ways are assigned to vRAN is identical to the throughput obtained when nine LLC-ways are allocated to it. When we allocated fewer than nine LLC-ways, we noticed a drop in throughput, and we could not able to run vRAN with fewer than six LLC-ways.

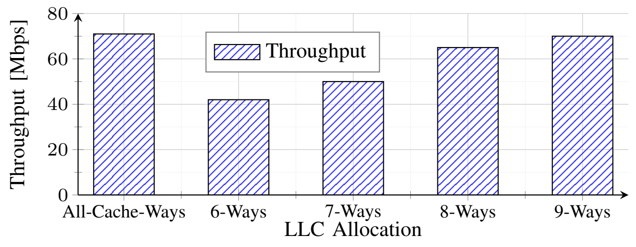


Fig. 6: Impact of LLC allocation on vRAN throughput for 100 PRBs.

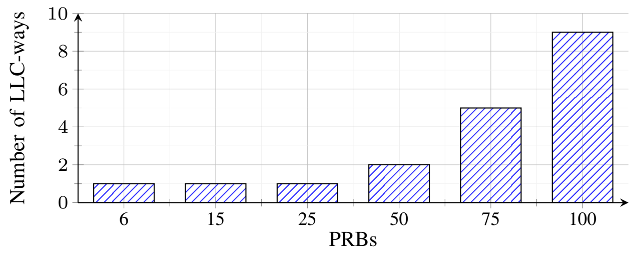


Fig. 7: Minimum number of LLC-ways required for vRAN to achieve maximum throughput for different PRBs.

We repeat this experiment for different PRB configurations to determine the minimum required LLC-ways (refer Fig. 7) to achieve the same throughput as when all of the LLC-ways are allocated to vRAN without any NNs.

Observation 3: Adequate allocation of LLC-ways and CPU cores along with core pinning is critical in achieving the expected performance of vRAN in the presence of NNs. Furthermore, the bare minimum LLC-ways is a function of PRBs.

To the best of our knowledge, no existing work considered the joint allocation of radio resources i.e., PRBs and system resources (CPU cores and LLC-ways) to vRAN when it is co-located with NNs on commodity servers. In the next section, we propose JARS, an efficient resource allocation scheme for dynamically adjusting the amount of radio and system resources allocated to vRAN while guaranteeing its KPIs.

IV. Proposed Work

This section first presents the system architecture of *JARS*, then the mechanism for profiling vRAN, and finally the proposed *JARS* scheme for dynamic resource allocation.

A. System Architecture

To capture the traffic characteristics and KPI requirements of vRAN, we propose *JARS* with four core components: Monitor Module, vRAN Profiler, Resource Table, and Control Module (refer Fig. 8). The **Monitor Module** is in charge of measuring the ingress traffic load of vRANs at one-second interval. The **vRAN Profiler** builds the lookup table by characterizing each vRAN in offline mode. The lookup table lists the bare minimum radio and system resources that need to be provisioned at the vRAN so that it achieves the maximum performance for each of the ingress traffic loads. The **Resource Table** keeps track of the resources (in terms of cores, LLC-ways, and PRBs) that are allocated to the vRAN and other co-located workloads deployed on the server. The **Control Module** gets ingress traffic load information of various vRANs deployed on the commodity server from the monitor module and determines if the allocated resources are sufficient or need to be adjusted by using a resource allocation scheme.

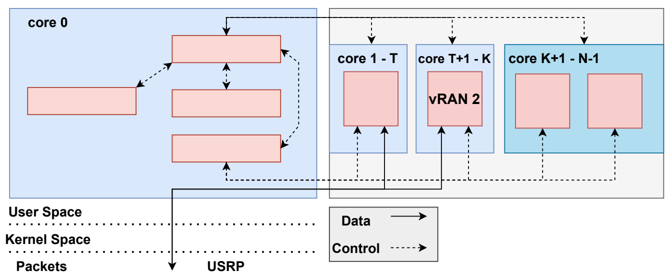


Fig. 8: System architecture of *JARS*.

