T-DCORAL: A Threshold-based Dynamic Controller Resource Allocation for Elastic Control Plane in Software-Defined Data Center Networks

Woojoong Kim, *Student Member, IEEE*, James Won-Ki Hong, *Member, IEEE*, and Young-Joo Suh, *Member, IEEE*

***Abstract*—Existing Elastic Control Planes (ECPs) suffer from the immediacy and the computing overhead issues in Software-Defined Data Center Networks (SD-DCNs). In this letter, we propose a Threshold-based Dynamic Controller Resource Allocation (T-DCORAL) which is a new ECP to mitigate both issues in SD-DCNs. T-DCORAL accelerates/decelerates the control plane by allocating a virtual CPU to controllers in runtime, whereas existing ECPs resize the controller pool. As a result, T-DCORAL maximally reduces the latency to adjust the control plane from 46 seconds to 38 ms, the average CPU load by 22%, and the average rule installation time by 64.28%.**

***Index Terms*—Auto-scaling, Distributed SDN, HotPlug CPU**

I. Introduction

THE Elastic Control Plane (ECP) is an effective way to handle an abrupt change of the control traffic in Software-Defined Data Center Networks (SD-DCNs). Most previous control planes (CPs) have used the static number of distributed controllers, which we called Static CPs (SCPs). SCPs cause the delay of rule installation time and the dissipation of CPU resources which represent the number of virtual CPUs (vCPUs) and the CPU capacity. Increasing the control traffic, controllers face increased control messages. Then, each control message for a rule installation is processed tardily; the rule installation time delays. Decreasing the control traffic, SCPs use excessive controllers, i.e., wasting CPU resources. Hence, the ECP is necessary to adjust the CP, dynamically [2].

To adjust the CP, previous ECPs have resized the controller pool the set of Active controllers. Dynamic Controller Provisioning (DCP) [1] shrinks/expands the controller pool in Wide Area Networks (WANs) according to control traffic costs. ElastiCon [2] and Efficient Distributed Hash Table (DHT) based Elastic SDN (EDES) [4] resize the controller pool according to the CPU load, while Elastic Adaptive SDN (EAS) rescales the controller pool in conformity with the network load. Profile-based Dynamic Distributed Controller Provisioning (PDDCP) [5] resizes the controller pool by means of the *network usage profile* modeled by Markov chain.

The main difference among ECPs is the resizing algorithm to expand/shrink the controller pool by activating/deactivating a target controller. This algorithm in DCP, EAS, and PDDCP only changes the CP topology the connection set between each network device and its controller. To activate an Inactive controller, the algorithm reattaches appropriate network devices from Active controllers to the Inactive controller. Deactivating an Active controller, the algorithm makes the Active controller disconnect its network devices and then connect the devices to the other Active controllers [1], [3], [5]. The resizing algorithm in ElastiCon and EDES comprise two steps: (i) to change the CP topology; (ii) to switch on/off a controller. This algorithm boots up an Inactive controller right before modifying the CP topology for an activation, whereas the algorithm shuts down an Active controller right after altering the CP topology for a deactivation [2], [4]. Finally, DCP, EAS, and PDDCP always run all controllers; ElastiCon and EDES adapt the number of running controllers.

Due to the difference, we found that ECPs have different drawbacks. DCP, EAS, and PDDCP still overuse CPU resources similar to SCPs because all Inactive controllers are operating. ElastiCon and EDES delay the time to adjust the CP compared to DCP, EAS, and PDDCP, since ElastiCon and EDES wait until the finish to power on/off a controller. Consequently, ElastiCon and EDES transmute the CP slowly, although the control traffic changes rapidly.

In this letter, we propose a Threshold-based Dynamic Controller Resource Allocation (T-DCORAL), a new ECP for SD-DCNs. T-DCORAL accelerates/decelerates the CP, not resizing the controller pool. For an acceleration, T-DCORAL assigns a vCPU to a controller in runtime. Also, T-DCORAL deprives a vCPU from a controller to decelerate the CP in runtime. As a result, T-DCORAL reduces (i) the latency to adjust the CP and (ii) the CPU resources being used.

