RLDRM: Closed Loop Dynamic Cache Allocation with Deep Reinforcement Learning for Network Function Virtualization

Bin Li†, Yipeng Wang†, Ren Wang†, Charlie Tai†, Ravi Iyer†, Zhu Zhou†, Andrew Herdrich†, Tong Zhang†,
Ameer Haj-Ali‡*, Ion Stoica‡, Krste Asanovic‡
†Intel Corporation
{bin.li, yipeng1.wang, ren.wang, charlie.tai, ravishankar.iyer, zhu.zhou, andrew.j.herdrich, tong2.zhang}@intel.com
‡ University of California, Berkeley
{ameerh, istoica, krste}@berkeley.edu

***Abstract*—Network function virtualization (NFV) technology attracts tremendous interests from telecommunication industry and data center operators, as it allows service providers to assign resource for Virtual Network Functions (VNFs) on demand, achieving better flexibility, programmability, and scalability. To improve server utilization, one popular practice is to deploy best effort (BE) workloads along with high priority (HP) VNFs when high priority VNF's resource usage is detected to be low. The key challenge of this deployment scheme is to dynamically balance the Service level objective (SLO) and the total cost of ownership (TCO) to optimize the data center efficiency under inherently fluctuating workloads. With the recent advancement in deep reinforcement learning, we conjecture that it has the potential to solve this challenge by adaptively adjusting resource allocation to reach the improved performance and higher server utilization. In this paper, we present a closed-loop automation system RLDRM[1] to dynamically adjust Last Level Cache allocation between HP VNFs and BE workloads using deep reinforcement learning. The results demonstrate improved server utilization while maintaining required SLO for the HP VNFs.**

I. Introduction

Network function virtualization (NFV) becomes more and more popular among telecommunication industry and cloud service providers due to its scalability, flexibility, and cost efficiency. NFV enables people to develop and deploy networking services on general purpose servers quickly, without relying on proprietary networking hardware [1]–[4].

With the new NFV paradigm, network service providers build virtual network functions (VNFs) such as virtual switches, gateways, and firewalls, and consolidate those VNFs to run on commodity servers. To maintain the service level objectives (SLOs), the servers running VNFs are usually over-provisioned, which leads to wasted power and hardware resources and increased cost of ownership [5], [6]. In order to further improve server utilization and reduce cost, service providers often launch certain best effort (BE) workloads on the same server with the high priority (HP) and latency sensitive VNFs to utilize any remaining resources. This is known as workload consolidation [7]. While workload consolidation improves server utilization, it also introduces performance-impacting contention on shared resources between the HP VNFs and the BE workloads, which results in possible SLO violations to the VNFs.

*Part of this work was done while Ameer Haj-Ali was in a summer internship at Intel Labs

[1]RLDRM: Reinforcement Learning Dynamic Resource Management

978-1-7281-5684-2/20/$31.00 ©2020 IEEE

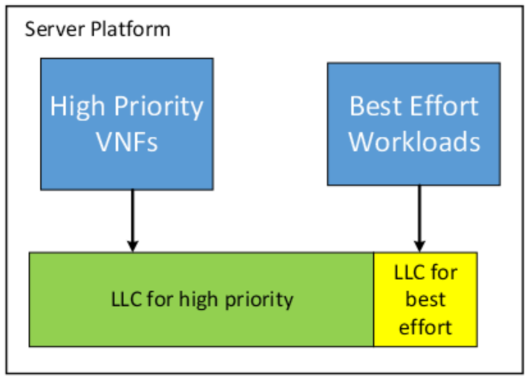

Fig. 1. Example showing HP VNFs and BE consolidation on server platform with Intel® RDT

To alleviate this shared resource contention problem, hardware vendors such as Intel® provides new technologies for users to control hardware resources with finer granularity. Intel Resource Director Technology (Intel® RDT) [8], [9] is one example that enables last-level cache (LLC) and memory bandwidth partitioning between different applications. The proper use of Intel® RDT can isolate the HP workloads and the BE workloads that run on the same platform, which helps increase the server utilization while avoid SLO violations.

One way of utilizing Intel® RDT is to profile the HP workloads offline, finding the resource allocation configurations that satisfy the throughput/latency requirements for the HP workloads at the worst case. The remaining resources are then allocated to the BE tasks as shown in Figure 1. We call this offline profiling based resource allocation method as static resource allocation. The static resource allocation is relatively simpler to set up. However, it is totally unaware of the dynamic

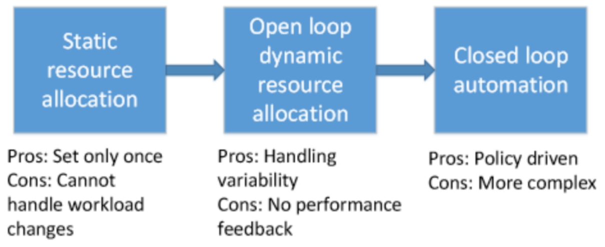
Static resource allocation
Open loop dynamic resource allocation
Closed loop automation
Pros: Set only once
Cons: Cannot handle workload changes
Pros: Handling variability
Cons: No performance feedback
Pros: Policy driven
Cons: More complex

Fig. 2. Resource allocation approaches summary

workload variance during run time.

In real data center environment, the amount of traffic processed by the networking servers typically has some coarse grained time-based patterns. For example, during the day time (8am to 5pm for example), there is more traffic as the users are active. And during night time (5pm to 8am for example), the traffic becomes lighter. Based on this observation, operators can set up an open-loop dynamic controller to change the RDT configuration based on day time and night time period, so that the HP and BE applications can get different shares of the hardware resources at different time. However, there is no performance feedback in this open-loop controller. As a result, it cannot handle the variance of traffic patterns on each day, and cannot cope with unexpected situations. It also requires human experience and data center statistics to decide which time period to profile, which confines the method to be coarse-grained only.