B. vRAN Profiling

In this section, we briefly describe the procedure followed for profiling vRAN. The main goal of profiling a vRAN is to understand its performance characteristics, and build a lookup table, thereby helping the *JARS* in allocating system resources in an effective manner. Generally, profiling takes place only once for each vRAN and can be done offline. For profiling, we consider a set S of PRB configurations that a mobile operator supports in 4G/5G. The maximum theoretical traffic load that various PRB configurations can support is known from 3GPP specs [25]. Using this information, we generate downlink TCP traffic using *iperf3* tool by setting the traffic rate according to the PRB's theoretical maximum.

First, we profile vRAN to determine the minimum number of cores required to handle the maximum theoretical traffic load. To achieve this, we configure vRAN with a PRB value from the set S and allocate all the cores in the server *i.e.,* no NNs; we then send the corresponding maximum theoretical traffic to the vRAN and monitor the achieved throughput. If the input traffic load and achieved throughput are comparable, we gradually reduce the number of cores given to the vRAN until we observe a noticeable difference between them. The latest core allocation that did not result in any throughput degradation is recorded as the best allocation. This procedure is repeated for other PRB values in the set S, and the lookup table is populated with the minimum core values for each of the PRBs. Next, we profile vRAN for the least number of LLC-ways required to handle the theoretical maximum traffic load. This procedure is similar to that of profiling done to determine the minimum number of cores, with the difference that the cores allocated to the vRAN are fixed based on the results of core profiling. The lookup table is populated with the minimum required LLC-ways and the minimum required cores for each of the PRB values using this profiling procedure. In addition, we also keep the minimum traffic load that each PRB configuration can support in the lookup table. The minimum traffic load for any PRB configuration is obtained by adding one to the maximum traffic load supported by the preceding PRB configuration in the set S. Table IV shows a few entries of the lookup table constructed by vRAN profiler. *JARS* picks the most appropriate entry from the lookup table based on the ingress traffic load of the vRAN and adjusts radio (PRBs) and system resources (cores and LLC-ways) allocated to the vRAN for each time interval.

TABLE IV: A sample lookup table from 4G vRAN profiling

Min. traffic load [Mbps]	Max. traffic load [Mbps]	PRB	Cores	LLC
35	52	75	4	5
53	71	100	6	9

C. Resource Allocation Scheme

As the input traffic load to vRAN changes over time, the required resources need to be reconfigured rapidly to meet its KPIs. The objective of the resource allocation scheme is to perform efficient allocation of various resources to the vRAN and other workloads running on the server for improving the overall resource utilization. The proposed resource allocation scheme is given in Algorithm 1, which is implemented in the control module of JARS architecture. There could be multiple vRANs (represented using an array $vRAN$) running on the same server, along with other workloads (represented using an array BE), and it is assumed that the server has enough resources to meet the peak traffic demands of all the vRANs deployed (N denotes the number of vRAN deployed on the server). First, JARS identifies the workloads running on the server and classifies them into two groups: vRAN and Best Effort (BE). JARS considers vRAN workload as high priority ones and other workloads as low priority, BE ones. We create a separate CLOS for each vRAN (represented as ρ[i]) to allocate dedicated system resources and a single CLOS (represented as ρ[N+1]) for all of the BE workloads together (lines 3-9). The proposed algorithm has two major phases: initial allocation and dynamic allocation.

Algorithm 1: *Proposed Resource Allocation Scheme*

```
1  Input: Ingress traffic load of vRANs: $\lambda_i$, $\forall i \in vRAN$; The
     lookup table from vRAN profiling
2  Output: Radio and system resource allocations to the
     vRANs and BE workloads
3  begin
       // Initialization for vRAN workloads
4      for $i$ = 1 to N do
5          $\rho$[i] $\leftarrow vRAN_i$
6      end
       // Initialization for BE workloads
7      for k = 1 to M do
8          $\rho$[N+1] $\leftarrow BE_k$
9      end
10     initial_allocation(); // at t=0
       // Dynamic allocation at each of other intervals
11     for $t = 1, 2, \ldots,$ do
12         re_allocate_resources();
13     end
14 end
```

