mathematics

Article

An Optimization Method of Large-Scale Video Stream Concurrent Transmission for Edge Computing

Haitao Liu [1,2], Qingkui Chen [1,3,*] and Puchen Liu [4]

[1] Business School, University of Shanghai for Science and Technology, Shanghai 200093, China; liuhaitao@lyu.edu.cn
[2] Office of Information, Linyi University, Linyi 276002, China
[3] School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China
[4] Department of Applied Statistics, Shanghai Polytechnic University, Shanghai 201209, China; pcliu@sspu.edu.cn
* Correspondence: chenqingkui@usst.edu.cn; Tel.: +86-13122381881

Abstract: Concurrent access to large-scale video data streams in edge computing is an important application scenario that currently faces a high cost of network access equipment and high data packet loss rate. To solve this problem, a low-cost link aggregation video stream data concurrent transmission method is proposed. Data Plane Development Kit (DPDK) technology supports the concurrent receiving and forwarding function of multiple Network Interface Cards (NICs). The Q-learning data stream scheduling model is proposed to solve the load scheduling of multiple queues of multiple NICs. The Central Processing Unit (CPU) transmission processing unit was dynamically selected by data stream classification, as well as a reward function, to achieve the dynamic load balancing of data stream transmission. The experiments conducted demonstrate that this method expands the bandwidth by 3.6 times over the benchmark scheme for a single network port, and reduces the average CPU load ratio by 18%. Compared to the UDP and DPDK schemes, it lowers the average system latency by 21%, reduces the data transmission packet loss rate by 0.48%, and improves the overall system transmission throughput. This transmission optimization scheme can be applied in data centers and edge computing clusters to improve the communication performance of big data processing.

Keywords: edge computing; multi-network port parallelism; link aggregation; Q-learning; load balancing; DPDK

MSC: 68M07; 68M20; 68Q32; 68T05

check for updates

Citation: Liu, H.; Chen, Q.; Liu, P. An Optimization Method of Large-Scale Video Stream Concurrent Transmission for Edge Computing. *Mathematics* **2023**, *11*, 2622. https://doi.org/10.3390/math11122622

Academic Editors: Ming Yang and Liqun Shan

Received: 2 May 2023
Revised: 30 May 2023
Accepted: 6 June 2023
Published: 8 June 2023

1. Introduction

In recent years, the video content business has grown rapidly both in China and overseas. International Data Corporation (IDC) reports that the video cloud market in China reached USD 5.04 billion in the second half of 2021 and is expected to reach USD 31.4 billion by 2025. According to Cisco's forecast, video streaming will account for 80% of the total Internet traffic in 2023 [1], and the proportion of video streams in the total Internet traffic has witnessed dramatic increases. Following the popularity and development of smart education, the cloud recording and live streaming system of university campuses has become a new modern teaching solution. The campus cloud recording and live streaming system is based on the Internet of Things (IoT) technology and uses high-speed transmission and edge network storage technology to achieve video acquisition and processing and usage. There is a high demand for real-time computing power and network transmission. Edge computing is an important solution for meeting the needs of video services. It provides a lower time delay network access capability, more optimized network bandwidth cost, rich heterogeneous arithmetic resources, and intelligent scheduling through high-quality edge nodes. Furthermore, it features a globally distributed network bandwidth and a variety of heterogeneous arithmetic resources to meet the demand of extreme High Definition (HD), real-time interaction, and an immersive experience of video services [2]. In video live streaming, recording, and on-demand scenes, the large-scale, highly concurrent, and continuous audio and video data are required to be transmitted quickly and frequently between the edge cameras, edge servers, network switches, and storage devices [3]. Due to the substantial scale increase in service data at the edge nodes, it is difficult for a single node in edge computing to complete the computing work, so the edge nodes need to be extended into edge clusters. With the sudden increase in the amount of data communicated within the edge cluster, a large amount of network traffic bandwidth is required to receive and forward video streams, which puts higher demands on the communication bandwidth within the edge cluster.

The performance optimization of large-scale and highly concurrent video stream accessing and forwarding in edge computing clusters has become an important challenge. The video service has large-scale access devices that receive highly concurrent video streams and forward them to user devices. When there is a bottleneck in the transmission performance of high-concurrency video streams, the computing overload of the service cluster, as well as the delay, freezes, or there is a frame loss of the application layer video stream, or even an inability to open the video, which seriously affects the user experience [4]. Therefore, the transmission performance of the network outlet, intermediate equipment, and internal network of the live broadcast recording system seriously affects the user's viewing experience.

The existing concurrent access technology solution faces challenges such as a high cost of access equipment and high packet loss rate of video transmission. In traditional technology, higher-speed equipment is usually used to increase the bandwidth of edge clusters, but the equipment costs are high, the construction period is long, and the system scalability is poor. Link aggregation technology is a technical solution to this problem, which bundles multiple physical interfaces into one logical interface on the basis of the original hardware to increase the link bandwidth. As the operating system is not friendly enough to support this technology, the development complexity is high.

Compared with previous works, the main contributions of this article are as follows:

(1) In this work, an edge data stream access forwarding scheme based on DPDK and link aggregation technology is proposed to satisfy the need of communication performance within edge clusters. This scheme is significant for the large-scale data-intensive computing job from the artificial intelligent applications.

(2) The scheme makes full use of DPDK's parallel packet processing support for multi-core and multi-NICs. It adopts a Q-learning data stream scheduling that not only enables the bandwidth overlap of multiple NICs but also adapts to the high speed, high concurrency, high traffic and strong real-time characteristics of communication systems to obtain a higher throughput at a lower cost [5–8].

The framework of this paper is structured as follows: Section 2 is the research foundation; Section 3 describes related work; Section 4 is a formal description of the issue; Section 5 details the design of the edge concurrent data flow pickup system; Section 6 explains the data flow relay on scheduling model based on Q-learning; Section 7 describes the solution for performance optimization; Section 8 is the experimental design and analysis; Section 9 is the conclusions.

2. Research Background

2.1. Streaming Media and Related Concepts

Streaming media is a form of media that delivers audio, video and multimedia files in a stream over the network. Streaming media compress the continuous audio and video information and store it on a network server, where users download the data and watch the video simultaneously. This is achieved by the key technology of stream transmission, which converts multimedia files such as animation, audio, and video into a sequence of compressed packets by a specific compression algorithm and transmits them in real time from the video server to the user. There are two main types of streaming services on the Internet: sequential streaming and real-time stream transmission.

The data transmitted by streaming media are stream data, which are a continuous stream of audio or video data transmitted and played in time sequence. Each stream is encapsulated into a packet sequence containing a timestamp header. When the packet sequence is received by the user, the audio or video is played back in real time based on the timestamp information.

Live broadcast is a broadcast mode in which the post-synthesis and broadcast of the program are carried out simultaneously. As Internet technology development progresses, there is an increased application of live web video broadcasting, and it is particularly popular in smart education application scenarios. In teaching broadcasts, teachers and students interact and learn online. In order to improve the educational effect of the online classroom, a high timeliness and video completeness are required. Broadcasting on demand is a way to play programs according to the viewers' requests. In the on-demand service, the pre-recorded multimedia programs are transmitted to the requesting user according to the user's choice, which requires a low timeliness and quality of data transmission.

2.2. Edge Live Broadcast and Recorded Broadcast Cluster

The edge recorded broadcast and live broadcast system is a localized distributed deployment solution for realizing live video broadcast and transcoding services. The system provides load balancing, web application services, transcoding services, push stream services, database services, caching services, etc., in the case of a large number of recorded and live broadcast needs, which ensures stability and scalability. The overall architecture is shown in Figure 1.