The static resource allocation and open-loop dynamic resource allocation methodologies leave a lot to be desired. The industry is asking for a closed-loop, online automation approach that automatically tunes the hardware resources at run time with considering complex workload behavior, and operates continuously and promptly to account for the application demands. The evolvement from static resource allocation to open-loop dynamic resource allocation, and to closed-loop automation are summarized in Figure 2.

In this paper, we propose a closed-loop automation framework RLDRM to dynamically adjust the last level cache (LLC) allocation between the HP VNFs and the BE workloads using deep reinforcement learning (RL). In deep RL, an agent observes an environment and interacts with it by taking actions. From these actions, the agent receives rewards and observations. The agent's ultimate goal is to learn a policy that maximizes the long term reward. In this work, the agent's goal is to ensure that the HP workloads meet SLO (*e.g.*, packet loss or throughput target) while maximizing the performance for the BE tasks, and thus to improve the server utilization and reduce the TCO. To the best of our knowledge, this is the first study that applies deep RL techniques to tackle dynamic cache allocation problem in real-time control system, and demonstrates the potential of deep RL in system resource management and optimization.

Our main contributions are as follows:

- We propose a closed-loop deep RL based resource allocation system RLDRM. The system dynamically allocates hardware resources between HP VNFs and BE workloads. The goal is to allocate the fewest possible resources for the HP workloads that to satisfy their SLOs, and allocating the remaining resources to the BE workloads to improve sever utilization and reduce the TCO.
- We implemented the proposed closed-loop dynamic resource allocation system RLDRM with deep RL on a real server system. Our evaluation results show that deep RL based dynamic cache allocation approach is effective in finding cache allocations that achieve improved performance for BE workloads while maintaining the SLO for HP VNFs.

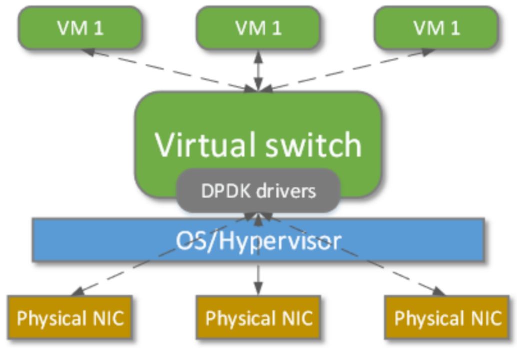

Fig. 3. A diagram showing the virtual switch delopyment with multiple VMs and kernel bypass.

The rest of the paper is organized as follows. Section II provides background information on NFV, Intel® RDT technology and deep RL. It also demonstrates the benefit of applying Intel® RDT for performance isolation between the HP VNFs and the BE workloads. Section III describes the design of our proposed RLDRM framework in detail. Section IV presents the implementation details and evaluation results. Section V reviews prior shared resource management works and works that apply RL for resource management and system scheduling. Finally, Section VI concludes the paper and discusses future works.

II. Background

A. Network Function Virtualization (NFV)

Network Function Virtualization (NFV) is the technology to virtualize hardware network function to virtual network function (VNF) . Compared to traditional hardware black boxes, VNFs can be easily scaled and configured, enabling much shorter development to production time, and reducing the cost of upgrade and maintenance. However, the hardware based networking functions in many scenarios still have significant throughput and latency advantages over VNFs. To this end, industry and academia have developed various techniques to improve the performance of NFV platform. Some examples include software algorithm improvements [10], [11], kernel bypass technologies [12], and in-core hardware accelerators [13], [14]. With these technologies, NFV become more and more popular in data centers. Figure 3 shows an example deployment of a virtual switch running with multiple virtual machines (VMs) through kernel bypass technology

DPDK [12]. In this scheme, networking packets are directly processed by the virtual switch in user space without involving the kernel stack, thus the latency and throughput are improved significantly. Meanwhile, since both of the virtual switch and the VMs are running on the same server platform, there could be undesirable performance interference, which harms the SLOs. In this paper, we are focusing on taking advantage of the state-of-the-art resource allocation technologies to solve the performance interference issues.

B. Intel® RDT Cache Allocation Technology

Modern CPUs use set associative cache, which comprises multiple sets and each set comprises multiple ways. Data that falls into the same set can evict each other following certain replacement policy. For example, one of the most popular replacement policy is least-recently-used (LRU) policy, which means the least recently used cache line is evicted in the case of replacement. In a modern CPU chip, multiple cores share a single logical LLC. Applications that are running simultaneously on different cores will compete for the limited LLC capacity. In other words, one application may evict the useful data of another application from the same cache set, thus interfere the performance of each other.

To alleviate the resource contention problem in multicore system, numerous software and hardware solutions have been proposed [15]–[26]. One of the most mature technologies that is available today is Intel® Resource Director Technology or Intel® RDT [8], [9]. Intel® RDT provides the capability to partition LLC to restrict the usage of each co-running application. This cache partitioning capability is also known as CAT, or Cache Allocation Technology [8]. CAT controls a fraction of the capacity based on classes of services (CLOS), which may map to ways [8], [9]. Each CLOS consists of a group of cores or threads that share the LLC ways allocated to this CLOS. Each CLOS can be mapped to one or more LLC ways. And each LLC way can be used by one or more CLOS. When a thread is assigned to a CLOS, it is only allowed to allocate data in the cache ways belonging to its own CLOS. Thus, the thread can never evict data of other applications that belong to a different CLOS and mapped to different cache ways.