Algorithm 2: initial_allocation()

```
1  begin
2      for $i$ = 1 to N do
3          Based on the ingress traffic load of $vRAN_i$, refer to
             the lookup table to allocate radio and system
             resources and update $\rho_i$
4      end
5      Distribute spare system resources to the $BE$ workloads
         and update $\rho_{N+1}$
6  end
```

1) *Initial Allocation:* Following the creation of CLOSs for workloads, JARS first determines the required resources for each vRAN based on the traffic loads (λ_i) for that vRAN by using the lookup table and allocates them to the respective vRANs. The remaining resources are then distributed to the BE workloads running on the same server. This is done by $initial\_allocation()$ procedure (Algorithm 2). After allocating resources to each workload (vRAN/BE), the resource table is updated to keep track of available resources.

2) *Dynamic Allocation:* Since the input traffic load of vRAN changes over time, the required resources must be reconfigured rapidly to meet the KPIs of the vRANs. For each time interval t, resources are dynamically adjusted to each vRAN based on its ingress traffic load at $t-1$ by calling $re\_allocate\_resources()$ procedure (Algorithm 3).

Algorithm 3: re_allocate_resources()

```
1  begin
2      $\lambda_i \leftarrow$ the current input traffic load of $vRAN_i$
       // Compute the current resource requirements using the
         lookup table
3      for $i$ = 1 to N do
4          Measure the difference between the currently
             allocated resources of $vRAN_i$ and the resource
             requirements based on the current traffic demand
5      end
       // Resource release
6      for $i$ = 1 to N do
7          if extra resources available at $vRAN_i$ then
8              Release the surplus resources and add them to
                 $resource\_pool$ and update $\rho_i$
9          end
10     end
       // Resource acquire
11     for $i$ = 1 to N do
12         if allocated resources at $vRAN_i$ are not sufficient
             then
13             if $resource\_pool$ has sufficient resources then
14                 Allocate additional required resources from
                     $resource\_pool$
15                 Update $\rho_i$ and $resource\_pool$
16             else
17                 Measure the additional resources needed
                     compared to those already available from
                     $resource\_pool$ and obtain from $BE$
                     workloads
18                 Update $\rho_i$
19             end
20         end
21     end
       // Resource allocation to BE workloads
22     if $resource\_pool$ has spare resources then
23         Distribute the spare resources from $resource\_pool$
             to the $BE$ workloads and update $\rho_{N+1}$
24     end
25 end
```

$resource\_pool$ is a two-dimensional array which maintains the available resources (i.e., cores and LLC-ways) in the system. The $re\_allocate\_resource()$ procedure has four major steps:

Step 1: Determines the required resources for the co-located vRANs based on their respective current traffic demands using the lookup table (lines 2-5). A vRAN may either require additional resources than its current allocation or release some of its allocated resources.

Step 2: vRANs which need fewer cores and/or LLC-ways than their current allocations release extra resources to the $resource\_pool$. Then, the CLOS of vRAN is updated accordingly (lines 6-10).

Step 3: Once all of the surplus resources from the co-located vRANs have been released, vRANs that require more cores and/or LLC-ways than their current allocations can claim additional resources from the $resource\_pool$. If adequate resources are available to satisfy vRAN's requirements, the appropriate resources are allocated. If resources are insufficient after being allocated from $resource\_pool$, some resources are claimed from the co-located BE workloads. Then, vRAN's

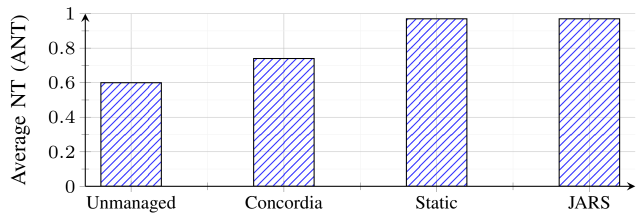


Fig. 9: Variation in ANT for different schemes.