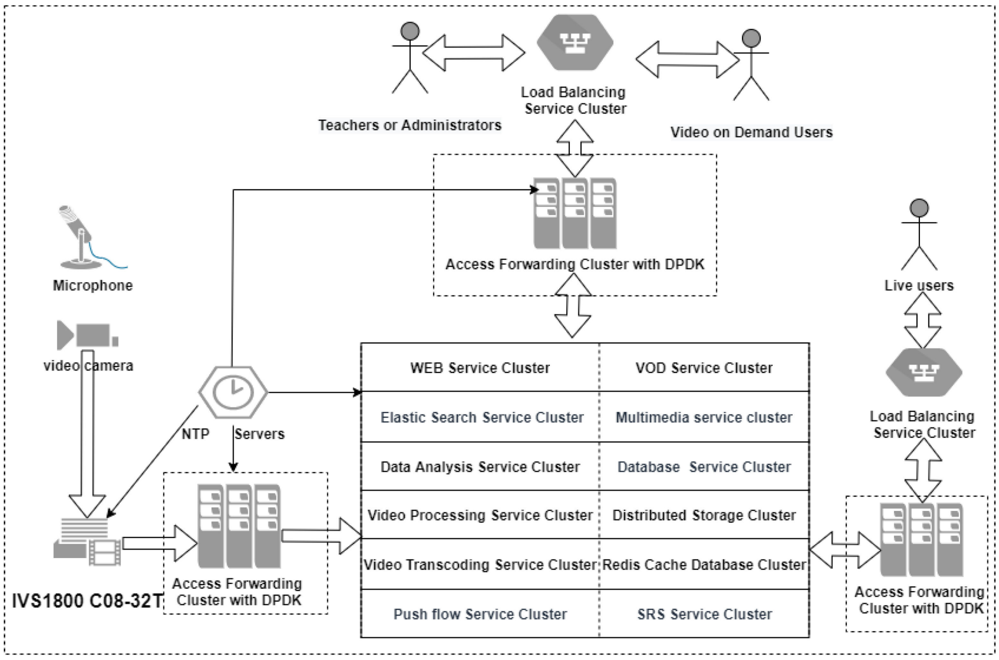

Teachers or Administrators
Load Balancing Service Cluster
Video on Demand Users
Access Forwarding Cluster with DPDK
Microphone
video camera
Live users
WEB Service Cluster
VOD Service Cluster
Elastic Search Service Cluster
Multimedia service cluster
Data Analysis Service Cluster
Database Service Cluster
Video Processing Service Cluster
Distributed Storage Cluster
Video Transcoding Service Cluster
Redis Cache Database Cluster
Push flow Service Cluster
SRS Service Cluster
NTP Servers
Load Balancing Service Cluster
Access Forwarding Cluster with DPDK
IVS1800 C08-32T
Access Forwarding Cluster with DPDK

Figure 1. Edge live broadcast and recorded broadcast (video on-demand) cluster architecture.

The system consists of acquisition equipment (microphone and video camera to realize the acquisition of sound and video signals in smart classrooms), encoding equipment (IVS1800 C08-32T: cloud video recorder with encoding, staging, and intelligent analysis), an access and forwarding cluster with DPDK acceleration (realizing the receiving and forwarding of highly concurrent multiple audio and video streams), a push flow service cluster (pushes content to the connected servers or clients), a live service cluster (SRS Live service cluster: live streaming video in real time), a distributed storage cluster (stores audio and video data for on-demand), a Redis cache database cluster (realizes the data caching function), a multimedia service cluster (processes audio and video data and completes interaction work on the user side), a data analysis service cluster (mainly for collecting user data), a database service cluster (stores business data of all business systems), an elastic search service cluster (also known as an elastic distributed analysis search service cluster, and is a distributed storage system that can store user behavior data on multiple nodes, which, in turn, is a highly scalable search and data analysis engine for statistical analysis and other functions of the direct recording system), a video transcoding service cluster (plays the corresponding content of the corresponding source video requested by users), a web service cluster (provides users with all web business api services), a video transcoding service cluster (transcodes recorded videos or uploaded videos), a load balancing service cluster (allows users' access to share the load to different physical servers through some control policies), and a global time synchronization service cluster (network time protocol servers ensure the whole system's time synchronization). The video live broadcast and recorded broadcast scenario usually includes core business modules such as audio and video capture, pushing flow, transcoding, storage, distribution, and playback, as well as interactive business logic such as Q&A, which supports 200 edge recorded broadcast and live broadcast classrooms and supports 5000 concurrent logins and online viewing. The user experience has high requirements for clarity, playback fluency, playback delay, etc. The edge recorded broadcast and live broadcast system has a large number of small packet transmissions of video streams, and the traditional way is to use interrupt scheduling NIC transmission, which seriously affects the transmission performance. The work in this paper focuses on accessing and forwarding clusters for the data streams in Figure 1 and using DPDK technology to accelerate the transmission of concurrent video streams.

3. Related Work

The optimization of video transmission efficiency has become an important challenge in networks. Researchers have proposed numerous transmission architecture optimization algorithms in order to improve network transmission efficiency.

Hu designed a video transmission optimization strategy that takes reinforcement learning in the edge computing platform to improve the video transmission efficiency and quality. However, compression and decoding in video transmission optimization are not analyzed. Thus, there is no direct reduction in redundant traffic in video transmission [9]. Many researchers have introduced DPDK into video delivery optimization to ensure that all video clips have the same size and to speed up video delivery by bypassing the operation system kernel. Qi introduced DPDK to ensure that all video segments with the same size may accelerate video transmission by bypassing the operation system kernel. In spite of this, a redundant delivery, high network load, and long delivery delay need to be optimized [10]. Gao proposed a cross-modal streaming media delivery architecture driven by edge-smart EM that leverages communication, caching, computation, and control capabilities. However, the architecture is not applied to large-scale and complex cross-modal multimedia applications [11]. Tianyu Xu proposed a new DDS middleware called 3DS by taking advantages of the user-space I/O stack of DPDK to improve the communication performance of DDS middle wares. However, the proposed 3DS has not been implemented on a real system [12]. Tashtarian proposed ROPL, a learning-based edge client request management solution that leverages recent breakthroughs in deep reinforcement learning. This work serves requests from concurrent users joining various HTTP-based live video channels. However, scalability is not achieved by leveraging a distributed RL approach [13]. Park compared and analyzed network layer high-speed packet processing techniques in Linux, such as DPDK and XDP. He showed experimentally that DPDK outperforms XDP for small-length packets, but streaming services use small-length packets to control information. It was finally concluded that DPDK and XDP can be applied to the data transmission of immersive streaming services [14]. Zeng proposed Middle

Net, a unified L2/L3 NFV and L4/L7 middle box framework. Middle Net uses DPDK for high-performance, zero-copy packet delivery in L2/L3 NFV [15]. The sleep-based approach proposed by Wu not only reduces the high not-used power waste and high CPU usage caused by DPDK's empty polling but also reduces the negative impact of DPDK's high CPU usage on other services [16].

4. Formalized Description of the Problem

In the cloud recorded live broadcast system, there are a large number of on-demand and live video streams that need to be transmitted in real time, and the two types of video streams compete for network bandwidth and cause network congestion. The live video streams are transmitted in a transport stream (ts) slice, and each ts packet size is 188 bytes, which is a small-scale data stream of continuous transmission, suitable for concurrent transmission with DPDK's multi-port multi-queue User Datagram Protocol (UDP) protocol; the recorded (on-demand) video streams are large-scale data streams, and usually video files of up to 1 G need to complete the transmission work in one time, suitable for concurrent transmission with DPDK's multi-port link aggregation multi-queue. This paper aims to realize classified and accelerated transmission according to the transmission characteristics of different video streams and solve the system data transmission performance bottleneck, as well as improve the system throughput.

4.1. Formal Description

Set the video stream $S_i = \{S_1, S_2, \ldots, S_n\}$ to be the concurrent video streams arriving at the edge computing cluster, where the live video stream is $\overline{S_j} = \left\{\overline{S_1}, \overline{S_2}, \ldots, \overline{S_k}\right\}$ and the on-demand video stream is $\hat{S_k} = \left\{\hat{S_1}, \hat{S_2}, \ldots, \hat{S_m}\right\}$. The receiving rate of the streaming data stream is denoted as V_r. Assume that the size of the video streams arriving in Δ_t time is $|S_i|_{\triangle t} = \left|\overline{S_j}\right|_{\triangle t} + \left|\hat{S_k}\right|_{\triangle t}$, which is the sum of the on-demand video stream and the live video stream; then, the receiving rate of the receiving system data stream V_r is: $V_r = \frac{|S_i|_{\triangle t}}{B}$ (B is the effective bandwidth of the network).

Then:

$$V_r = \lim_{\triangle t \to 0} \frac{\left|\overline{S_j}\right|_{\triangle t} + \left|\hat{S_k}\right|_{\triangle t}}{B} \tag{1}$$

As shown in Equation (1), the network packet loss rate tends to be zero when the system receiving rate is infinitely close to V_r.

When video stream S_i is transmitted in the network, it can be seen as a UDP five-tuple stream sequence (containing a 16-bit source port number, 16-bit destination port number, 16-bit UDP length, 16-bit UDP checksum, and data sequence) of data stream data.

4.2. Symbol Descriptions

The basic symbols used in this article are described in Table 1.

$$\omega = \frac{1}{|\mathrm{UN}|} \sum_{\mathrm{unity} \in \mathrm{UN}} \left| \frac{\mathrm{load}_{\mathrm{ij}}^{\mathrm{used}}}{\mathrm{load}_{\mathrm{ij}}^{\mathrm{total}}} - \mathrm{avg\_load}_{cur} \right| \tag{2}$$

$$\mathrm{avg\_load}_{\mathrm{cur}} = \frac{1}{|\mathrm{UN}|} \sum_{\mathrm{unity} \in \mathrm{UN}} \left| \frac{\mathrm{load}_{\mathrm{ij}}^{\mathrm{used}}}{\mathrm{load}_{\mathrm{ij}}^{\mathrm{total}}} \right| \tag{3}$$

Table 1. Symbol description.