The key contribution of this letter is three-fold. (i) We categorized previous ECPs into two types and analyzed them. (ii) We initially proposed a new ECP, T-DCORAL, to accelerate/decelerate the CP. (iii) To evaluate T-DCORAL practically, we used ONOS [8] the widely-used SDN controller and implement the orchestrator to run T-DCORAL and other ECPs.

This letter has the following assumptions. Controllers support distributed approach defined in OpenFlow 1.2 and beyond. A VM runs a controller software in a Data Center (DC). All VMs can use multiple vCPUs which can share physical CPUs (pCPUs) with an N:1 vCPU to pCPU ratio.

II. Motivation

Existing ECPs have two algorithms: the rebalancing algorithm to balance the control traffic among controllers by changing the CP topology; the resizing algorithm to scale the controller pool. With the resizing algorithm, we classified them into two types: The type ECP-1 is no need to switch on/off a controller like DCP, EAS, and PDDCP; ECP-1 only amends the CP topology. The type ECP-2 powers on/off controllers and reforms the CP topology like ElastiCon and EDES.

However, there is no definition how to switch on/off controllers in ECP-2. Thus, we define the process to power on/off a controller, intuitively. Shutting down the controller, ECP-2 switches off the controller software first, and then pauses the VM which ran the controller software. To boot

Manuscript received July 26, 2018; revised October 12, 2018; accepted November 7, 2018. This research was supported by the IITP grant funded by the Korea government (MSIT) (2015-0-00575, Global SDN/NFV Open Source Software Core Module/Function Development); supported by the ICT R&D program of MSIT/IITP (2018-0-00749, Development of virtual network management technology based on artificial intelligence); supported by the KETEP and MOTIE of the Republic of Korea (20172510102150); supported by Fire Fighting Safety & 119 Rescue Technology Research and Development Program funded by MPSS (MPSS-FirefightingSafety-2015-78).

W. Kim, J. W.-K. Hong, and Y.-J. Suh are with the Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), Korea email: {woojoong, jwkhong, yjsuh}@postech.ac.kr

up the controller, ECP-2 resumes the VM and then switches on the controller software. Of course, ECP-2 can directly pause/resume the VM without switching on/off the controller software. To see what happens in the above situation, we tested to directly pause/resume the VM running ONOS [8]. When ECP-2 resumed a directly paused VM, the ONOS in the VM still had older states than the others, i.e., inconsistency. Finally, the inconsistent ONOS faced unexpected errors. Thus, ECP-2 needs to switch on/off the controller software for consistency.

We analyzed ECP-1 and ECP-2 with two perspectives: the immediacy and the computing overhead. The immediacy means the latency to scale the controller pool; the computing overhead means how much CPU resources an ECP wastes. To achieve the high immediacy, an ECP agilely adjusts the CP. An ECP with the high computing overhead squanders CPU resources (i.e., the high CPU usage and/or overused vCPUs).

First, ECP-1 has higher immediacy than ECP-2. ECP-1 only has the step to change the CP topology, while ECP-2 includes two steps: (i) the step used in ECP-1 and (ii) the step to switch on/off a controller. Besides, the second step spends longer time than the first step[1]. Thus, ECP-1 finishes altering the controller pool faster than ECP-2. Due to the low immediacy, ECP-2 may face the lack of controllers even during expanding the controller pool, when the control traffic increases.

Next, ECP-2 outperforms ECP-1 in terms of the computing overhead, since ECP-1 keeps all controllers in operation, i.e., overspending the CPU resources. Eventually, the ECP-1 influences the other services (e.g., applications, VMs, containers) in the same Physical Machines (PMs) because all services share pCPUs in the same PM [7]. For instance, Central Office Re-architected as a DC (CORD) is one of Network Function Virtualization (NFV) platform in SD-DCNs [6]. CORD has two types of nodes: computing nodes to run various Virtual Network Functions (VNFs); a head node to manage SD-DCNs and to orchestrate VNFs with SDN controllers and VNF managers, respectively. When the controller overspends CPU resources, VNF managers face the lack of CPU resources.