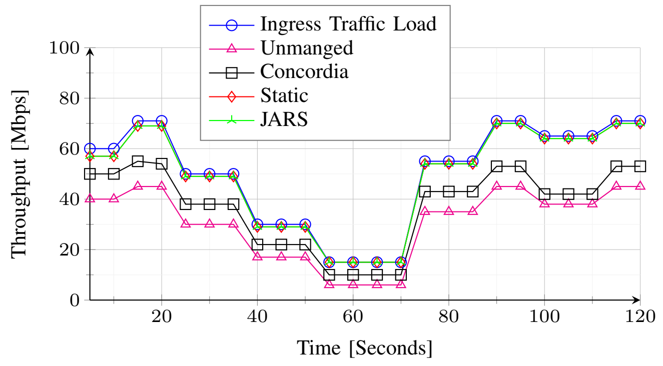


Fig. 10: Variation in the observed throughput of vRAN vs time for different input traffic loads.

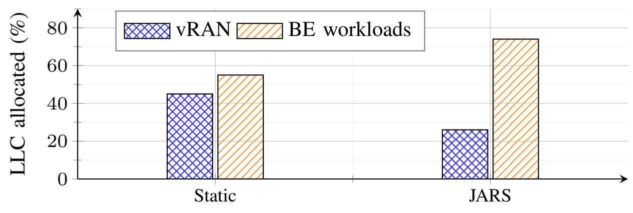


Fig. 11: Average percentage of LLC for *static* and *JARS* schemes.

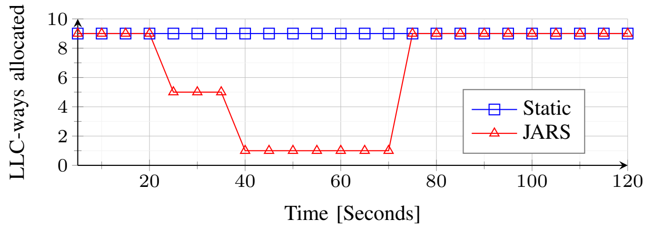


Fig. 12: LLC allocations over time in case of *static* and *JARS* schemes.

CLOS is updated accordingly (lines 11-21).

Step 4: Consequently, if any free resources are available after distributing to the co-located vRANs, they are allocated to the co-located BE workloads deployed on the same server (lines 22-24).

V. Performance Evaluation

Proposed *JARS* is compared with the following state-of-the-art and baseline schemes:

1) *Unmanaged (UM):* In this scheme, the resource allocation procedure operates in a conventional manner *i.e.,* there is no control over sharing of system resources. So, *UM* is a contention-unaware scheme where all of the co-located workloads contend for system resources like CPU cores and LLC-ways without any isolation.
2) *Concordia* [4]: Depending upon the incoming traffic load to vRAN, this scheme allocates only the required number of CPU cores to it exclusively without providing any LLC-ways exclusively. Like *JARS*, this also runs at one-second interval and adjusts the number of CPU cores allocated to it.
3) *Static:* In this scheme, system resources are liberally assigned to vRAN based on the peak traffic load *i.e.,* the maximum resources are allocated from the lookup table irrespective of the current ingress traffic load to the vRAN.

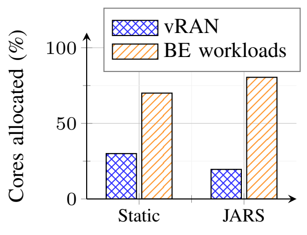
In the following, we present experimental and simulation studies conducted to evaluate the performance of *JARS*.