Symbol	Description
UN	set of transfer processing units data
unit_{ij}	the transfer CPU processing thread core corresponding to the jth interface of the ith NIC
data	the set of data blocks to be transferred for processing
data_{k}	the load amount demanded for the kth data block
$\text{load}_{\text{ij}}^{\text{old}}$	the original load of unit_{ij} before scheduling starts
$\text{load}_{\text{ij}}^{\text{used}}$	the existing load of unitij at a certain point in the scheduling process
$\text{load}_{\text{ij}}^{\text{total}}$	the total load capacity of unit_{ij}
$\text{avg\_load}_{\text{cur}}$	the average load utilization rate of all units in the current round
$\chi_{\text{k}}^{\text{ij}}$	0–1 variables, set to 1 if data_{k} are assigned to unit_{ij} processing; otherwise, set to 0
ω	indicates the deviation of the load utilization of unit_{ij} in the current round relative to the average load utilization of all transfer processing units

Based on Equations (2) and (3), the problem is formulated as follow:
Goals:

$$\min \omega \tag{4}$$

Constraints:

$$\frac{\text{data}_{\text{k}} \cdot \chi_{\text{k}}^{\text{ij}} + \text{load}_{\text{ij}}^{\text{used}}}{\text{load}_{\text{ij}}^{\text{total}}} \leq \theta \quad (\theta \text{ is the upper limit of the setting}) \tag{5}$$

$$\sum_{\text{unity}_{\text{ij}} \in \text{UN}} \chi_{\text{k}}^{\text{ij}} \geq 1 \tag{6}$$

Constraint 1 indicates that, after assigning data_{k} to a transfer processing unit unit_{ij}, the load utilization of that processing unit cannot be higher than the specified upper limit θ.

Constraint 2 indicates that at least one available processing unit needs to be allocated for each data_{k}.

5. System Model

5.1. System Framework Design

The system model mainly consists of a Q-learning scheduler, DPDK multi-network card multi-port transmission module, memory pool and multi-loop queue for receiving and forwarding, etc. The overall architecture is shown in Figure 2. The detailed design scheme is as follows.

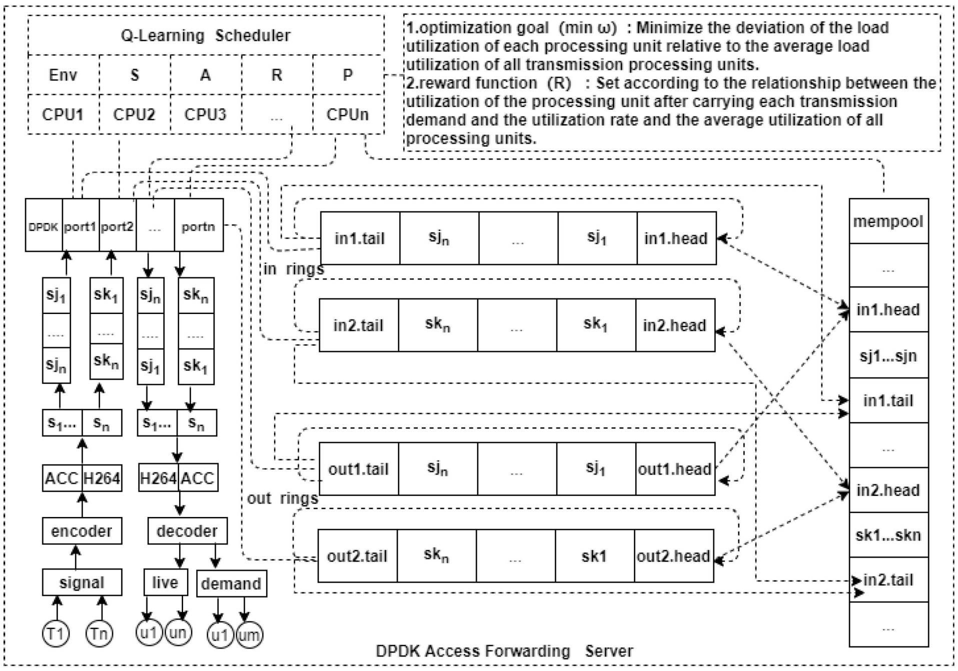
(1) Q-learning scheduler

It realizes the allocation and binding of CPU cores and dynamic adjustment functions. The other cores of the CPU are bound to the corresponding multi-network port multi-queue to complete data sending and receiving processing [17]. Five elements are first determined in the Q-learning algorithm: environment (Env), state space (S), action space (A), reward value function (R), and policy or probability of action selection (P). In each round of load distribution, the selection operation for the current data block transfer processing unit is considered as an Env state, and the selection of all available processing units is considered as an optional action for the current state. Each available processing unit is assigned a reward value R. The algorithm selects the processing unit with the largest R value or randomly selects a processing unit to reach the next state. Furthermore, the Q-learning-based load allocation algorithm is described in the next section. We illustrate how to select an available processing unit across different states after these five elements are determined.

(2) DPDK multi-NIC multi-port transmission module

Network interface manager: this manages the whole system configuration information, such as the NIC port type and number information, as well as the port distribution and binding of multiple network cards. DPDK data flow splitter: this realizes the classification of access data according to data flow characteristics. Data flow aggregator: this completes data aggregation according to data flow characteristics and then carries out parallel transmission.

(3) Memory pool and multiple cyclic queues for receiving and forwarding (Mempool and I/O rings): these are responsible for managing the memory pool in the system, managing the cyclic multiple queues on the receiving and forwarding sides, the number of buffers, the table of forwarding core–buffer relationships, and the control for data flow overflow situations [18,19].



Q-Learning Scheduler
Env S A R P
CPU1 CPU2 CPU3 ... CPUn
1.optimization goal (min ω) : Minimize the deviation of the load utilization of each processing unit relative to the average load utilization of all transmission processing units.
2.reward function (R) : Set according to the relationship between the utilization of the processing unit after carrying each transmission demand and the utilization rate and the average utilization of all processing units.
DPDK port1 port2 ... portn
in rings
in1.tail sjn ... sj1 in1.head
in2.tail skn ... sk1 in2.head
out rings
out1.tail sjn ... sj1 out1.head
out2.tail skn ... sk1 out2.head
mempool ... in1.head sj1...sjn in1.tail ... in2.head sk1...skn in2.tail ...
sj1 .... sjn
sk1 .... skn
sjn sj1
skn sk1
s1... sn
ACC H264
H264 ACC
encoder
decoder
signal
live
demand
T1 Tn
u1 un
u1 um
DPDK Access Forwarding Server

Figure 2. DPDK multi-network port multi-cycle queue data stream access forwarding system model.

5.2. System Working Principle

The working mechanism of the system is as follows: the scheduling controller manages the CPU scheduling, memory and data flow overflow, resource scheduling, data flow distribution, and network interface from a system-wide perspective. It collects the status and feedback information of various management modules in the system and finally implements scheduling control. When the highly concurrent data packets arrive, the data stream distributor on the receiving side carries out classification work, divides the audio-video data streams into two categories—small streams and large streams—based on the data header characteristics, and then distributes the corresponding access data streams to each receiving port; the CPU manager is responsible for binding the CPU cores to the corresponding receiving ports and forwarding ports; the memory and overflow manager gives the receiving ports and forwarding ports a shareable queue and then uses the Q-learning data stream receiving scheduling model after passing the stream classification in order to achieve the binding of multiple queues and CPU cores; the network interface manager completes the task of audio–video stream assignment, forwards the forwarding port number of the CPU core by querying the network interface configuration table, and sends the data to the corresponding receiving port to enter the circular queue on the receiving side, respectively. When the scheduler controller detects data reception, it schedules the corresponding port on the forwarding side to forward it in time. When there is an overflow situation that exceeds time requirements to be received, the overflow data stream is put into the buffer. The scheduler system maintains multiple circular audio and video stream buffers, and the number of buffers is dynamically adjusted with the change in data streams. The massive amount of highly concurrent video streams are balanced and loaded onto CPU cores and eventually forwarded to other servers for storage computation processing, etc. [20].

6. The Q-Learning-Based Scheduling for Video Stream Transmission

Although DPDK works in the user state, the scheduling of threads still relies on the kernel, and the kernel thread switching can easily lead to performance overhead caused by cache misses and write backs. If bi-directional information on the same stream is processed on different cores, data synchronization problems between multiple cores occur. Therefore, the specific tasks are bound to specific cores for processing by using the affinity binding of CPUs of threads, which can reduce the frequent switching between different cores. The symmetric Receive Side Scaling (RSS) [21] technology for the DPDK platform effectively processes the opposite data packets of the same flow on different cores, but when there is a hash collision or a large-scale flow of the same connection, there will be load imbalance. In summary, it can be seen that static hashing techniques cannot meet the performance requirements for the concurrent processing of data messages, and usually deteriorates the load on a single core, reducing the processing efficiency of multicore CPU [22].

