Demonstration of Geo-Distributed Data Processing and Aggregation in MEC-empowered Metro Optical Networks

Jiawei Zhang[1,*], Lu Cui[1], Zhen Liu[1], and Yuefeng Ji[1]

[1]*State Key Lab of Information Photonics and Optical Communications, Beijing Univ. of Posts and Telecommunications (BUPT), Beijing, China*
**zjw@bupt.edu.cn*

Abstract: We experimentally demonstrate a geo-distributed data processing and aggregation (GDPA) scheme in the MEC-empowered metro optical networks. The demonstration results show that the proposed scheme can improve resource utilization and reduce average job completion time.
OCIS codes: (060.4250) networks; (060.4254) Networks, combinatorial network design.

1. Introduction

Recently, multi-access edge computing (MEC) is proposed to sink the computing and storage capability from the cloud to the edge of network [1]. Data analytics service enabled by MEC is a promising application for content providers, like Facebook, Twitter, and Netflix [2]. They gather data of end users for a variety of analytical purposes such as finding the popular content of users or monitoring the QoS metrics. This service involves a large amount of data generated by geographically distributed sources (e.g., agents, sensors, etc.), which requires to be processed at local edge datacenters (EDCs), such as filtering irrelevant data from raw data, to alleviate the burden of network bandwidth. The processed data (or intermediate data) is transmitted over metro optical networks to a target EDC for aggregation (further processing). After aggregation, analysts query the target EDC for retrieving the final analyzed data. The above scenario is called geo-distributed data processing and aggregation (GDPA) in the literatures [3, 4].

The GDPA follows a "Job-Task" model [4], in which a job (e.g., a data analytics request) contains multiple tasks, and each of them conducts data processing at the local EDC. Fig. 1 illustrates the GDPA scenario. Job A has three tasks (A1, A2 and A3) locating at geo-distributed EDCs. The local EDC of task A3 is selected as the target EDC, thus task A1 and A2 should transmit their intermediate data to EDC3 for aggregation, then the final result of job A will be presented to the analysts. According to the "Job-Task" model [4], job completion time (JCT) depends on the last task arrival time at the target EDC. Because data aggregation cannot start until all the tasks are completed and transmitted to the target EDC. However, due to the different data sizes, processing power of EDCs, and available bandwidth of metro optical networks, tasks may arrive at target EDC at different time slots, which causes a waste of resources and a long average JCT under multi-job scenario. Therefore, it raises a problem, which is how to allocate processing and bandwidth resources for GDPA to reduce the average JCT and improve the resource utilization.

In this paper, we propose and experimentally demonstrate a joint computing and bandwidth allocation scheme for the GDPA in the MEC-empowered metro optical networks. Experimental results show that the proposed scheme can achieve significant average JCT reduction and high resource utilization.

2. Resource Allocation Schemes and System Implementation for GDPA

2.1 Description of Resource Allocation Schemes

A key challenge of GDPA is how much computing and bandwidth resources should be allocated for a task. Fig. 2 shows four resources allocation schemes for job A and its tasks in Fig. 1. Let t_p denote the processing time, t_c denote the transmission time and t_d denote the time difference between the first and last arrived tasks at target EDC. In Fig.

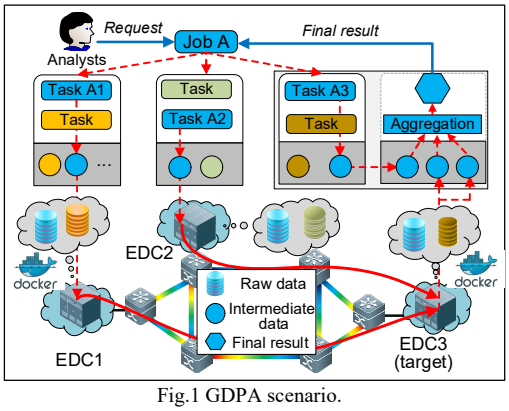

Fig.1 GDPA scenario.