III. Proposed Scheme

A. System Model

We modeled discrete time series: $T = \{t_1, t_2, ...\}$, where t_c is a time slot and the duration of t_c is τ. We defined the set of k PMs $\mathbf{M} = \{M_1, ..., M_k\}$ and the set of n controllers $\mathbf{C} = \{C_1, ..., C_n\}$. Furthermore, we defined the set of relationship matrices between each PM and each controller: $\mathbf{R} = \{\mathbf{R_1}, ..., \mathbf{R_k}\}$, where ith matrix $\mathbf{R_i} \in \mathbf{R}$ is defined as

$$\mathbf{R_i} = \begin{bmatrix} r_{i,1} & \dots & r_{i,n} \end{bmatrix}, r_{i,j} = \begin{cases} 1 & \text{if } C_j \text{ runs in } M_i \\ 0 & \text{otherwise} \end{cases}.$$

Let $\mathbf{V} = [v_1 ... v_k]$ represent the maximum number of vCPUs for each PM and $\mathbf{P} = \{\mathbf{P^{(1)}}, ..., \mathbf{P^{(k)}}\}$ be the set of vCPU pools for all PMs. The ith vCPU pool $\mathbf{P^{(i)}}$ for M_i is defined as

$$\mathbf{P^{(i)}} = \begin{bmatrix} \mathbf{P_1^{(i)}} \\ \vdots \\ \mathbf{P_n^{(i)}} \end{bmatrix} = \begin{bmatrix} p_{1,1}^{(i)} & \dots & p_{1,v_i}^{(i)} \\ \vdots & \ddots & \vdots \\ p_{n,1}^{(i)} & \dots & p_{n,v_i}^{(i)} \end{bmatrix},$$

[1]The first step spends ~3 seconds. The second step devotes ~46/~13 seconds for switching on/off a controller. Section IV will show it in detail.

$$where\ p_{j,u}^{(i)} = \begin{cases} 1 & \text{if } C_j \text{ has vCPU } u \text{ in } M_i \text{ and } r_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases}$$

and $\mathbf{P_j^{(i)}}$ means the vCPU subpool for C_j in M_i. Each C_j should occupy at least δ vCPUs. The vCPU bitmap $\mathbf{B}(M_i)$ which represents the occupation of vCPUs in M_i is calculated by $\mathbf{B}(M_i) = \mathbf{R_i} \times \mathbf{P^{(i)}}$. We define the number of vCPUs for C_j as

$$n(vCPUs|C_j) = \sum_{i=1}^{k} (\mathbf{P_j^{(i)}} \times \mathbf{J_{v_i \times 1}}),$$

where $\mathbf{J_{v_i \times 1}}$ is a $v_i \times 1$ size all-one matrix $[1...1]_{v_i \times 1}$. Likewise the number of vCPUs in M_i is defined as

$$n(vCPUs|M_i) = \mathbf{B}(M_i) \times \mathbf{J_{v_i \times 1}}.$$

Last, we defined $L(\mathbf{C}) = \{L(C_j)|C_j \in \mathbf{C}\}$ which is the CPU load for each controller:

$$L(C_j) = \frac{\sum_{\text{each vCPU}} vCPU\_load\ [\%]}{n(vCPUs|C_j)},$$

where $vCPU\_load$ is a percentile load of each vCPU in C_j.

B. Goal

Our goal is two-fold: (i) to improve the immediacy and (ii) to reduce the computing overhead. We formulated our goal based on weighted sum model:

$$min\ (\overbrace{\omega_1 T_A}^{\text{term 1}} + \overbrace{\omega_2 \sum_{M_i \in \mathbf{M}} n(vCPUs|M_i)}^{\text{term 2}} + \overbrace{\omega_3 \sum_{C_j \in \mathbf{C}} L(C_j)}^{\text{term 3}})$$
$$\text{s.t. } \omega_1 + \omega_2 + \omega_3 = 1 \text{ and } T_A = \sum_{o_k \in \mathbf{O}} T_{o_k},$$

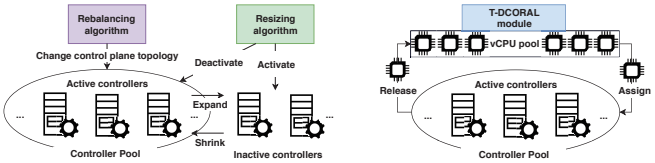
where ω is the weight value, T_A is the latency to adjust the CP, $\mathbf{O}$ is the set of operations to adjust the CP such as expanding/shrinking the controller pool, and T_{o_k} is the latency to run the operation o_k. The term 1 minimizes the latency to adjust the CP, which increases the immediacy. The term 2 and 3 minimize the number of used vCPUs and the CPU load (reducing the CPU resources being used), which is for reducing the computing overhead.