The goal of this section is to design a solution for receiving and scheduling highly concurrent data streams in edge computing clusters, especially for burst data streams. The widespread burst streams bring more challenges to the multi-queue's and CPU's core allocations [23,24]. The DPDK classification technique is used to divide the video streams into small streams for live broadcast and large streams for recorded broadcast and on-demand broadcast. The balanced load capacity of the system is quantified by the balanced CPU load degree, and Q-learning is used to allocate the queues and CPU cores. Different computing cores operate different queues to avoid the overhead of locks caused by the simultaneous access to a queue by multiple threads. If the number of logical cores is larger than the number of sending queues contained on each interface, then the mapping of queues to compute cores needs to be implemented efficiently. First, one CPU core (called the "master" core) is used exclusively to control and manage the server, whereas the other CPU cores are used to process packets. N denotes the number of CPU cores dedicated to packet processing, i.e., excluding the master core. Secondly, the DPDK distributes the load from each port evenly over the N CPU cores, and each CPU computing core is responsible for processing the packets from each port. Modern NICs perform load balancing by allowing each port to assign incoming packets to N separate logical queues (called receive (Rx) queues) [6]. Each Rx queue is assigned to a CPU core, and each CPU core processes as many Rx queues as the total number of NIC ports. The Rx queue size, denoted by K, is the maximum number of packets queued at the same time. When a CPU core has finished processing a packet, the packet is forwarded from its Rx queue to the transport (Tx) queue associated with the output port. At this stage, the packet waits to be transmitted on the next link and does not require any further CPU core processing resources. Figure 3 illustrates the mapping between CPU cores, ports, and Rx queues.

6.1. Model Description

The implementation of the DPDK hierarchical scheduler block is discussed here to explore the causes of performance degradation. Firstly, the DPDK scheduler ensures that the complexity of queue operation thread safety is high, especially in terms of multi-core scaling requirements, where spin locks severely affect the performance; secondly, the shared data structure leads to a significantly reduced cache hit rate [25]. All these make the multicore collaboration of the DPDK scheduler less efficient. To fit the workload for scheduling jitter-sensitive video traffic, an efficient strategy for allocating CPUs and multiple queues with Q-learning's load balancing and constrained buffer management is utilized.

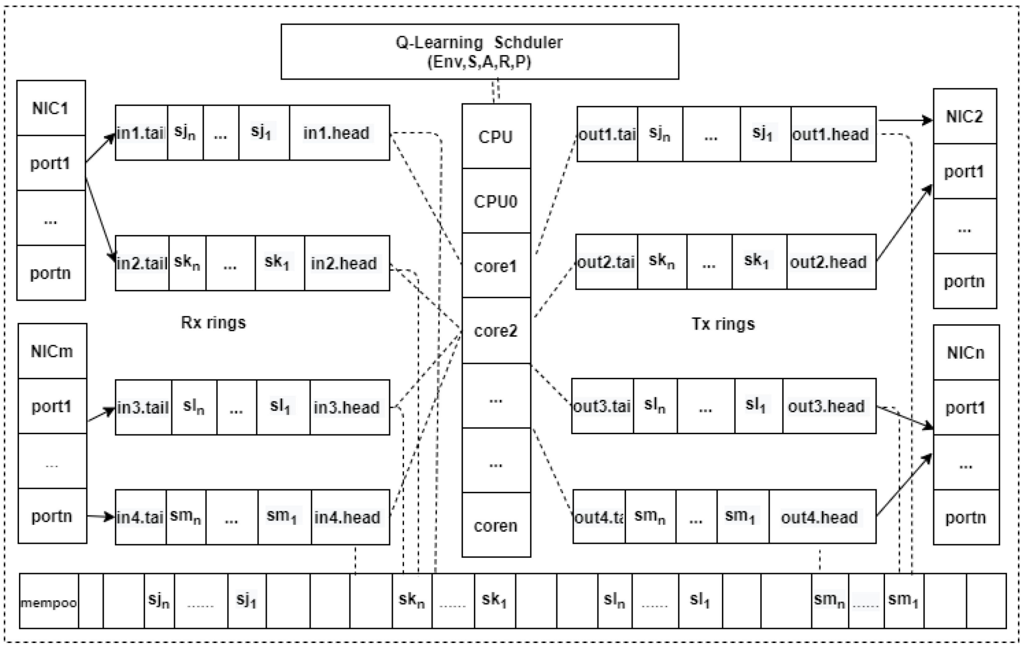

Q-Learning Schduler
(Env,S,A,R,P)
NIC1
port1
...
portn
NICm
port1
...
portn
in1.tail sj_n ... sj_1 in1.head
in2.tail sk_n ... sk_1 in2.head
Rx rings
in3.tail sl_n ... sl_1 in3.head
in4.tai sm_n ... sm_1 in4.head
CPU
CPU0
core1
core2
...
...
coren
out1.tai sj_n ... sj_1 out1.head
out2.tai sk_n ... sk_1 out2.head
Tx rings
out3.tai sl_n ... sl_1 out3.head
out4.ta sm_n ... sm_1 out4.head
NIC2
port1
...
portn
NICn
port1
...
portn
mempoo sj_n ...... sj_1 sk_n ...... sk_1 sl_n ...... sl_1 sm_n ...... sm_1

Figure 3. Q-learning scheduler architecture with multiple NICs, multiple queues, and multiple CPU cores.

6.2. Scheduling Algorithm Design

At any given moment, there are often several concurrent transfer demands of data blocks. The goal of this algorithm is to reasonably allocate the concurrent transfer requirements to some processing units of the server and to achieve load balancing as much as possible. As the concurrent data streams to be transferred and processed in the video live broadcast and recorded broadcast application scenarios are close to PB in magnitude, the number of queues generated can reach several hundred. Therefore, how to reasonably allocate the transfer load generated by a large amount of concurrent data to numerous queues is an important issue. Large-scale clusters are usually configured with dozens or even hundreds of CPU cores to concurrently process multi-queue packets; however, the communication overhead between CPU cores reaches an unacceptable level for users [4]. Therefore, it is difficult to approximate the optimal solution in polynomial time using simple heuristic algorithms. However, reinforcement learning is suitable for solving optimization and scheduling problems with unknown models, and has a wide range of applications in optimization featuring a high complexity and large scale [26–30]. Therefore, the reinforcement-learning-based Q-learning method was selected to solve the data stream receiving scheduling problem. Q-learning is a model-free learning method that allocates the load according to the reward value magnitude. It is particularly important to design the reward value function to find the appropriate processing unit for all the data blocks, as well as to give preference to the less loaded processing units. For each data block's transfer requirement, a Q-value matrix is learned by Q-learning. An available processing unit is selected based on the Q matrix and recorded in the processing unit selection set. When all the transfer demands are processed, the load allocation scheme is obtained.

6.2.1. Algorithm Principle

The number of processing units and the load are promptly updated by switching time slices according to the number of live and on-demand viewers. Multiple iterations are set for finding the optimal solution, and, in each iteration, the Q-learning-based load allocation algorithm is executed as Table 2.

Table 2. The Q-learning-based load allocation algorithm.

1: Initialize the load allocation array LOAD_DIST and read in the load demands $\text{load}_k \in$ LOAD for each data block transferred.
2: Initialize all elements in the Q-value table old_Q to 0.
3: Initialize the discount factor γ (0 < γ < 0.5, the default is 0.3) in the Q-value table update equation and the reward value matrix R. The matrix R is the reward value matrix and the elements in the matrix take the reward value obtained by assigning processing units to port j of NIC i. The port of NIC forms a mapping relationship with the receiving queue.
4: If the processing unit has not been assigned for the current load demand: 4.1: Initialize a random number rand, if rand < ε, then randomly select an available processing unit from the current state to reach the next state; otherwise, select that load unit with the largest R value from it according to the R matrix.
4.2: $Q(s,a) = R(s,a) + \gamma \cdot \max[Q(s', all\_actions)]$ (7) $Q(s,a)$ is the value Q corresponding to the selected action a in state s, $R(s,a)$ is the reward value R obtained after selecting a processing unit, γ is the discount factor (0 < γ < 0.5), and max[] is the expected value obtained after selecting the processing unit with the largest reward value in the state s′. 4.3: Select the next load demand, take the next state as the current state, and go back to step 4.
5: If the Q-value table does not converge, add 1 to the number of iterations. Reinitialize the current state as the starting point, set old_Q as the current Q-value table, and go back to step 4 to continue learning. If the Q-value table converges, the load allocation scheme is obtained according to the Q-value table. Add the selected set of processing units in the allocation scheme to the load allocation array LOAD_DIST.
6: Exit the loop, return to the LOAD_DIST array, and update the processing unit status.

6.2.2. Algorithm Implementation

The Q-learning algorithm first needs to determine five elements: the environment (Env), the state space (S), the action space (A), the reward value function (R), and the policy or probability of action selection (P). The determination of the reward value function R requires a combination of the final load balancing goal of the algorithm and the upper limit θ of the available load of the processing unit. The larger the R value of a processing unit, the better the performance of the unit. Under the assumption that all the information of the processing unit is known, the R_{ij} function of the processing unit $unit_{ij}$ is designed as follows:

$$R_{ij} = \begin{cases} 0, \frac{data_k + load_{ij}^{used}}{load_{ij}^{total}} > \theta \\ \alpha \cdot e^{-\omega}, \frac{data_k + load_{ij}^{used}}{load_{ij}^{total}} \le avg\_load_{cur} \\ \beta \cdot \alpha \cdot e^{-\omega}, avg\_load_{cur} < \frac{data_k + load_{ij}^{used}}{load_{ij}^{total}} \le \theta \end{cases} \tag{8}$$

$$\beta = \frac{data_k + load_{ij}^{used}}{load_{ij}^{total}} \tag{9}$$

where α denotes the scale factor of the reward value, β denotes the discount factor, and θ denotes the maximum utilization allowed for the processing unit, $0 < \theta < 1$.