By using Intel® RDT, one can guarantee the performance SLOs of the critical networking functions while allowing the BE applications to run on the same platform. In this paper, we propose a novel closed-loop controller to dynamically balance the resources between the HP and BE workloads by leveraging the ability to update Intel® RDT configurations during run time.

C. Deep RL

RL is an important area in machine learning, where an agent interacts with an environment and learns to take actions that would maximize cumulative rewards [27]. RL does not learn from labeled input/output pairs as in supervised learning. Instead, it learns based on its own interaction with the environment. Using a deep neural network to learn the policy that optimizes the actions is called deep RL. Recent advancements in deep RL can solve complex problems, learn complex functions, or predict actions in states that were not seen previously [28], [29]. In more details, RL has an agent and environment in the system as shown in Figure 4. The agent takes an action and then observes the state and reward from the environment. The reward is then used to improve the policy of the agent. By trial and error, the agent learns to take actions that would yield the most cumulative long term rewards.

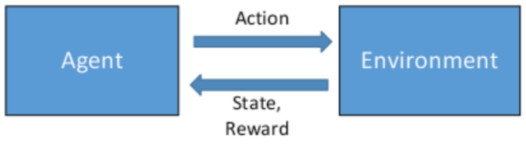

Fig. 4. Illustration of the standard RL architecture

In this work, we use Deep Q-Network (DQN) [30] based algorithm to learn the deep RL policy. DQN has been applied to computer games such as the Go game and Atari games [28], [29]. DQN bootstraps and learns a Q-function, which estimates the long term reward from taking an action, which improves its sample efficiency. Two techniques has been proposed to further improve the performance and stability of DQN: (1) Double DQN [31]: The action choice and target Q-value generation is decoupled, which mitigates the overestimation of the Q value as in DQN. (2) Dueling DQN [32]: the value function and advantage function are separately computed and then combined into a single Q-function. This improvement results in better policy evaluation. Combining these two techniques can achieve better performance and faster convergence.

DQN and its variances have been used to tackle problems in system optimization such as dynamic power management [33], [34], and resource allocation in the cloud [35]–[37]. In this paper, we developed a deep RL based methodology to dynamically control LLC resource allocation in server platform based on traffic load. Unlike most prior works that use simulators to evaluate the system, we evaluate and run our deep RL algorithms in a real system.

D. Benefit of Intel® RDT

In this section, we show the benefit of applying static LLC capacity allocation between a critical HP VNF and a BE workload for performance isolation, as well as potentials of dynamically partitioning the LLC capacity between HP VNF and BE workload to further improve server utilization.

To mimic workload consolidation scenario in networking platform, we run a DPDK-based IPv4 traffic forwarding benchmark as the HP VNF workload to process contiguous incoming traffic. We then co-run a BE workload omnetpp from SPEC CPU2006 benchmark suite [38] on the same platform (details about the workloads and experiment setup can be found in Section IV). We first allow the two workloads to run without any RDT partitioning. In this experiment, the HP VNF and BE workload compete for the LLC cache resource and interfere with each other due to contention. This

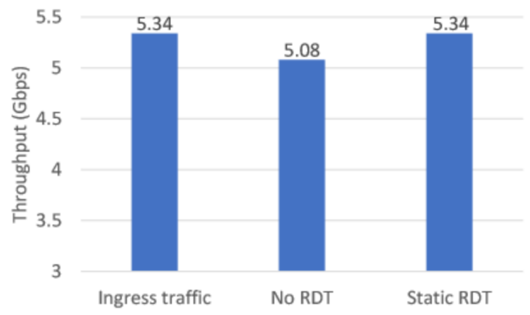

Fig. 5. HP VNF Performance Comparison (higher is better). *No RDT*: applications share LLC. *Static RDT*: Static LLC partition between applications. Partition remains the same during execution.

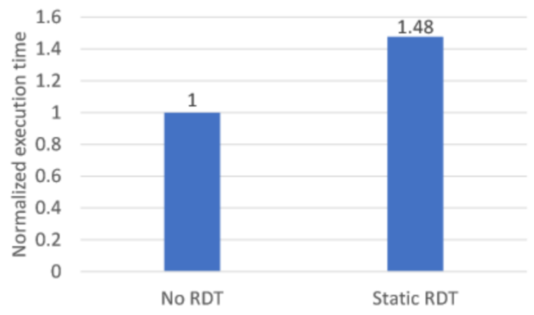

Fig. 6. BE SPEC omnetpp Execution Time Comparison (lower is better)

is referred to as *No RDT* scenario. Our experiment shows that the throughput of the high priority VNF can degrade by 4.9% due to the cache contention as shown in Figure 5.

We then statically allocate nine LLC ways to the HP VNF workload, and allocate the remaining two LLC ways to the BE workload. The LLC way allocation between the HP VNF and BE workload remains to be the same throughout the execution. This is referred to as *Static RDT* scenario. In this case, the BE workload will not interfere with the HP VNF workload so that the performance of the HP VNF can be guaranteed. With this static cache way enforcement, the performance of the HP VNF has been restored to its peak performance as shown in Figure 5. This demonstrates that the isolated LLC way allocation is effective in protecting performance for HP VNF. On the other hand, the BE workload's performance drops significantly. The execution time is now increased by 48% when applying static RDT allocation compared to when there is no RDT restriction as shown in Figure 6. This is expected as the BE workload is now restricted to use only two LLC ways.