A. Experimental Studies

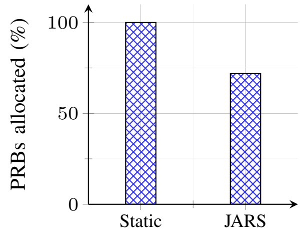
We run a single instance of vRAN (srsENB from srsLTE platform) along with the BE workloads (multiple instances of *redis*) on a commodity server with 20 CPU cores and 20 LLC-ways. The experimental setup described in Section III-B is used for conducting real-time experiments as well; downlink TCP traffic is generated from EPC to UE for 120 *secs* using *iperf3* tool and observed the achieved throughput for four different resource allocation schemes. The ingress traffic load to vRAN is taken from the real traces available from [26]. Normalized Throughput (NT) is considered the performance metric, which is defined as the ratio of achieved throughput by the vRAN over the input traffic load to the vRAN. NT is calculated for each time interval (with a granularity of one second) and the average of NT over all the time instances is reported as Average NT (ANT) in the plots.

Fig. 9 shows the observed ANT for four different resource allocation schemes. *JARS* achieves 38% and 23% more ANT than *unmanaged* and *concordia* schemes. This is due to the fact that other schemes are not controlling either LLC or both CPU and LLC, for which all of the workloads deployed on the server compete, and therefore vRAN is not getting sufficient resource(s). Whereas *static* scheme achieves the same throughput as *JARS* due to the over-provision of resources. As a result of over-provisioning, the underlying resources remain underutilized and negatively impact other co-located workloads. Fig. 10 shows the observed throughput of vRAN over time for the considered traffic load for different schemes.

In addition to evaluating ANT, we measured the average percentages of resources allocated to vRAN and other workloads using *static* and *JARS* schemes for the traffic load shown in Fig. 10. Fig. 11 shows the average percentages of LLC allocated to vRAN and other workloads. *JARS* saves approximately 42% more LLC than *static* allocation while maintaining the same performance. This means that other workloads had access to only 55% of LLC in *static* allocation, but in *JARS* allocation, they had access to 72% of LLC. Fig. 12 shows the allocation of LLC to vRAN over time for these two schemes. Fig. 13a shows the average of cores



(a) Core allocation



(b) PRB allocation

Fig. 13: Percentage of system resource allocation for different workloads in case of *static* and *JARS* schemes.

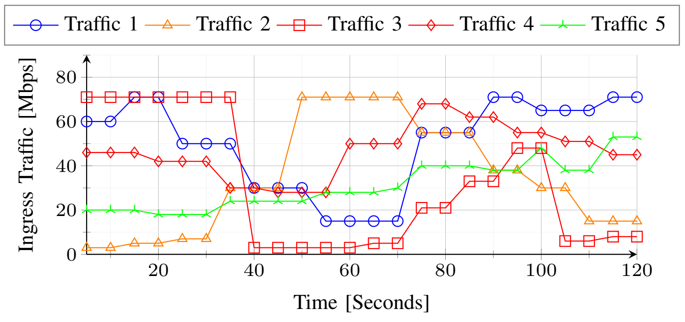


Fig. 14: Ingress Traffic for different vRANs.

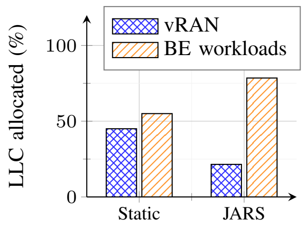
allocated in percentages. Using *static*, only 70% of cores were allocated to other workloads, whereas *JARS* scheme allocated 80% of cores. Fig. 13b shows the average allocation of PRBs in percentages, where *JARS* can save 28% of PRBs which could help in power savings. For all resource allocations made using *JARS*, more resources are allocated to other workloads on average without affecting the performance of vRAN. This is due to the fact that vRAN is configured with the bare minimal, but sufficient resources based on its ingress traffic load.