From the above equation, the processing unit is divided into three classes:

(1) When the utilization rate of the processing unit after bearing the transfer demand exceeds the limit of the utilization limit θ, the reward value of this processing unit is set to 0, i.e., this processing unit cannot meet the current transfer demand.

(2) When the remaining available load capacity of the processing unit is greater than the load of the transfer demand and the utilization rate of the processing unit after bearing the transfer demand is not greater than the average utilization rate of all processing units, then this processing unit is the optimal choice. The significance of achieving this is to balance the load.

(3) When the remaining available load capacity of the processing unit is greater than the load of the transfer demand and the utilization of the processing unit after bearing this transfer demand is greater than the average utilization rate of all processing units but does not exceed the upper limit of utilization rate θ, then the processing unit is the

second best choice. Therefore, a discount term is added to (2) to reduce the R-value of this processing unit. The reason why the R-value of such a processing unit is not directly set to 0 is to find a feasible solution more quickly.

In summary, this reward value function tends to meet the transfer demand while trying to balance the load on the processing units. To reflect the small gap between variables, an exponential function with a natural logarithm base was used, and the flow chart of the algorithm is shown in Figure 4.

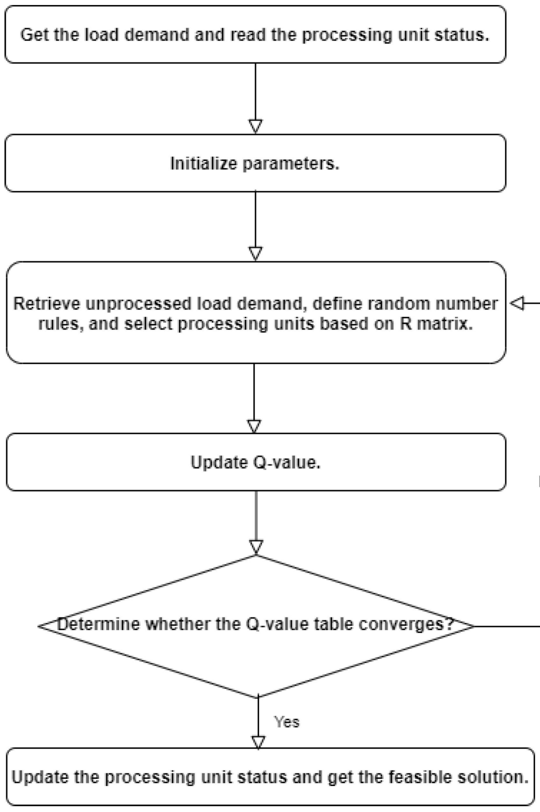

Get the load demand and read the processing unit status.
Initialize parameters.
Retrieve unprocessed load demand, define random number rules, and select processing units based on R matrix.
Update Q-value.
Determine whether the Q-value table converges?
No
Yes
Update the processing unit status and get the feasible solution.

Figure 4. Algorithm flow chart.

The basic flow of the algorithm is Table 3:

Table 3. The basic flow of the algorithm.

```
1: Initialize available unit set and selection operation set.
2: For each demand:
3: Start Q-learning;
4: If Q table is convergent:
5: Break;
6: Use the Q table to obtain unit selection decision.
7: Add the selection decision to selection operation set.
8: Display the load distribution result.
```

We analyzed the convergence of the Q-learning-based load allocation algorithm as shown in Table 2. Firstly, the ε-greedy rule in Step 4.1 ensures that the algorithm will not fall into the local optimum. Secondly, the setting of reward values can guide the selection of the processing unit for a fair load balance. Thirdly, the load allocation scheme is obtained in Step 5 according to the Q-value table if the Q-value table converges. The above measures can effectively ensure the convergence of the algorithm.

7. System Performance Optimization Methods

7.1. Memory Buffer Optimization Strategy

Ideally, the scheduler should be able to control the order in which buffered packets are sent. However, the approach of the re-circulating buffers of the network processor is not customizable. Packets are first copied from the receive (Rx) ring buffer area to the shared transmit (Tx) ring buffer area, and are fed into multiple First Input First Output (FIFO) queues in the traffic manager. This results in packets of all classes becoming mixed in the Tx buffer and treated equally upon exit. If no scheduling is available, the large traffic in the low priority class may easily overwhelm the small traffic in the high priority class.

In order to improve the transfer efficiency, the characteristics of the queue were used to design the binding of the receiving port and the forwarding port, as well as share the same lock-free circular queue, as shown in the mempool part of Figure 2. Meanwhile, during the receiving and forwarding of data processing, data that are not received and forwarded out in time need to be cached. For the setting of the buffer, the previous scheme accommodates more data by dynamically increasing the buffer length. However, when the data flow increases and the receiving and forwarding capacity is limited, the continued increase in the buffer length will cause issues such as memory overflow and wasted system resources, which affect data processing. Considering this problem, a dynamic multicore multi-cycle buffer model was designed to enhance the data processing receiving and forwarding capacity while vertically expanding the buffer to ensure that data can be received within the processing cycle and simultaneously forwarded to other nodes in time for processing and storage [6,7,16,19,31].

DPDK uses a memory pool to manage data. After receiving data, the receiving core first applies for an object from the memory pool and, after completing the de-capsulation of the data, it caches the packets into the cache line corresponding to the write pointer according to the packet identifier and processing cycle, i.e., each line corresponds to a concurrent data stream with the same processing cycle. After completing the caching of data, the maintenance forwarding table, the forwarding table, and the circular data stream buffer table are bound correspondingly. When the corresponding buffer is full, the forwarding table marks the corresponding location as full and records the number of packets cached in each small block. The forwarding core is bound to the circular audio–video stream buffer. It wakes up/sleeps with the amount of data changes and is responsible for data forwarding. The sending pointer points to the concurrent audio–video stream data stream in the buffer that currently needs to be sent. The forwarding core queries the forwarding table and forwards the prepared data according to the configuration table information. After completing data forwarding, the memory space is released and the forwarding table is updated simultaneously [6,7,16].

The cyclic audio–video stream buffer is used to cache the audio–video stream data streams by a processing cycle, where each line of data belongs to the same processing time.

7.2. Data Flow Distribution Optimization Strategy

Data packet scheduling is enforced through a classifier, multiple queues, and a scheduler. Data classification is defined by a user policy and corresponds to separated queues. The queues provide service to aggregated flows through the same conduction policy. The outgoing interface messages are first matched with filtering rules and then grouped into queues. Meanwhile, the scheduler serves these queues to send packets.

Stream classification is the technique by which a network card classifies packets according to their characteristics. The classified information can be presented to the packet processor in different ways, such as recording the classified information in a descriptor and importing the data stream into a certain queue.

The advanced NICs can all analyze the packet type, which is carried in the receiving descriptors. The application can be presented to the packet handler in different ways. The NIC device can also determine its keywords based on the packet type and thus determine its receiving queue based on keyword matching.

7.3. Link Aggregation

Link aggregation refers to the aggregation of multiple physical ports together to form a logical port and achieve the load sharing of outgoing/incoming traffic throughput on each physical port. Subsequently, according to the user-configured load sharing policy, it is decided from which port the network packet is sent to the peer. When a link failure is detected on one of the ports, packets are stopped from being sent through that port. The sending port of the packet is calculated again in the remaining links according to the load-sharing policy, and the faulty port is reassigned as the sending and receiving port after recovery. Link aggregation is a very practical and important technology in increasing link bandwidth, enhancing link transmission resilience and engineering redundancy. The implementation of dynamic link aggregation is based on the LACP (Link Aggregation Control Protocol) protocol, which is used to negotiate link information between two systems. The data link layer between the Media Access Control Address (MAC) sublayer and the physical layer is implemented as a functional module, which provides the MAC sublayer with the same invocation interface as the physical layer, and the aggregated link is a physical interface for the MAC sublayer. Therefore, when the MAC sublayer forwards data, it only needs to send the data to be forwarded to the link aggregation functional module through the logical port provided by the link aggregation [7,16].

8. Experimental Design and Analysis

The Q-learning data stream receiving scheduling model was experimentally validated. An edge cluster was constructed using 10 servers and a 48-port Gigabit 2-layer exchange board. The server configuration details are shown in Table 4. Each server CPU is an 8-core Intel Xeon E3-1230V2@3.3Ghz and two quad-port Intel-I350-T4 Gigabit NICs containing 8 network ports. The system was developed on the basis of DPDK Version 17.11.3 and OVS version 2.9.0. The DPDK consists of eight large pages of memory, multiple port receiving queues, and multiple sending queues. This enables a dynamic adjustment of the number of network ports to be accessed and forwarded. Table 4 is the host configuration information.