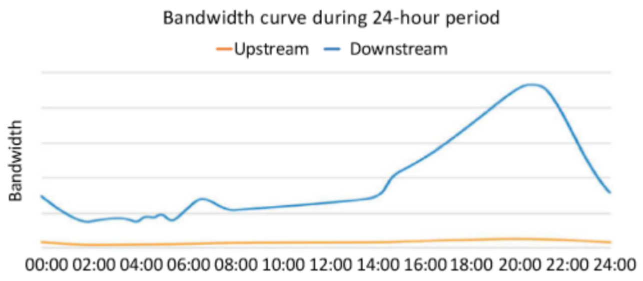

Fig. 7. Network traffic 24-hour pattern

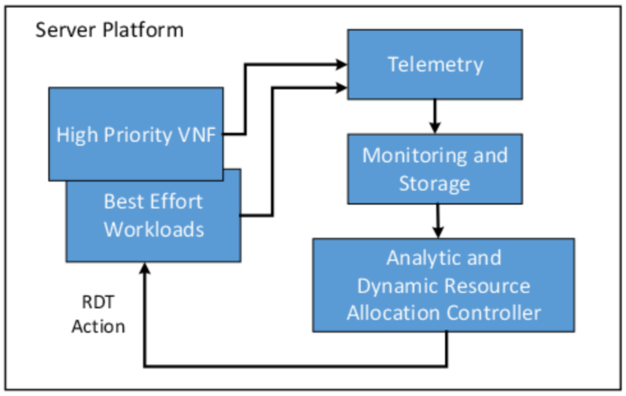

Fig. 8. RLDRM: Closed-loop dynamic resource allocation framework.

In real deployment, the load of networking traffic varies significantly over time. Figure 7 shows a typical 24 hour traffic fluctuation in networking [33]. Conservative static allocation targeting worst case usage protects HP workloads. However, it reduces the opportunities to achieve higher performance for BE workloads, and hence higher server utilization. Dynamic resource allocation based on the fluctuated traffic condition and platform utilization is thus desirable. With dynamic RDT allocation, we can guarantee the performance of the critical HP VNF workloads while allocating as much resource as possible to the BE workloads. How to determine the amount of RDT resource to allocate at run time with varying traffic remains a challenge.

In this paper, we propose to use deep RL to address the challenge, where the RL algorithms can learn the RDT allocation policy in a dynamic environment and take actions proactively. Deep RL has the following capabilities: it learns the optimal policy to achieve defined goals by itself, it can adapt to changing environment, and it is capable of handling varying workload under different situations. As a result, deep RL fits the dynamic resource allocation task well.

III. Closed-loop Dynamic RDT Resource Allocation Design

In this section, we present our closed-loop dynamic RDT allocation framework RLDRM. Although the framework focuses on VNFs, the general concept can also be applied to other use cases such as partitioning cloud applications.

A. *RLDRM Framework Overview*

Figure 8 shows the RLDRM framework of our closed-loop system for dynamic hardware resource allocation with deep RL. It works for a platform that runs both HP and BE workloads. The HP VNF workloads are usually the user facing, latency critical workloads. Meanwhile, system schedulers schedule the BE workloads on the same server to improve server utilization. The telemetry tool periodically collects telemetry data (such as platform performance counters, application throughput, *etc*). The telemetry data are then stored and processed further by the Analytic and Dynamic Resource Allocation Controller to make decision for the resource allocation for the next time window.

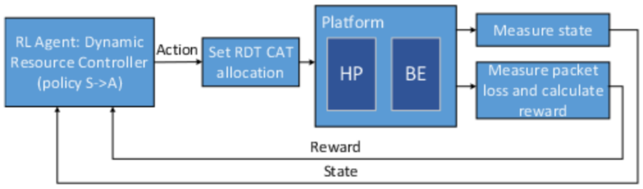
RL Agent: Dynamic Resource Controller (policy S->A)
Action
Set RDT CAT allocation
Platform
HP
BE
Measure state
Measure packet loss and calculate reward
Reward
State

Fig. 9. Deep RL design for RDT allocation

B. Deep RL design for dynamic RDT allocation

Figure 9 shows an overview of our proposed deep RL based framework for dynamically controlling RDT allocation among multiple workloads at run time. The RL agent continuously interacts with the system and learns a policy that maximizes the long term reward.

There are four key components in RL: (1) action, (2) state, (3) policy, and (4) reward. The policy takes the state and outputs the Q-values for each possible action. The action with the maximum Q-value is applied by the agent to the environment. After each action, a new state and reward is obtained from the system. The reward is then used to improve the policy of the agent.

The key challenge of applying deep RL to solve real world problems is to select the right algorithm for the problem, and to define the appropriate states (feature selection), actions, and the rewards. Most prior works that apply deep RL for optimization are simulator based, such as gaming, which do not need to consider algorithm sample efficiency. In our design, since we train the deep RL model on real machines, sample efficiency is a major consideration when choosing the algorithm. Due to this reason, we choose dueling double deep Q-learning (DDDQN) [31], [32] with prioritized experience replay as our deep RL algorithm due to its sample efficiency and stability. We also experimented with the original DQN model as well. However, DDDQN model gives better performance compared to DQN due to its improved performance and stability features as explained in Section II-C.

Below is the detailed design for the DDDQN algorithm for controlling RDT:

Actions A: The action A is the number of RDT LLC ways allocated to HP and BE workloads for the next time window.

State S: The state S consists of the ingress traffic rate for the past N time windows, as well as the current RDT LLC way allocation to the HP VNF and the BE workload.