To achieve our goal, an ECP needs to abide by following policies. (i) The ECP should not switch on/off controllers for minimizing the latency to adjust the CP (for term 1). (ii) The ECP imposes upon the minimum number of controllers to reduce the computing overhead (for term 2 and 3). (iii) Those minimal controllers are able to process all control messages.

C. Threshold-based Dynamic Controller Resource Allocation

Based on the above policies, we propose a new EPC, a Threshold-based Dynamic Controller Resource Allocation (T-DCORAL). Basically, T-DCORAL exploits the minimal controllers that are not switched on/off. The crux of T-DCORAL is to accelerate/decelerate the CP, not resizing the controller pool to handle an abrupt change of the control traffic. For an acceleration/deceleration, T-DCORAL assigns/reclaims a vCPU to/from a controller in runtime. In fact, the more vCPUs are on each controller, the more control messages can be processed (i.e., the higher controller throughput) [2]. That is the reason why T-DCORAL allocates vCPUs in runtime.

Literally, T-DCORAL accelerates/decelerates the CP according to thresholds. T-DCORAL has two threshold values: the upper threshold (η_u) and the lower threshold (η_l). With the CPU load or network load more than the upper threshold



(a) Previous ECPs (b) T-DCORAL

Fig. 1: Overviews of previous ECPs and T-DCORAL

Algorithm 1 Threshold-based DCORAL

1: **procedure** run_TDCORAL:
2: **for every timeslot** $t_c \in T$ **do**
3: $L(\mathbf{C}) \leftarrow$ getMonitorResults($t_c - 1$); init $\mathbf{A} \leftarrow [a_1 ... a_n]$;
4: **for each** $C_j \in \mathbf{C}$ **do**
5: $a_j \leftarrow (L(C_j) > \eta_u)\,?\,1 : (L(C_j) < \eta_l)\,?\,-1 : 0$;
6: **end for**
7: **for each** $\mathbf{R_i} \in \mathbf{R}$ **do**
8: release_vCPU($\mathbf{R_i}$, $\mathbf{A}$ XOR $(-1) \cdot \mathbf{R_i}$);
9: assign_vCPU($\mathbf{R_i}$, $\mathbf{A}$ XOR $\mathbf{R_i}$);
10: **end for**
11: **end for**
12: **function** release_vCPU($\mathbf{R_i}$, $\hat{\mathbf{A}}$):
13: **for each** $a_j \in \hat{\mathbf{A}}$ & $a_j = -1$ & $n(vCPUs|C_j) > \delta$ **do**
14: u = Last 1 in $\mathbf{B}(M_i)$; Release u of C_j in M_i; $p^{(i)}_{j,u} \leftarrow 0$;
15: **end for**
16: **function** assign_vCPU($\mathbf{R_i}$, $\hat{\mathbf{A}}$):
17: **for each** $a_j \in \hat{\mathbf{A}}$ & $a_j = 1$ & $n(vCPUs|M_i) < v_i$ **do**
18: w = First 0 in $\mathbf{B}(M_i)$; Assign w to C_j in M_i; $p^{(i)}_{j,w} \leftarrow 1$;
19: **end for**

(less than lower threshold), most existing ECPs, e.g., ElastiCon, EDES, and EAS, expand (shrink) the controller pool. Likewise, T-DCORAL accelerates (decelerates) the CP when the CPU load of each controller is more than η_u (less than η_l).

Fig. 1 shows overviews of existing ECPs and T-DCORAL. Existing ECPs consist of the rebalancing algorithm and the resizing algorithm (Fig. 1a). However, T-DCORAL comprises only one module, T-DCORAL module. This module is responsible for the management of the vCPU pool which includes spare vCPUs. Also, this module assigns/reclaims spare vCPUs to/from target controllers to accelerate/decelerate the CP.