Table 4. Host configuration information.

Name	Model/Version	Description
CPU	Intel (R) Xeon (R) E31230 V2@3.3Ghz	2 × 4 cores with 8 threads
Memory	DDR31600Mhz	32 G
OS	Debian9.0/Centos7.0	-
kernel	Linux version 3.10.0862.14.4.e17.x86_64	-
NIC	I350-T4-4-port 1 Gb/s each port PCI-e X4	-
DPDK	DPDK17.11.3	8-Ports (2 × 4)
OVS	2.9.0	-

To verify the transmission performance of Q-learning, several experiments were designed. Traditional UDP transmission was selected for the comparative experiment. In the tests, the traffic generator Pktgen [32] provided by DPDK was used as a packet generation tool to generate packets of different sizes. Each of the 10 servers of the Local Area Network (LAN) has a different number of application connections, and the applications of different nodes mutually communicate. The receiving server receives these messages, filters and resolves them, and then distributes them to the destination application. The transmission performance of traditional UDP and this paper's scheme in each node port is discussed. The transmission rate, packet loss rate, and transmission delay were counted by sending 10 GB of data 10 times and selecting packet sizes of 64 B, 128 B, 256 B, 516 B, and 1024 B, respectively.

8.1. Port Rate

The purpose of the experiments in this section is to verify the optimization effect of the parallel transmission performance of multiple NICs. The benchmark solution is a multi-network, multi-port concurrent transmission. A total of 2 NICs were designed here, each with 4 ports and 8 ports transmitting data individually; the optimization scheme is to use link aggregation transmission, setting 4 ports of the sending server and receiving server NICs as bond 1–4 and bond 5–8 through the balance mode of the ovs-bond system. The Q-learning scheduling transmission scheme of DPDK was used to test the transmission rate, packet loss rate, and time delay when transmitting 64 B small stream packets.

The experimental results are shown in Figure 5. The link aggregation transmission scheme obtains 3.6 times the actual bandwidth and obtains the ideal port transmission optimization effect. When transmitting 64 B small packets, Figure 5 lists the data stream transmission rate of each port. As can be seen, the transmission rate is significantly better than traditional UDP. The reduction in processing time results in a significant reduction in packet loss. When the on-demand stream packets are larger, the on-demand stream is guided through the bond port for transmission. The Q-learning stream transmission scheduling is better than the original DPDK and UDP transmission.

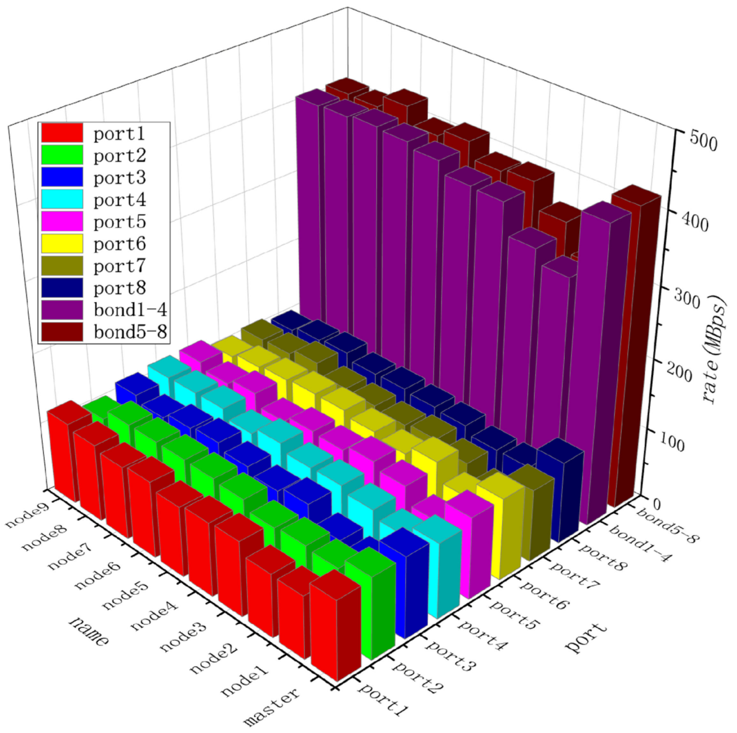


Figure 5. Data flow rate (MBps) of the network port.

8.2. Kernel Counts, Queue Counts, and Queue Length Tests

The purpose of the experiments in this section is to test the effect of the number of cores, the number of pairs of queues, and the queue length on Q-learning scheduling transmission. The emphasis is on packet length, number of packets sent, bandwidth, and other influencing factors, as well as on measuring the transmission delay, number of packets received, packet loss rate, average CPU load, and utilization of each port. In the experiment, the number of CPU cores is 8, the number of queues is 32, and the maximum queue length is 4096 B. The server has 8 cores, 4 LAN ports in each I350 NIC, and 4 packet sending queues and 4 packet receiving queues inside each LAN.

First, the impact of the number of queues on the transfer performance was considered. In the traditional DPDK, the performance of concurrent scheduling is optimal when the number of queues is less than or equal to the number of CPU computing cores, and the average performance is usually better when the number of queues is three times the number of cores. When the number of queues is more than 16, the performance optimization result are not obvious. However, when Q-learning multi-core multi-port scheduling is used, the performance improvement is still relatively obvious.

Next, the impact of the receiving and forwarding packet queue length on the transmission performance was considered. (1) The receiving packet queue length, i.e., the number of receiving packet descriptors allocated to each receiving queue, reflects the maximum ability to cache packets before the software driver reads the received packets. The default length is 128 B. (2) The outgoing queue length, i.e., the number of outgoing packet descriptors assigned to each outgoing queue, is 512 B. The default length is 512 B. If the length is too long, it takes up resources and more latency, and if too short, it tends to cause frequent packet loss by CPU scheduling. In this paper, the length of the Rx ring of DPDK directly reflects the processing capacity: when the input rate from NIC is greater than the processing rate, the length of the Rx ring will definitely increase; otherwise, when the in-put rate is smaller than the processing rate, the Rx queue is destined to decrease. In general, the forwarding queue length is four times the access queue length, and the incoming packet queue length rxd and the outgoing packet queue length txd range from $1 \leq N \leq 65{,}535$; in general, rxd and txd do not exceed 4000 B. When overload occurs, the Rx ring is close to full, which can lead to a sharp drop in performance. For 64 B small packets, the queue lengths were selected to be 512 B and 4096 B, and the packet loss rate was tested.

As shown in Table 5, when the number of sending packets is 100,000,000, by setting the same number of sending and receiving queues, the packet loss rate decreases gradually as the number of cores and queues increase. The concurrent performance of the traditional DPDK and the DPDK with Q-learning scheduling was compared by adjusting the size of the sending packets, the number of CPU cores, the number of sending queues, the number of receiving queues, the length of sending queues, and the length of receiving queues to test the number of received packets and the latency.

Table 5. Test information on the number of cores and the number of queues and queue lengths, as well as the corresponding number of packets sent and received.

Package Length (Byte)	pps	Mbps	CPU Cores	RxTx Queue Numbers	RxTx Queue Length	Number of DPDK Packets Received	DPDK-Delay (s)	Number of Q-Learning-DPDK Packets Received	Q-Learning-DPDK-Delay (s)
512	230,000	960	1	24	512	100,000,000	36.53	100,000,000	31.35
384	295,600	950	1	24	512	100,000,000	32.12	100,000,000	26.17
256	512,000	952	1	24	512	100,000,000	18.74	100,000,000	12.54
128	1,019,000	997	1	24	512	100,000,000	12.35	100,000,000	8.67
64	1,483,000	996	1	1	512	99,271,456	9.27	99,482,355	6.54
64	1,483,000	996	1	24	512	99,907,154	9.14	99,950,126	6.32
64	1,483,000	996	1	24	512	99,147,518	78.56	99,467,518	61.16
64	1,483,000	996	2	24	512	99,192,432	74.44	99,252,432	58.47
64	1,483,000	996	4	24	512	99,084,322	68.35	99,784,322	60.41
64	1,483,000	996	8	24	512	99,160,316	69.42	99,862,157	63.22
64	1,483,000	996	8	24	4096	99,117,416	75.33	99,999,366	65.36

As shown in Figure 6 and Table 5, the Q-learning algorithm has a lower packet loss rate than the traditional DPDK algorithm, with a maximum reduction of 0.48%. In the Q-learning scheduling scheme, the performance continues to improve when the number of queues increases to 24, eventually reducing the average time delay by 21%.