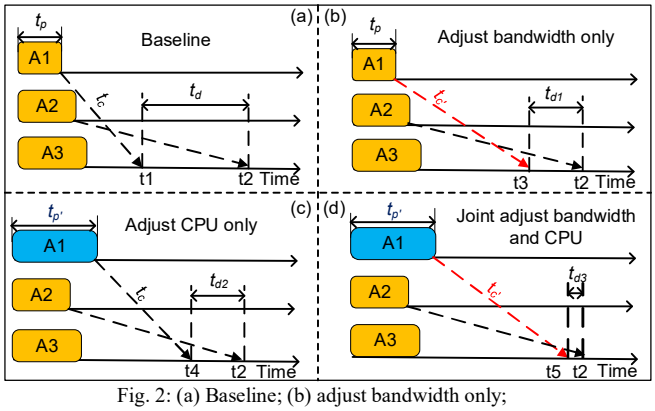

Fig. 2: (a) Baseline; (b) adjust bandwidth only; (c) adjust CPU only; (d) Jointly adjust bandwidth and CPU.

Table 1 JCBA algorithm

Algorithm: Joint computing and bandwidth adjustment (JCBA)

Input: job set ***J***, task set for each job in ***J***, CPU and network states.
Output: CPU and bandwidth allocation for ***J***.

1. Select a target EDC for each job of ***J*** according to the location of the largest-size task in the job.
2. Sort ***J*** in a descending order of T_{pre}.
3. for job ***i*** in ***J***
4. calculate the JCT of ***i*** as T_i, by allocating the maximal available resources for all the tasks in ***i*** (baseline).
5. for task ***j*** in ***i***, while ***j*** ≠ *the last arrived* task in ***i*** do:
6. get task completion time $T_{i,j}$, adjust CPU and bandwidth for task ***j*** to ***min*** $(T_i - T_{i,j}) \geq 0$
7. end for
8. if there is not enough resources for job ***i***, keep ***i*** in ***J*** for waiting free resources. Updates CPU and bandwidth states
9. end for

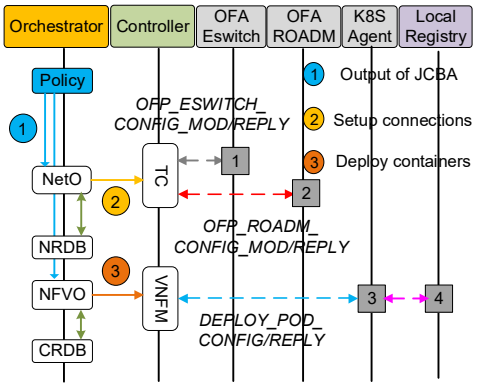

Fig. 3 Interaction of GDPA

2(a), the baseline allocates the maximal available computing (e.g., CPU) and bandwidth resources for the tasks. It may cause a large t_d, due to the difference of maximal available resources for the tasks. For example, task A1 has to wait a long time until task A2 arrives at EDC3. This will not only waste a large storage space of EDC3, but also occupy more CPU and bandwidth resources when processing and transmitting task A1. To overcome this issue, Fig. 2(b-c) are to adjust bandwidth and CPU capacity for the tasks respectively to extend the transmission ($t_{c'}$) and processing ($t_{p'}$) time. The two schemes not only shorten t_d, but also release some CPU and bandwidth resources that could be used by other jobs. In some extend, adjusting one resource may not achieve the best optimization effort, thus a joint computing and bandwidth adjustment (JCBA) scheme is designed to further improve the performance in Fig. 2(d). The detail of JCBA scheme is shown in Table 1. First, we select a target EDC for each job according to the location of the largest-size task in the job. Second, we sort the jobs in a descending order of pre-JCT T_{pre}. T_{pre} is calculated when all the jobs are allocated to the same CPU and bandwidth resources. Note that, in this stage, we do not actually accommodate jobs yet. Third, we accommodate job i by calculating its JCT T_i according to the baseline, whose routing is shortest path. We suppose that T_i is the time slot of the last arrived task plus the aggregation time. With T_i unchanged, we jointly adjust CPU and bandwidth for the tasks whose arrival time is ahead of the last arrived task of job i, while making the completion time of task j ($T_{i,j}$) smaller than T_i. We allocate resources for task j according to the minimal value of T_i - $T_{i,j}$. To guarantee the minimal JCT, we do not adjust the resources of the last arrived task of a job. The states of CPU and bandwidth resources will update once a job is accommodated.