Algorithm 1 describes T-DCORAL running on the T-DCORAL module in Fig. 1b. The procedure *run_TDCORAL* runs T-DCORAL in every timeslot $t_c \in T$ (line 2-11). At t_c, this procedure initially gets the CPU load of each controller monitored at the right before timeslot t_{c-1}, and defines an array $\mathbf{A} = [a_1 ... a_n]$ (line 3). Next, T-DCORAL classifies controllers into three groups: (i) $L(C_j)$ exceeds η_u; (ii) $L(C_j)$ is beneath η_l; (iii) the others. If a C_j is corresponds to the first (second) group, T-DCORAL marks a_j as 1 (-1). Otherwise, a_j is 0 (line 5). Finally, T-DCORAL calls *release_vCPU* and *assign_vCPU* for each controller which corresponds to the first ($a_j = 1$) and second group ($a_j = -1$), respectively (line 8-9).

The *release_vCPU* and *assign_vCPU* release and assign a vCPU from and to target controllers, respectively (line 12-19). Releasing a vCPU from C_j which owns vCPUs more than δ, the *release_vCPU* searches the vCPU u which is the last 1 in $\mathbf{B}(M_i)$. Then, the function releases the vCPU u from C_j and updates the vCPU pool (line 14). On the other hand, the *assign_vCPU* operates to assign a vCPU to C_j, when the vCPU pool is not empty, i.e., $n(vCPUs|M_i) < v_i$. To assign a vCPU to C_j, the function figures out the vCPU w which is the first 0 in $\mathbf{B}(M_i)$. Finally, the function assigns the vCPU w to C_j and updates the vCPU pool (line 18).

To allocate vCPU in runtime, we found two ways. The first way is to use *CPU HotPlug* functions supported by VM Monitors (VMMs) such as VirtualBox. The next way is to change Linux system files, which specify target vCPUs as online/offline in each VM. For example, let the vCPU pool contains n vCPUs for m VMs running Ubuntu. Then, we initially allow that each VM can take advantage of all n vCPUs. To exploit k vCPUs for each VM ($mk \leq n$), we then specify $n - k$ vCPUs as offline and k vCPUs as online. For specifying vCPUs as online or offline, we modify Linux system files *online* and *offline*[2]. In this letter, we use the second way to support all VMMs.

IV. Performance Evaluation

We evaluated T-DCORAL with four generalized ECPs derived from ElastiCon, EDES, and EAS: (i) ECP-1 according to the CPU load (CPU1), (ii) ECP-2 according to the CPU load (CPU2) to cover ElastiCon and EDES, (iii) ECP-1 according to the network load (NET1) to cover EAS, and (iv) ECP-2 according to the network load (NET2). The rebalancing algorithms in ElastiCon and EAS were used in CPU1/CPU2 and NET1/NET2, respectively. We did not analyze T-DCORAL with DCP and PDDCP. DCP is for WAN and PDDCP considers the traffic patterns in WAN, while T-DCORAL aims at SD-DCNs generating the random traffic [11]–[14].

We implemented an orchestrator to run all ECPs and to monitor the CPU and control traffic loads with VirtualBox and OFMon [9], respectively. We conducted our experiment for 130 5-second timeslots[3]. As there was no definition to select thresholds, we experimentally determined parameters: η_u = 60% and η_l = 40% for CPU1/CPU2 and T-DCORAL; η_u = 42 Mbps and η_l = 28 Mbps for NET1/NET2[4]. We used a PM ($k = 1$) to operate 9 VirtualBox VMs which performed ONOS [8] ($n = 9$) with $\delta = 2$ and $v_1 = 18$. Starting the experiment, we set 3 Active controllers. Each controller had 2 vCPUs and serviced the same number of network devices. We assumed that all ECPs ran at least 3 Active controllers[5].

For the data plane, we used Mininet to emulate six 6-pod Fat-Tree [10] topologies (270 switches and 324 hosts). We used iPerf3 to generate the DC traffic with regards to [11]–[14], where each TCP flow had 100 KBps for 10 seconds. To observe the adjustment of the CP, we defined the Hill scenario: (i) linearly increasing the number of flows from 0 to 1,800 at every 10 seconds until the middle of our experiment; (ii) after that, linearly decreasing the number of flows from 1,800 to 0 at every 10 seconds until the end of our experiment.

TABLE I shows the time to adjust the CP for each ECP. CPU1/NET1 spent ~3 seconds for all operations, because they only modified the CP topology. CPU2/NET2 expended

[2]Those files are located in */sys/devices/system/cpu/cpu<index>/* directory.