The impact of the number of CPU cores and the number of queues on the average CPU load factor was further considered. As shown in Figure 7, when the number of CPU cores is the same, the longer the queue, the higher the number of receiving burst packets; the higher the queues number, the higher the capacity of receiving data burst packets, and thus the better the transmission performance of the video quantity. When the number of queues and queue length are the same, the larger the number of CPU cores, the larger the number of burst packets received, indicating a better forwarding performance of the system. As the number of CPU cores queues and queue length increase, the number of burst packets received in the experiment is larger, indicating a better forwarding performance. However, the more difficult the scheduling and the longer the scheduling period, the more prominent the boosting advantage of Q-learning scheduling becomes for CPU load balancing. Under the same conditions, Q-learning's DPDK scheduling has a lower average CPU load ratio than RSS scheduling, as shown in Figure 7.

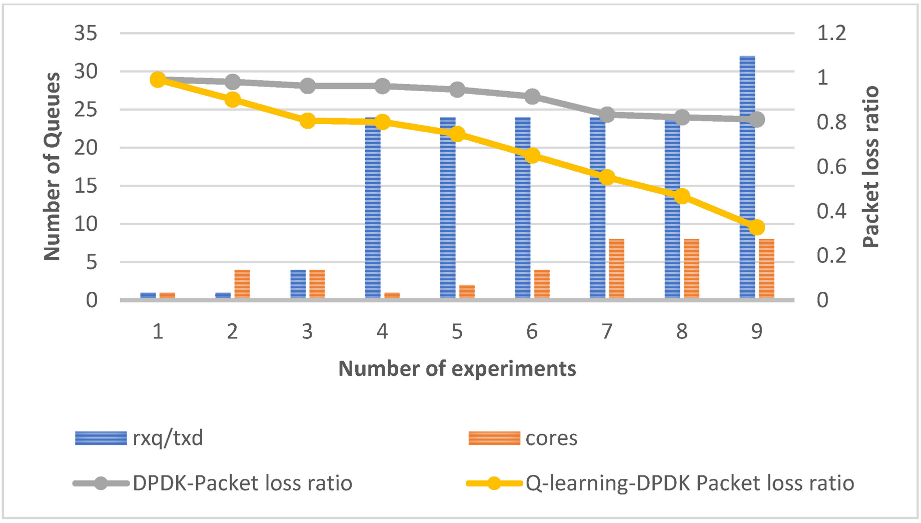


Figure 6. Number of cores, queues, and corresponding packet loss rate when transmitting 64 B small packets.

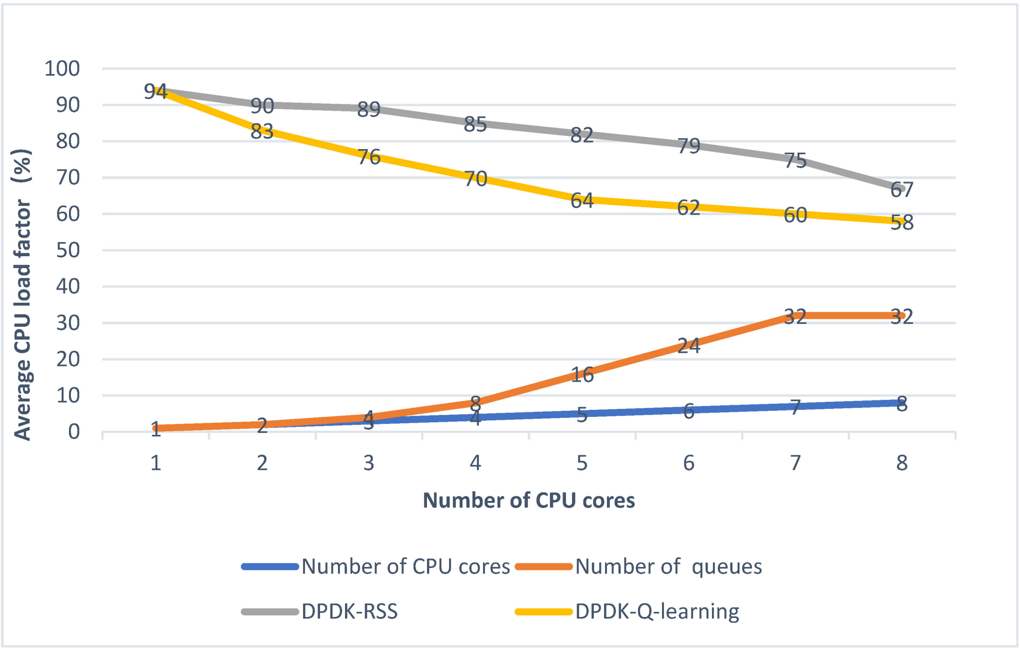


Figure 7. The corresponding relationship between the average CPU load factor of the scheduling algorithm and the number of cores and queues.

8.3. Adjustment in the Number of Ports

The purpose of experiments in this section is to verify the impact of dynamic port optimization on transmission performance. The main idea is to adjust the number of NIC ports according to the classified transmission volume of live and on-demand packets received and test the impact on the transmission performance after dynamically adjusting the number of ports. In the experiments, 2 network cards with 8 ports were used to send 10 GB of data multiple times under an equal or unequal number of network ports. The group sizes were 64 B, 128 B, 256 B, 516 B, and 1024 B. The load factor and packet loss rate of each network port of the sending and receiving servers were calculated. Based on the experiments in the previous section, the traffic was limited to 4 Gb/s, the information of each server node was collected every 5 s, and the utilization rate of each network port was calculated. Figure 8 shows the node network port utilization before and after Q-learning load balancing, which significantly reduces the NIC load rate.

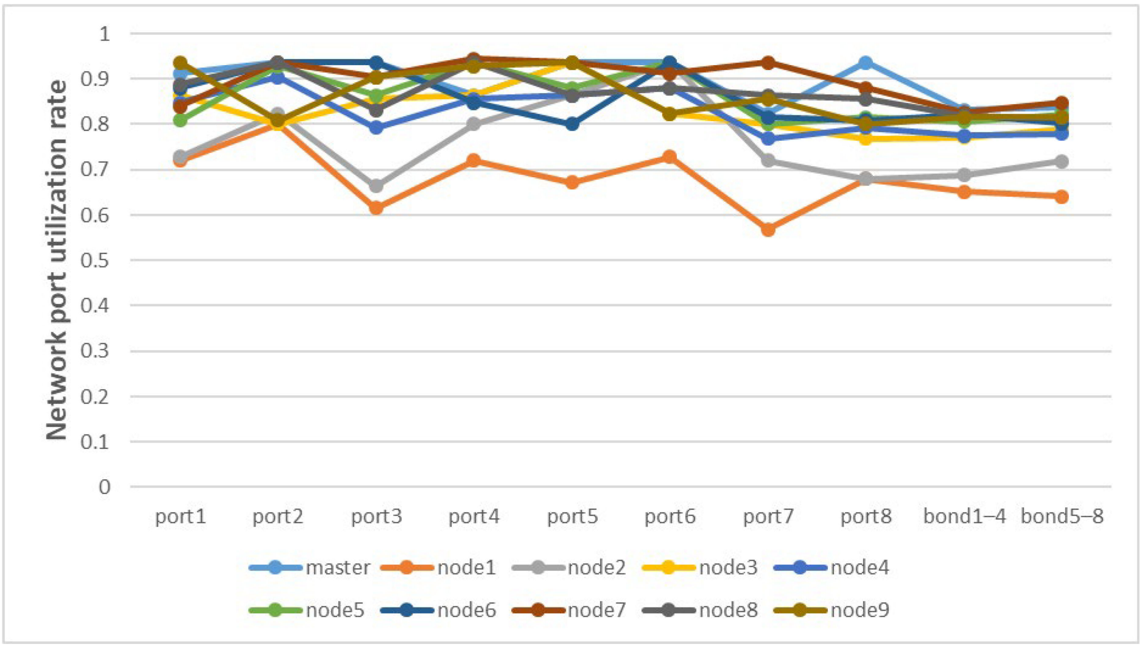


Figure 8. Network port utilization rate for each server.

When the number of receiving ports is close to the number of sending ports, the average load ratio of all cores is the smallest and the performance is optimal. The data loss rate is positively correlated with the number of sending ports; the data stream receiving performance is correlated with the receiving ports; the greater the amount of receiving ports, the better the ability to handle burst traffic. The average CPU load rate is significantly lower when comparing Q-learning's DPDK scheduling with RSS scheduling, and the corresponding relationship between the CPU load rate and the number of receiving and forwarding ports is shown in Figure 9.