Reward R: In our design, the reward reflects the goal of allocating the fewest possible LLC ways for the HP workloads with lowest possible packet loss, and allocating the remaining LLC ways to the BE workloads to improve sever utilization. We design the reward function as follows:

$$R_{pkt_loss} = \begin{cases} -m_1 & if\ pkt_{loss} > th_1 \\ -m_2 & else\ if\ pkt_{loss} > th_2 \\ -m_3 & else\ if\ pkt_{loss} > th_3 \\ +m_4 & else\ if\ pkt_{loss} <= th_3 \end{cases} \quad (1)$$

TABLE I
SYSTEM CONFIGURATION

Component	Description
Processor	Intel® Xeon® Platinum 8176 CPU @ 2.10GHz
L1 D-cache	32KB
L1 I-cache	32KB
L2 cache	1MB
L3 cache	38.5MB
Memory	192GB

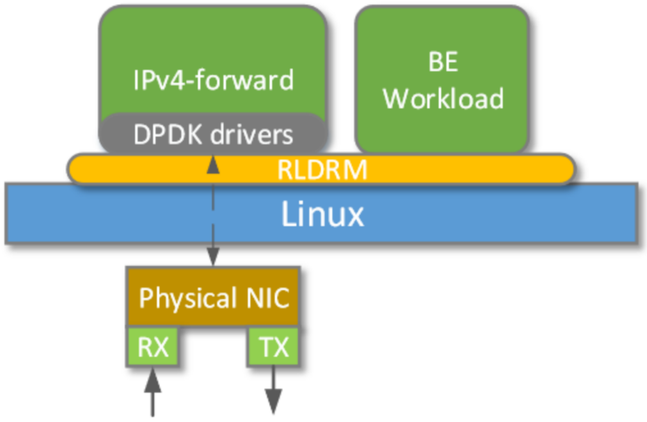

Fig. 10. A diagram showing the evaluation platform.

$$R_{rdt} = \begin{cases} LLC_{HP} & if\ pkt_{loss} > th_3 \\ Total_{LLC} - LLC_{HP} & if\ pkt_{loss} <= th_3 \end{cases} \quad (2)$$

$$R_{total} = R_{pkt_loss} + R_{rdt} \quad (3)$$

Here the pkt_{loss} is the number of packets being dropped during the current time window. R_{pkt_loss} is the reward for packet loss. If the pkt_{loss} is smaller than a predefined acceptable threshold th_3 (can be either zero packet loss or low packet loss depending on the use cases), we assign a positive reward for the R_{pkt_loss}. If the packet loss is above this threshold th_3, we assign a negative reward for the R_{pkt_loss} as penalty. The greater the pkt_{loss}, the bigger the penalty ($m1 > m2 > m3$) is. R_{rdt} is the reward for LLC way allocation. When the pkt_{loss} is smaller than threshold th_3, we give higher reward for using less LLC ways for the HP workloads. When the pkt_{loss} is above this threshold th_3, we give higher reward for using more LLC ways for HP workload. Total reward R_{total} is the sum of R_{pkt_loss} and R_{rdt}, which takes consideration of both packet loss and LLC way allocation.

As the model being trained, it will take the current application and platform parameters as the inputs, and output the RDT LLC way allocation strategy for the next time window.

IV. Evaluation

A. Experiment setup

We evaluate the RLDRM framework on a two-socket server with Intel® Xeon® Platinum 8176 CPU (code name Skylake). The CPU runs at 2.1 GHz and has a 38.5MB LLC with 11

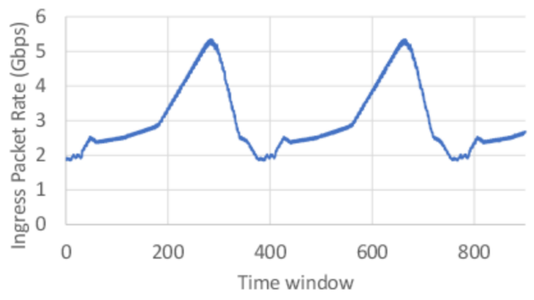

Fig. 11. Ingress network traffic

ways to be partitioned between applications [39]. Detailed system configuration is shown in table I. We run IPv4-forwarding benchmark from DPDK [40] as the HP VNF workload, and omnetpp from SPEC suite [38] as the BE workload. omnetpp is a simulation software to simulate large communication systems which we find it to be cache sensitive. Each cache way capacity is 3.5 MB. The server is connected to a TRex [41] traffic generation server with two 10GbE network interfaces. To compare with the deep RL based dynamic allocation algorithm, the baseline static LLC allocation allocates nine ways to the HP IPv4-forwarding and two ways to the BE omnetpp (baseline static RDT allocation). We found nine LLC ways to the HP VNF can satisfy the QoS requirement for HP IPv4-forwarding at peak throughput. Thus, we allocate the remaining two ways to the BE omnetpp workload.

The IPv4-forwarding workload is a slightly modified version of DPDK L3fwd sample application (the flow pattern in the flow table is modified to match the flow generated by the traffic generator, and the SW prefetcher is turned off to have a more determined behavior). The IPv4-forwarding application works as a L3 router that forwards each packet to a specific port. The forwarding application maintains an internal flow table which is implemented by a hash table. It looks up the five-tuple (i.e. IP addresses, port numbers, and protocol) of each packet from the flow table to decide if the packet should be forwarded or not. We insert 1 million flows into the table which occupies around 24MB memory space.