B. Simulation Studies

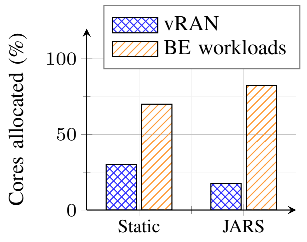
To demonstrate the efficacy of *JARS* in a bigger setting, we conducted simulation experiments by considering five homogeneous servers (20 cores and 20 LLC-ways) running one vRAN instance each along with BE workloads. Each vRAN is subjected to a different ingress traffic load for 120 *secs* as shown in Fig. 14. The average percentages of LLC allocated by *static* and *JARS* schemes are shown in Fig. 15a. *JARS* outperforms *static*, saving 54% of LLC compared to static. In *static*, BE workloads get 55% of LLC, whereas, in *JARS*, they receive 78% of LLC. Fig. 15b shows the average percentages of allocated cores, where *JARS* outperforms *static* by saving 41% of cores. With *static*, BE workloads receive 70% of the cores on average, whereas in *JARS*, they receive 78% of LLC. Fig. 15c shows the average percentages of PRBs allocated, where *JARS* outperforms *static* by reducing 36% of PRBs. Thus, by effectively assigning resources for vRAN, BE workloads receive extra resources.

C. Limitations of JARS

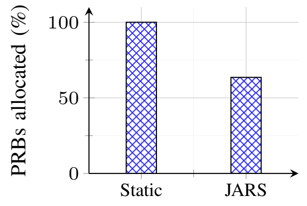
Predicting Ingress Traffic Load: In *JARS*, we considered certain traffic load distribution to vRAN while adjusting



(a) LLC allocation



(b) Core allocation



(c) PRB allocation

Fig. 15: Average percentage of system and radio resource allocations to workloads using *static* and *JARS* schemes.

radio and system resources at different time intervals. But, it is not possible to accurately know the future traffic load in real-world scenarios. Hence, the mobile operators have to estimate the input traffic load for the next time interval using historical traffic patterns received from the monitoring module by employing forecasting mechanisms [27].

vRAN Profiling: The proposed *JARS* scheme relies on profiling to characterize vRAN. We argue that the task of profiling is not overhead as it takes place only once for each vRAN and can be done offline. Moreover, we target network services like vRAN that require stringent performance guarantees.

VI. Conclusions and Future Directions

In this work, we studied the impact of resource contentions on the vRAN KPIs and demonstrated why isolation both at the core level and LLC are essential. Based on the observations, we proposed *JARS* which jointly allocates sufficient number of radio resources (PRBs) and system resources (CPU cores and LLC-ways) to vRAN when it is co-located with other workloads on commodity servers. Such an allocation not only guaranteed the KPIs of the vRANs but also improved the overall resource utilization of the server by offering spare resources to the co-located workloads. Experimental results on a prototype system showed that the proposed *JARS* scheme improved the performance by 38% over a baseline mechanism and 23% over *Concordia*, a state-of-the-art scheme. Future directions include extending the proposed solution to integrating ML mechanisms to predict traffic loads to vRAN. In addition, we would like to study the effect of energy consumption of vRAN in Noisy Neighbor scenarios.

Acknowledgment

This work has been supported by the project “CCRAN: Energy Efficiency in Converged Cloud Radio Next Generation Access Network” funded by Intel India.

References

[1] Andres Garcia-Saavedra, Xavier Costa-Perez, Douglas J. Leith, and George Iosifidis. Fluidran: Optimized vran/mec orchestration. In *Proc. of IEEE INFOCOM*, 2018.

[2] Jingchu Liu, Sheng Zhou, Jie Gong, Zhisheng Niu, and Shugong Xu. Statistical multiplexing gain analysis of heterogeneous virtual base station pools in cloud radio access networks. *IEEE Transactions on Wireless Communications*, 15(8):5681–5694, 2016.

[3] Debashisha Mishra, Himank Gupta, Bheemarjuna Reddy Tamma, and A. Antony Franklin. Kora: A framework for dynamic consolidation and relocation of control units in virtualized 5g ran. In *Proc. of IEEE ICC*, 2018.

[4] Xenofon Foukas and Bozidar Radunovic. Concordia: Teaching the 5g vran to share compute. In *Proc. of ACM SIGCOMM*, 2021.

[5] Tarik Taleb, Abdelquoddouss Laghrissi, and Djamel Eddine Bensalem. Toward ml/ai-based prediction of mobile service usage in next-generation networks. *IEEE Network*, 34(4):106–111, 2020.