2.2 System Implementation of GDPA

The proposed schemes are demonstrated on a converged edge access network platform (CEANP), which is an SDN/NFV enabled testbed that converges radio, optical and EDC networks [5]. There are three planes—application plane, control plane and data plane, which are interconnected via a northbound interface (NBI) and a southbound interface (SBI). In the data plane, each network node attaches with an OpenFlow-agent (OFA) that communicates with the controller through an extended OpenFlow protocol (OFP). The EDC node is a docker-container based virtual system, which is controlled and managed by Kubernetes (k8s). A container with computing and storage capabilities is assigned for a task to perform data processing and aggregation. The local docker repository provides ready-to-use docker images for EDCs to increase the speed of docker deployment. In the control plane, an orchestrator, including network orchestrator (NetO) and NFV orchestrator (NFVO), is to coordinate abstracted computing and bandwidth resources that are stored in the container resource database (CRDB) and network resource database (NRDB) respectively. Controller consists of transport controller (TC) and VNF manager (VNFM), which are responsible for connection setup and container deployment. Fig. 3 details the interaction between the modules.

3. Experimental Setup and Results

A 12-node experimental topology is shown in Fig. 4(a). There are three real docker-container based EDC nodes (red circle marked) with dual-core Intel CPU working at 2.10 GHz. The EDCs are interconnected by an Ethernet-over-DWDM metro optical network. Each fiber link has six wavelengths working at 6x10 Gbps. We setup real WSS-based ROADMs, wavelength tunable transponder cards (WTTC), and Ethernet switches, which are commercial devises or prototypes programmed via an extended OpenFlow protocol. Besides the real nodes, others are emulated by OvS-agents (OpenvSwitch) through virtual machines. To emulated the data processing, we use Linux command to generate a number of files with random sizes in [1, 10] GB, and each file conducts MD5 algorithm to emulate task processing at local EDCs. The processing time depends on the allocated CPU resource for a container. The CPU capacity is given in mili-cpus and can be adjusted by k8s. For data transmission, we use Socket to control the port

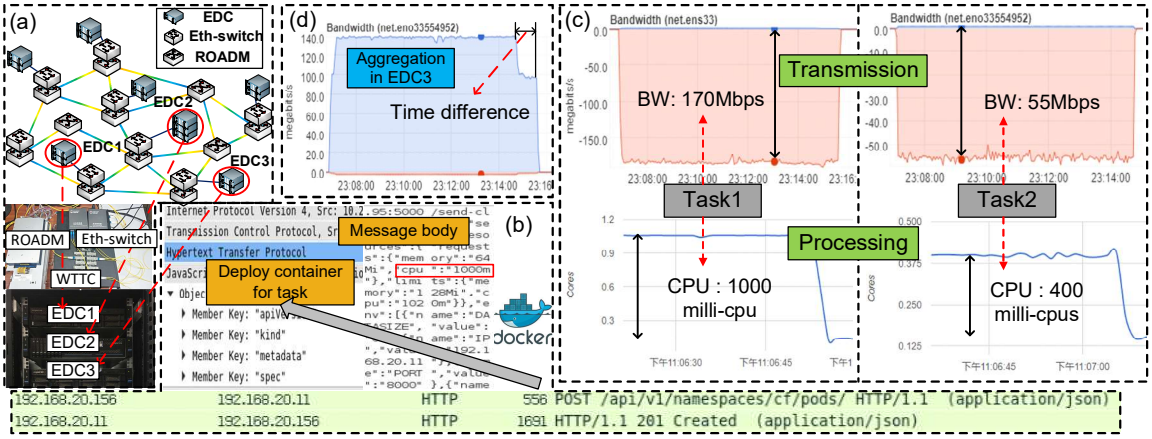

Fig. 4 (a) Experimental topology; (b) Wireshark capture of container deployment by k8s; (c) allocated bandwidth and CPU resources for task 1 and task 2 by JCBA scheme; (d) time difference of job.

rate of a TCP connection to emulate the allocated bandwidth for a task. The physical port rate of an EDC is 1 Gbps in the experiment. Table 2 shows the testing data of transmission time and processing time under different data sizes with different allocated bandwidth and CPU resources, which provides a reference for the demonstration.