We implement the proposed closed-loop dynamic RDT allocation framework RLDRM on the target Skylake server platform. We use TRex [41] as our traffic generator. The traffic generator generates network traffic to mimic a 24-hour network traffic pattern as shown in Figure 11. The generated packets belong to the 1-million flows were inserted into the flow table and have random sequence. We choose to collect the states and make resource allocation decision every 2 seconds assuming traffic pattern does not change rapidly within 2 seconds. To match with this, each time window in our experiment is set to 2 seconds.

We use collectd [42] to collect both platform hardware telemetry data as well as the VNF specific application performance data. We configure the collectd to collect data at the end of each 2-second time window. Collectd feeds the collected telemetry data (including the ingress packet rate and packet loss) into a database managed by InfluxDB [43]. The analytic

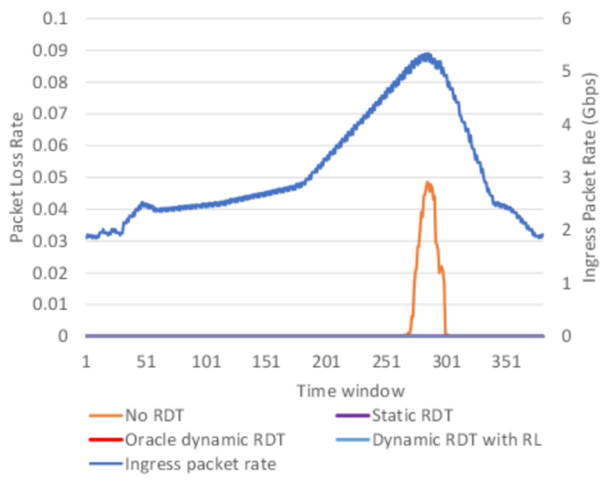

Fig. 12. High priority VNF workload packet drop. Curvers for static RDT, oracle dynamic RDT, and dynamic RDT with deep RL are overlapping with each other

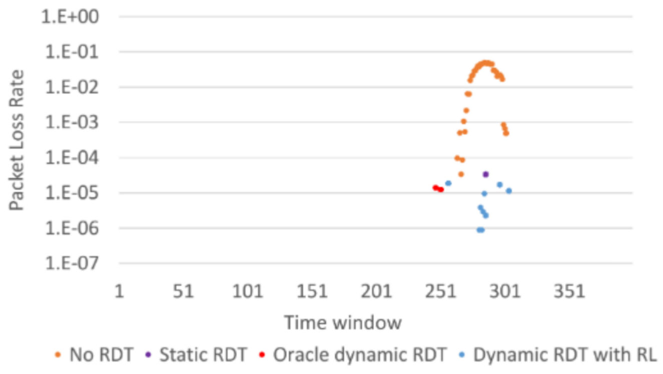

Fig. 13. High priority VNF workload packet drop in log scale

agent then reads the data from InfluxDB and issues actions every 2 seconds. Figure 10 shows the simplified scheme of the platform that we use to evaluate RLDRM.

Our designed DDDQN model has two dense layers each having 256 neurons. The input *state* includes the ingress traffic for the previous N=40 time windows and the current RDT allocation, which forms a 41 feature input to the model. We experimented with multiple values for the N past time windows, and found N=40 to be a good setting that balances between accuracy and overhead. The output *action* is the next time window's RDT allocation. The RDT allocation for HP VNF ranges from two ways to nine ways (nine ways is the default setting). The remaining LLC ways are then allocated to BE workload (ranges from nine ways to two ways). This gives totally eight discrete actions. To run our deep RL algorithms, we use RLlib [44], a unified open-source library that provides scalable software primitives for RL. RLlib is built on top of Ray [45], a high-performance distributed execution framework. The initial learning rate is set to 0.001.

B. Evaluation results

We compare the performance of the HP IPv4 forwarding VNF and the BE workload omnetpp from SPEC in four scenarios. The first experiment is when the HP VNF and

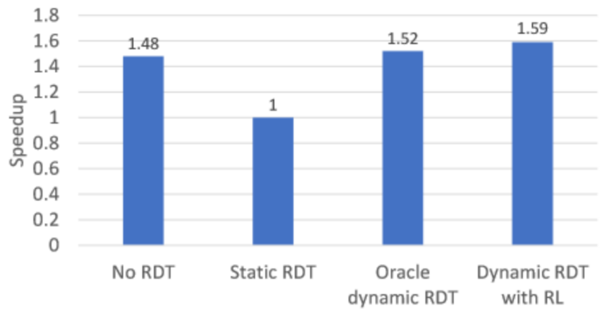

Fig. 14. Best effort workload speedup (higher is better)

the BE workload share the LLC resources without any RDT enforcement. The second experiment is applying the static RDT allocation policy between the HP VNF and the BE workload (nine LLC ways are allocated to HP VNF and two LLC ways are allocated to BE). This is our baseline scenario. The third experiment is the oracle policy which always leave the maximum possible LLC ways to BE while maintain the throughput of HP VNF. The fourth experiment is our proposed dynamic RDT allocation scheme using deep RL.

Figure 12 shows the packet drop rate as the metric in the above mentioned four scenarios. From the figure, we can see that when there is no RDT applied in the system, the HP VNF and the BE workload compete for the shared resource. As a result, HP VNF experiences significant packet drop (4.9%) when the ingress packet rate is high. Meanwhile, BE performance increases by 48% compared to baseline static RDT case as shown in Figure 14. When applying static RDT policy in the system, HP VNF's packet drop rate is minimal (zero packet drop for most of the times, happens at 10^{-5} rate occasionally). On the other hand, BE workload has the lowest performance since it has limited LLC resources now, which is expected.