[6] Jaime Llorca, Antonia M. Tulino, Kyle Guan, Jairo Esteban, Matteo Varvello, Nakjung Choi, and Daniel C. Kilper. Dynamic in-network caching for energy efficient content delivery. In *Proc. of IEEE INFOCOM*, 2013.

[7] Tuyen X. Tran, Abolfazl Hajisami, Parul Pandey, and Dario Pompili. Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges. *IEEE Communications Magazine*, 55(4):54–61, 2017.

[8] Xiaocheng Liu, Chen Wang, Bing Bing Zhou, Junliang Chen, Ting Yang, and Albert Y. Zomaya. Priority-based consolidation of parallel workloads in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 24(9):1874–1883, 2013.

[9] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D. Sutton, Pablo Serrano, Cristina Cano, and Doug J. Leith. Srslte: An open-source platform for lte evolution and experimentation. In *Proc. of ACM WiNTECH*, 2016.

[10] Mojgan Barahman, Luis M. Correia, and Lúcio Studer Ferreira. A qos-demand-aware computing resource management scheme in cloud-ran. *IEEE Open Journal of the Communications Society*, 1:1850–1863, 2020.

[11] Yongqin Fu and Xianbin Wang. Traffic prediction-enabled energy-efficient dynamic computing resource allocation in cran based on deep learning. *IEEE Open Journal of the Communications Society*, 3:159–175, 2022.

[12] Francisco Muro, Eduardo Baena, Sergio Fortes, Lars Nielsen, and Raquel Barco. Noisy neighbour impact assessment and prevention in virtualized mobile networks. *IEEE Transactions on Network and Service Management*, 2022.

[13] Hedi Bouattour, Yosra Ben Slimen, Marouane Mechteri, and Hanane Biallach. Root cause analysis of noisy neighbors in a virtualized infrastructure. In *Proc. of IEEE WCNC*, 2020.

[14] Paul Veitch, Chris Macnamara, and John J Browne. Balancing nfv performance and energy efficiency. In *Proc. of IEEE ICIN*, 2022.

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

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

[17] Venkatarami Reddy Chintapalli, Madhura Adeppady, Bheemarjuna Reddy Tamma, and Antony Franklin A. Restrain: A dynamic and cost-efficient resource management scheme for addressing performance interference in nfv-based systems. *Journal of Network and Computer Applications*, 201, 2022.

[18] Bin Li, Yipeng Wang, Ren Wang, Charlie Tai, Ravi Iyer, Zhu Zhou, Andrew Herdrich, Tong Zhang, Ameer Haj-Ali, Ion Stoica, and Krste Asanovic. Rldrm: Closed loop dynamic cache allocation with deep reinforcement learning for network function virtualization. In *Proc. of IEEE NetSoft*, 2020.

[19] Isolcpus. https://rb.gy/zdbmfs.

[20] perfstat. https://linux.die.net/man/1/perf-stat.

[21] pqos. https://manpages.ubuntu.com/manpages/xenial/man8/pqos.8.html.

[22] iperf3. https://iperf.fr/iperf-download.php.

[23] Taskset. https://man7.org/linux/man-pages/man1/taskset.1.html.

[24] redis. https://redis.io/topics/benchmarks.

[25] 3GPP. Ts 36.213, evolved universal terrestrial radio access (e-utra); physical layer procedures.

[26] Mehul Sharma, Ujjwal Pawar, A Antony Franklin, and Bheemarjuna Reddy Tamma. Proactive clustering of base stations in 5gc-ran using cellular traffic prediction. In *Proc. of IEEE NetSoft*, 2022.

[27] Jorge Martín-Pérez, Koteswararao Kondepu, Danny De Vleeschauwer, Venkatarami Reddy, Carlos Guimarães, Andrea Sgambelluri, Luca Valcarenghi, Chrysa Papagianni, and Carlos J. Bernardos. Dimensioning v2n services in 5g networks through forecast-based scaling. *IEEE Access*, 10:9587–9602, 2022.