We experiment the GDPA scenario among the real nodes, where task 1 and task 2 are locally processed in EDC1 and EDC2, and their intermediate data is transmitted to EDC3 for aggregation. Fig. 4(b) shows the Wireshark captures of container deployment in the three real EDCs through k8s system. With the k8s, we can deploy container with different CPU capacities. Fig. 4(c) shows the allocated bandwidth and CPU resources for task 1 and task 2 by JCBA scheme, which are visualized through Netdata and cAdvisor (open source software) respectively. In addition, with the help of JCBA, we can achieve a shorter time difference between task 1 (first arrived task) and task 2 (last arrived task), which is shown in Fig. 4(d).

In addition, the four schemes in Fig. 2 are implemented in our testbed to compare their performance. Each point in Fig. 5 is averaged over five experimental results. As shown in Fig. 5(a), we observe that the average JCT increases with the growing number of jobs. The reason is that more jobs would compete network resources, which causes more queuing delay. The JCBA is a straightforward but efficient algorithm that achieves the lowest average JCT. This is because it releases the resources of earlier arrived tasks, which can be used by other jobs to reduce their queuing delay. Fig. 5(b) shows the time difference between the first and last arrived tasks of a job. The JCBA achieves the smallest time difference, because it jointly adjusts the CPU and bandwidth resources to extend the arrival time of the tasks to realize the minimal t_d.

4. Conclusion

We proposed and experimentally demonstrated a joint computing and bandwidth allocation scheme for geo-distributed data processing and aggregation in the MEC-empowered metro optical networks. We verified its feasibility for improving the service performance, such as average JCT.

Acknowledgment This work was supported by the National Key R&D Program of China (No. 2018YFB1800802), the National Nature Science Foundation of China Projects (No. 61971055), the Beijing Natural Science Foundation (No. 4192039), the fund of State Key Laboratory of Information Photonics and Optical Communications, China, IPOC2019ZT05.

Reference

[1] T. Taleb, et al., *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657-1681, 2017.
[2] A. Vulimiri, et al., *in Proc. ACM SIGMOD*, pp. 1087–1092, Melbourne, Australia, 2015.
[3] Z. Hu, et al., *in Proc. IEEE INFOCOM*, pp. 1-9, San Francisco, USA, 2016.
[4] Z. Liu, et al., *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, no. 7, pp. B152-B163, 2018.
[5] S. Zhang, et al., *in Proc. ECOC*, Dublin, Ireland, 2019.

Table 2 Testing data of processing and transmission time

Data size (GB)	Transmission time (s)				Processing time (s)			
	800 Mbps	400 Mbps	200 Mbps	100 Mbps	1000 mili-cpu	750 mili-cpu	500 mili-cpu	200 mili-cpu
1	11.2	22.004	45.001	88.124	8.939	11.607	17.334	44.781
3	34.831	67.576	129.395	258.248	27.814	35.298	47.446	139.351
5	57.187	119.942	215.371	441.043	46.684	59.027	78.001	233.884
7	80.246	166.589	312.722	614.099	65.725	83.472	106.701	329.283
9	106.821	204.458	392.039	802.928	85.253	106.701	137.923	427.122

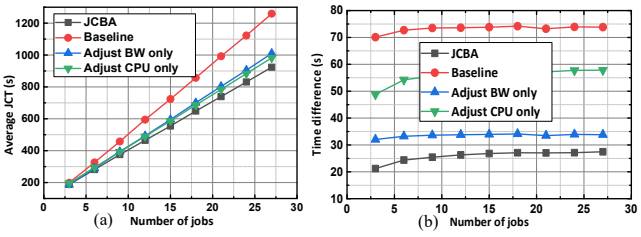

Fig. 5: (a) Average JCT; (b) time difference (t_d).