When we apply the oracle dynamic RDT allocation policy in the system based on ingress traffic rate, where RDT LLC way allocation between HP VNF and BE workload is determined at the optimal point, the packet drop rate for HP VNF is close to static allocation case as shown in Figure 12 (zero packet drop for most of the times, packet drop happens occasionally at the order of 10^{-5} drop rate when ingress packet rate is high). Figure 13 shows the packet drop rate for HP VNF in log scale. BE performance has been improved by 52%. This demonstrates that with oracle dynamic resource allocation policy, one can guarantee the SLO of HP VNF while improve BE workload's performance significantly, thus improving server utilization.

We then apply the proposed deep RL based methodology to train the system to self-learn the RDT allocation policy for the dynamically injected traffic. By running the trained deep RL model as the controller for dynamic RDT allocation based on ingress traffic, BE workload's performance has been improved by 59% compared to the baseline static RDT allocation while packet drop rate for HP VNF is maintained at a similar range as in baseline and oracle (happens occasionally between 10^{-7} to 10^{-5} range). When we look at the packet drop rate in Figure 13, we can see that the packet drop occurs more times in deep RL based model than in oracle, which means that the trained deep RL model is a bit more aggressive in minimizing cache allocation for HP VNF than oracle, thus causing more packet drops for HP VNF. Even though, the packet drop rate in deep RL based methodology is still in the same range as in oracle. On the other hand, BE benefited from having more cache allocation, resulting in higher performance improvement as shown in Figure 14 (59% improvement in deep RL based model while 52% improvement in oracle). In summary, our experiment result demonstrates that the proposed deep RL methodology is effective in learning the dynamic RDT allocation policy which is very close to the oracle policy.

It is worth noting that although one could come up with heuristic policies like the oracle one we use in the experiments without machine learning, it requires enormous human effort to profile the workloads. Deep RL fully automates the process, and thus it is more scalable and practical.

V. Related Work

There are many prior studies for architectural and system support to partition shared resources, which provide performance isolation, improve fairness, or maximize system throughput [15]–[26]. These prior works have paved the way for exploring QoS and shared resource management. Some of the proposals have been implemented in commercial processors [21].

Recently, there have been works that explore dynamic resource management techniques for workload consolidation in servers where user-facing workloads and background best effort workloads are running simultaneously on the platform [6], [46], [47]. For example, Heracles [6] develop heuristics to isolate latency-sensitive jobs while maximizing resources for best effort jobs and applying Intel® RDT technology for dynamic cache allocation. However, the cache allocation policy is adjusted gradually instead of being promptly set to the optimal point, thus cannot react quickly enough to short-term behavior. Dirigent [46] relies on profiling the latency-critical application offline first, and then predicts the completion time at run time, and makes resource allocation decision accordingly. While effective, this approach can be hard to be generalized to different workloads. Fast and prompt response is critical for networking workloads since sudden packet loss is unacceptable in many cases. Our deep RL based algorithm learns to react ahead of time to prevent packet loss from happening.

Recent advancement in deep RL provides new opportunities to optimize resource management and scheduling [33], [48]–[53]. Most of the prior works focus on resource allocation for job scheduling and power management, which is orthogonal to our work. There has been very limited work to explore RL for dynamic resource allocation in a fine granularity.

VI. Conclusion

In this paper, we demonstrate the use of Intel® RDT technology to help with efficient workload consolidation. We show that the static RDT allocation can be too conservative. We proposed a closed-loop dynamic RDT allocation system RLDRM. The RLDRM system consists of telemetry collection and analysis, and generates action based on a deep RL algorithm. Our experiment results show that deep RL based dynamic RDT allocation approach is effective in finding dynamic RDT allocation policy, resulting in improved performance for BE workloads while maintaining the required SLO for HP VNFs. Future research plan includes exploring more benchmarks and various deep RL algorithms to balance sample efficiency and computation complexity given more complex systems. We also plan to study online model update to track the changing traffic pattern and operation environment.

REFERENCES

[1] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang, "Network Virtualization in Multi-Tenant Datacenters," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, Seattle, WA, apr 2014.

[2] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, Savannah, GA, Nov. 2016.

[3] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling Network Function Parallelism in NFV," in *Proceedings of the 2017 ACM SIGCOMM Conference (SIGCOMM'17)*, Los Angeles, CA, aug 2017.

[4] G. P. Katsikas, T. Barbette, D. Kostić, R. Steinert, and G. Q. M. Jr., "Metron: NFV Service Chains at the True Speed of the Underlying Hardware," in *Proceedings of 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, Renton, WA, Apr. 2018.

[5] F. Xu, H. Zheng, H. Jiang, W. Shao, H. Liu, and Z. Zhou, "Cost-effective cloud server provisioning for predictable performance of big data analytics," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1036–1051, May 2019.

[6] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Improving resource efficiency at scale with heracles," *ACM Trans. Comput. Syst.*, vol. 34, no. 2, pp. 6:1–6:33, May 2016.

[7] X. Liu, C. Wang, B. B. Zhou, J. Chen, T. Yang, and A. Y. Zomaya, "Priority-based consolidation of parallel workloads in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1874–1883, Sep. 2013.

[8] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer, "Cache qos: From concept to reality in the intel® xeon® processor e5-2600 v3 product family," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 657–668.

[9] "Intel 64 and IA-32 Architectures Software Developer's Manual."

[10] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, Oakland, CA, May 2015, pp. 117–130.

[11] Y. Wang, T. C. Tai, R. Wang, S. Gobriel, J. Tseng, and J. Tsai, "Optimizing open vswitch to support millions of flows," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017, pp. 1–7.