[3]Since the latency of monitoring + rebalancing is up to 5 seconds, $\tau = 5$.

[4]In our experiment, each controller faced up to 70 Mbps. Accordingly, we defined η_u = 42 Mbps ($70 \times 60\%$) and η_l = 28 Mbps ($70 \times 40\%$).

[5]Even if we can use 1-2 controllers, we used at least 3 controllers to achieve not only scalability but reliability. Using 1-2 controllers, SD-DCNs become unreliable when controllers crash.

ECPs	Shr/Dec	Exp/Acc	Reb
CPU1	3179.17	2811.67	2629.63
CPU2	12945.50	46992.17	3032.88
NET1	3330.49	3246.73	3252.97
NET2	13228.20	46403.80	3157.04
T-DCORAL	32.08	38.55	N/A

TABLE I: Time for operations [ms]

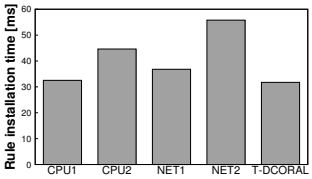


Fig. 2: Avg. rule installation time

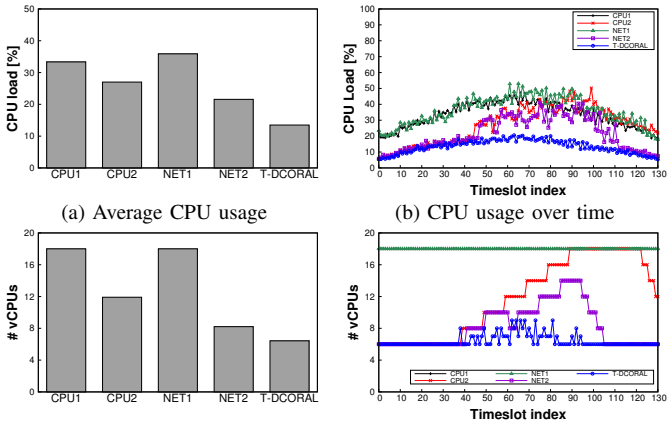
~3, ~13, and ~46 seconds for the rebalancing, shrinking, and expanding operations, respectively. In particular, those ECPs spent (i) ~3 seconds to reform the CP topology, (ii) ~10 seconds to switch off the controller software, and (iii) up to 100 ms to pause the VM, when shrinking the controller pool. In contrast, CPU2/NET2 consumed (i) up to 100 ms to resume the VM, (ii) ~43 seconds to boot up the controller software, and (iii) ~3 seconds to change the CP topology, when expanding the controller pool. We saw that CPU2/NET2 spent the most time to boot up the controller software due to the three substeps: (i) to switch on the controller software, (ii) to associate with other controllers, and (iii) to associate with all network devices. T-DCORAL, however, required just ~30 ms to adjust the CP because T-DCORAL just modified Linux system files to allocate/release vCPUs.

Fig. 2 illustrates the average rule installation time which is the average delay to install flow rules from each controller into target network devices. This time can be calculated by the time difference between the transmission time of *PACKET_IN* message and the reception time of *PACKET_OUT* message in each target network device. As a result, T-DCORAL suffered from the average rule installation time very similar to CPU1/NET1 which always occupied the maximum number of vCPUs. However, T-DCORAL experienced 29.9% and 43.1% faster average rule installation time than CPU2/NET2.

In Fig. 3a,b which depict the result of the CPU usage, T-DCORAL outperformed other ECPs. Compared to CPU1, NET1, CPU2, and NET2, T-DCORAL reduced the CPU usage up to 19.85%, 22.40%, 13.51%, and 8.09%, respectively (Fig. 3a). In Fig. 3b, CPU1/NET1 always experienced the CPU load more than the T-DCORAL case due to the guest OS, controller software, and threads to process inter-controller and OpenFlow messages. In the beginning, CPU2/NET2 experienced the CPU usage similar to the CPU usage with T-DCORAL. Increasing the number of flows, CPU2/NET2 increased the CPU usage to be similar to the CPU usage with CPU1/NET1, gradually.