[12] Intel Corporation, "Data Plane Development Kit (DPDK)," https://www.dpdk.org, 2018.

[13] Y. Yuan, Y. Wang, R. Wang, and J. Huang, "Halo: Accelerating flow classification for scalable packet processing in nfv," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 601–614.

[14] Intel Corporation, "Intel® Data Direct I/O (DDIO)," https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html, 2018.

[15] S. Cho and L. Jin, "Managing distributed, shared l2 caches through os-level page allocation," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, Dec 2006, pp. 455–468.

[16] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, Dec 2006, pp. 423–432.

[17] R. Iyer, "Cqos: A framework for enabling qos in shared caches of cmp platforms," in *Proceedings of the 18th Annual International Conference on Supercomputing*, ser. ICS '04. New York, NY, USA: ACM, 2004, pp. 257–266.

[18] R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt, "Qos policies and architecture for cache/memory in cmp platforms," in *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '07. New York, NY, USA: ACM, 2007, pp. 25–36.

[19] B. Li, L. Zhao, R. Iyer, L.-S. Peh, M. Leddige, M. Espig, S. E. Lee, and D. Newell, "Coqos: Coordinating qos-aware shared resources in noc-based socs," *J. Parallel Distrib. Comput.*, vol. 71, no. 5, pp. 700–713, May 2011.

[20] B. Li, L.-S. Peh, L. Zhao, and R. Iyer, "Dynamic qos management for chip multiprocessors," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 3, pp. 17:1–17:29, Oct. 2012.

[21] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer, "Cache QoS: From Concept to Reality in the Intel® Xeon® Processor E5-2600 v3 Product Family," in *Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*, Barcelona, Spain, Mar. 2016.

[22] V. Selfa, J. Sahuquillo, L. Eeckhout, S. Petit, and M. E. Gómez, "Application clustering policies to address system fairness with intel's cache allocation technology," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2017, pp. 194–205.

[23] C. Xu, K. Rajamani, A. Ferreira, W. Felter, J. Rubio, and Y. Li, "dcat: Dynamic cache management for efficient, performance-sensitive infrastructure-as-a-service," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: ACM, 2018, pp. 14:1–14:13.

[24] H. Kasture and D. Sanchez, "Ubik: Efficient cache sharing with strict qos for latency-critical workloads," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 729–742.

[25] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and efficient fine-grain cache partitioning," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, June 2011, pp. 57–68.

[26] H. Cook, M. Moreto, S. Bird, K. Dao, D. A. Patterson, and K. Asanovic, "A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 308–319, Jun. 2013.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* The MIT Press, 1998.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 12 2013.

[29] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, Jan 2016.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[31] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 2094–2100.

[32] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Inter-*

national Conference on Machine Learning - Volume 48, ser. ICML'16. JMLR.org, 2016, pp. 1995–2003.

[33] Z. Zhou, T. Zhang, and A. Kwatra, "Nfv closed-loop automation experiments using deep reinforcement learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2019, pp. 696–701.

[34] U. Gupta, S. K. Mandal, M. Mao, C. Chakrabarti, and U. Y. Ogras, "A deep q-learning approach for dynamic management of heterogeneous processors," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 14–17, Jan 2019.

[35] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 31–37, Dec 2017.

[36] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, Jan 2018.

[37] A. Haj-Ali, N. K. Ahmed, T. Willke, J. Gonzalez, K. Asanovic, and I. Stoica, "A view on deep reinforcement learning in system optimization," *arXiv preprint arXiv:1908.01275*, 2019.

[38] Standard Performance Evaluation Corporation, "SPEC CPU 2006 benchmark suite,"
https://www.spec.org/cpu2006/.

[39] Intel Corporation, "Intel® Xeon® Platinum 8176 Processor,"
https://ark.intel.com/content/www/us/en/ark/products/120508/intel-xeon-platinum-8176-processor-38-5m-cache-2-10-ghz.html.

[40] "DPDK Programmer's Guide: Access Control,"
https://doc.dpdk.org/guides/sample_app_ug/l3_forward.html.

[41] Cisco Systems Traffic Generators,
https://trex-tgn.cisco.com/.

[42] "The system statistics collection deamon,"
https://collectd.org.

[43] "Influxdb database,"
https://influxdata.com/influxdb.

[44] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, "Ray rllib: A composable and scalable reinforcement learning library," *arXiv preprint arXiv:1712.09381*, 2017.

[45] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan *et al.*, "Ray: A distributed framework for emerging ai applications," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 561–577.

[46] H. Zhu and M. Erez, "Dirigent: Enforcing qos for latency-critical tasks on shared multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 50, no. 2, Mar. 2016.

[47] C. Xu, K. Rajamani, A. Ferreira, W. Felter, J. Rubio, and Y. Li, "dcat: Dynamic cache management for efficient, performance-sensitive infrastructure-as-a-service," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: ACM, 2018, pp. 14:1–14:13. [Online]. Available: http://doi.acm.org/10.1145/3190508.3190555

[48] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16, New York, NY, USA, 2016, pp. 50–56.

[49] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *Comm. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[50] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, Jan 2018.

[51] C.-Z. Xu, J. Rao, and X. Bu, "Url: A unified reinforcement learning approach for autonomic cloud management," *J. Parallel Distrib. Comput.*, vol. 72, no. 2, pp. 95–105, Feb. 2012.

[52] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2017, pp. 64–73.

[53] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: A reinforcement learning approach to virtual machines auto-configuration," in *Proceedings of the 6th International Conference on Autonomic Computing*, ser. ICAC '09, New York, NY, USA, 2009, pp. 137–146.