In Fig. 3c,d which show the result of the vCPU usage, T-DCORAL surpassed other ECPs. T-DCORAL utilized the average of 6.35 vCPUs, which is the average number of vCPUs 64.28%, 46.01%, 64.28%, and 21.68% less than CPU1, CPU2, NET1, and NET2 cases, respectively (Fig. 3c). In Fig. 3d, CPU1/NET1 always used 18 vCPUs, while CPU2/NET2 changed the number of vCPUs with the *stair-shaped* graph due to the latency for shrinking/expanding the CP. T-DCORAL adjusts the number of vCPUs with a sharp fluctuation, because of the low latency to accelerate/decelerate the CP. Thus, T-DCORAL used the CPU resources less than other ECPs; we can use more services with T-DCORAL than other ECPs.

Finally, we analyzed the time complexity of all ECPs. Let n, m, and k be the number of controllers, network devices, and PMs, respectively. The time complexity of ElastiCon, EDES, and EAS are $O(n^m)$, $O(nm)$, and $O(n^2m)$ respectively. Those ECPs spent the most time in the rebalancing algorithm, not the resizing algorithm. Since T-DCORAL has no rebalancing algorithm and contains the loop with n and k, the time complexity of T-DCORAL is $O(kn)$. Since T-DCORAL uses minimal controllers ($n = 3$) and k is up to n (1 controller on 1 PM), the time complexity is similar to $O(9) \approx O(1)$.



(a) Average CPU usage (b) CPU usage over time

(c) The average number of vCPUs (d) The number of vCPUs over time

Fig. 3: CPU Resources for each scheme

V. Conclusion

In this letter, we propose T-DCORAL a new ECP accelerating/decelerating the CP. T-DCORAL outshined pervious ECPs in terms of the immediacy and the computing overhead. However, T-DCORAL cannot allocate a vCPU when the vCPU pool is empty. Also, T-DCORAL only assigns/releases a single vCPU even if the CP experienced a surge of the control traffic. Next, we studied no rebalancing algorithm for T-DCORAL. As our future work, we will propose a new ECP based on T-DCORAL to solve those drawbacks. For an optimization, we will also deeply dive all parameters used in T-DCORAL.

References

[1] M. F. Bari, *et. al.*, "Dynamic Controller Provisioning in Software Defined Networks," in Proc. *IFIP CNSM*, 2013.
[2] A. Dixit, *et. al.*, "ElastiCon: An Elastic Distributed SDN Controller," in Proc. *ACM ANCS*, 2014.
[3] Y. Chen, *et. al.*, "Towards Adaptive Elastic Distributed Software Defined Networking," in Proc. *IEEE IPCCC*, 2015.
[4] G. Prathyusha, *et. al.*, "An Efficient DHT-based Elastic SDN Controller," *IEEE COMSNET*, 2017.
[5] D. M. F. Mattos, *et. al.*, "Profiling Software Defined Networks for Dynamic Distributed Controller Provisioning," in Proc. *NoF*, 2016.
[6] L. Peterson, *et. al.*, "Cental Office Re-Architected as a Data Center," *IEEE Commun. Mag.*, vol. 54, no. 10, 96–101, 2016.
[7] S. D. Lowe, "Best Practice for Oversubscription of CPU, Memory, and Storage in vSphere Virtual Environments," *White Paper*, Dell, 2013.
[8] P. Berde, *et. al.*, "ONOS: Towards an Open, Distributed SDN OS," in Proc. *ACM HotSDN*, 2014.
[9] W. Kim, *et. al.*, "OFMon: OpenFlow Monitoring System in ONOS Controllers," in Proc. *IEEE NetSoft Conference and Workshops*, 2016.
[10] M. Al-Fares, *et. al.*, "A Scalable, Commodity Data Center Network Architecture," in Proc. *ACM SIGCOMM*, 2008.
[11] T. Benson, *et. al.*, "Network Traffic Characteristics of Data Centers in the Wild," in Proc. *ACM IMC*, 2010.
[12] A. Greenberg, *et. al.*, "VL2: A Scalable and Flexible Data Center Network," *Commun. of the ACM*, vol. 54, no. 3, 95–104, 2011.
[13] S. Kandula, *et. al.*, "The Nature of Data Center Traffic: Measurement & Analysis," in Proc. *ACM IMC*, 2009.
[14] T. Benson, *et. al.*, "Understanding Data Center Traffic Characteristics," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, 2010.