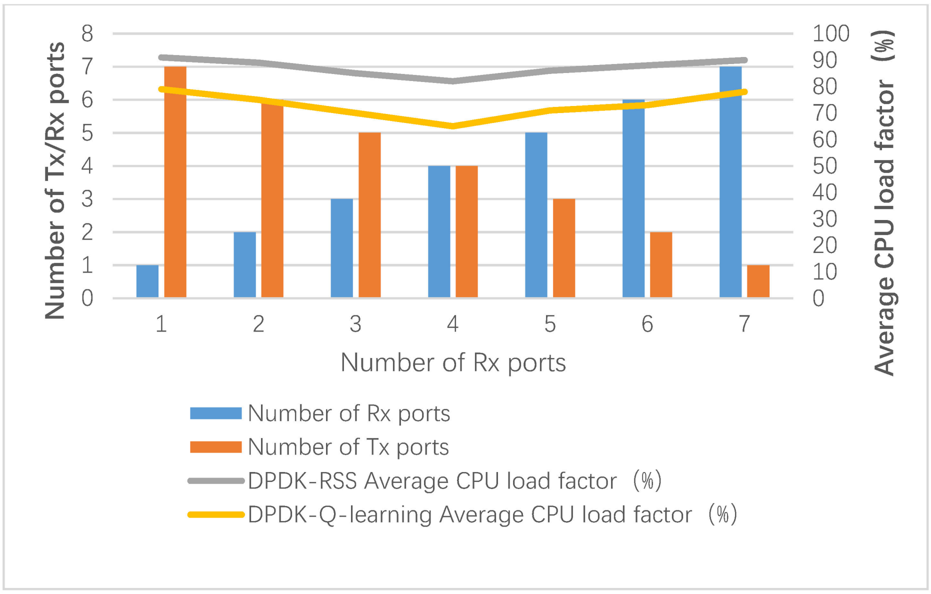


Figure 9. Correspondence between the average CPU load rate of the scheduling algorithm and the number of ingress and egress ports.

8.4. Algorithm Comparison

The experiments in this section compared and analyzed the Q-learning scheduling transmission algorithm with the classical UDP and traditional DPDK port balancing algorithms. In the experiment, these packets with sizes of 64 B, 128 B, 256 B, 512 B, and 1024 B were selected by sending 10 GB of data several times, and the load rate and packet loss rate of each network port were counted.

As shown in Figure 10, the overall throughput improves considerably at different packet loss rates. The comparison between Q-learning and the benchmark UDP transmission algorithm reveals that: 3 times the throughput is obtained at a 0 packet loss rate; 5 times the throughput is obtained at a 0.01 packet loss rate. The comparison between Q-learning and the RSS algorithm reveals that: there is a 37.5% improvement in throughput at a 0 packet loss rate; there is a 23.25% improvement in throughput at a 0.01 packet loss rate.

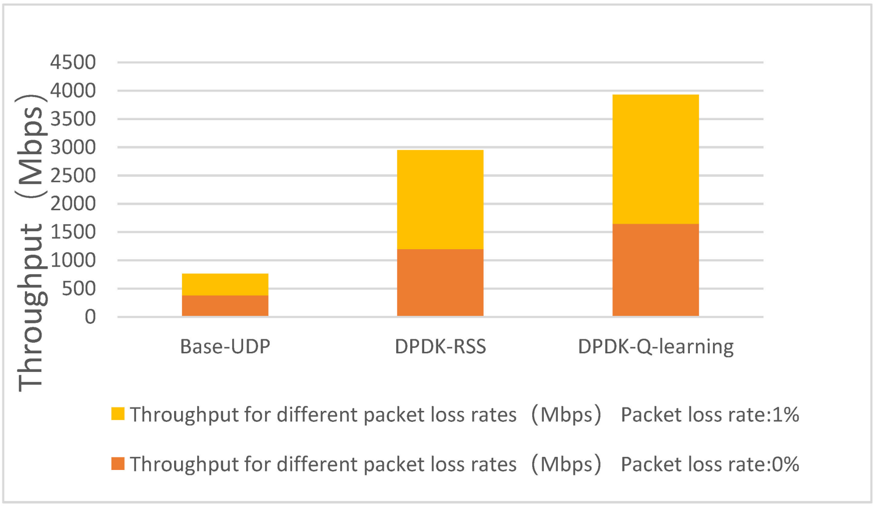


Figure 10. Throughput for different packet loss rates (Mbps).

As seen in Figure 11, the utilization of each network port varies when using the RSS load balancing that comes with the system, and there is a large gap in the performance of different network ports. After adopting Q-learning load balancing, the utilization of each network port becomes relatively smooth, reducing the CPU load rate by 18%. This not only avoids packet loss caused by the concurrent transmission overload of data streams but also improves the server throughput.

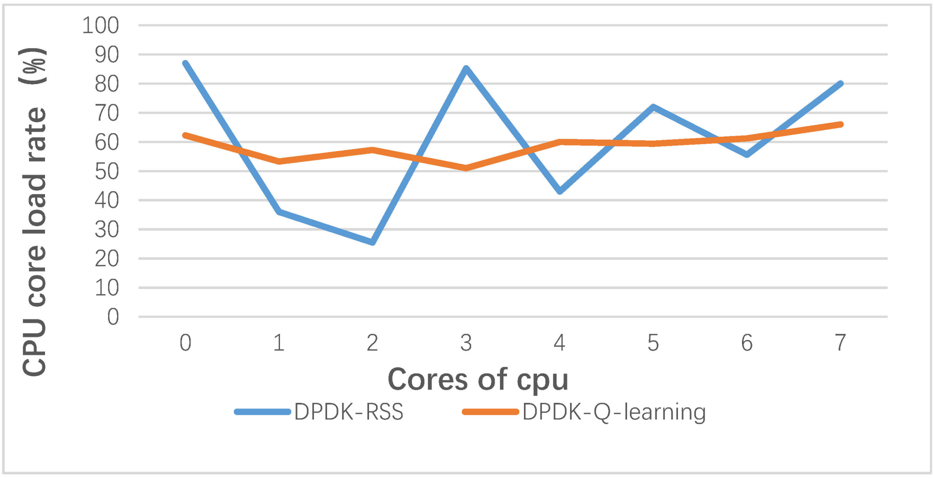


Figure 11. Comparison of the CPU core load rate.

9. Conclusions

For the application scenario of transmitting live and on-demand video streams within an edge cluster, an access system for building highly concurrent data streams using low-cost, low-speed NICs is proposed. The link aggregation of multiple NICs was realized through

DPDK technology to achieve logical high-bandwidth links. The Q-learning algorithm was taken as the core to design the data stream scheduling model, which switches the time slice according to the number of live and on-demand viewers and dynamically updates the number of CPU processing units and transmission load. The Q-learning method was used to select the processing unit with a lower transmission load as the preference; a reward value function was set for achieving load balancing and improving processing efficiency, and the candidate processing units were divided into three different levels to complete the forwarding selection; CPU balanced load scheduling with multiple NICs and multiple queues was used to optimize the system's overall transmission performance in order to achieve the minimization of the multi-core load balancing of CPUs. The proposed method not only provides a single-link multi-NIC bandwidth overlap function but also reduces the programming complexity of the concurrent transmission of large data. The core scheduling algorithm ensures the reliability and performance of network transmission while improving the scalability of the overall system. However, more types of network card tests and verification of more detailed system parameters are lacking. In the future, the DPDK-based Q-learning data stream transmission scheduling will be used in the storage system of the edge cluster and in the large-scale data transmission with a high concurrency, high throughput, and low delay in data centers, as well as to optimize model parameters to improve the applicability of the system. In addition, we will put data encryption and decryption and DPDK classification technology into future work.

Author Contributions: Conceptualization, H.L.; methodology, Q.C.; software, H.L.; validation, H.L. and P.L.; formal analysis, H.L. and P.L.; investigation, H.L.; resources, H.L. and Q.C.; data curation, H.L.; writing—original draft preparation, H.L.; writing—review and editing, H.L. and P.L.; visualization, H.L.; supervision, Q.C.; project administration, H.L.; funding acquisition, Q.C. and H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Shanghai Key Science and Technology Project (19DZ1208903); National Natural Science Foundation of China (Grant Nos. 61572325 and 60970012); Ministry of Education Doctoral Fund of Ph.D. Supervisor of China (Grant No. 20113120110008); Shanghai Key Science and Technology Project in Information Technology Field (Grant Nos. 14511107902 and 16DZ1203603); Shanghai Leading Academic Discipline Project (No. XTKX2012); Shanghai Engineering Research Center Project (Nos. GCZX14014 and C14001); Introduction and Cultivation Program for Young Innovative Talents of Universities in Shandong (2021QCYY003).

Data Availability Statement: Data available in a publicly accessible repository that does not issue DOIs. Publicly available datasets were analyzed in this study.

Acknowledgments: The authors would like to thank the editors and anonymous reviewers for their valuable comments and suggestions. The authors would like to acknowledge the support from the Flow Computing Laboratory at University of Shanghai for Science and Technology.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cisco. Cisco Annual Internet Report (2018–2023) White Paper. 2020. Available online: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (accessed on 2 May 2023).
2. Zheng, Y.; Xiaowu, H.; Jiaxing, W.; Jing, W.; Yi, Z. Edge computing Technology for Real time Video Stream Analysis. *China Sci. Inf. Sci.* **2022**, *52*, 1–53.
3. Jedari, B.; Premsankar, G.; Illahi, G.; Di Francesco, M.; Mehrabi, A.; Ylä-Jääski, A. Video caching, analytics, and delivery at the wireless edge: A survey and future directions. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 431–471.
4. Altamimi, S.D.; Shirmohammadi, S. QoE-Fair DASH Video Streaming Using Server-side Reinforcement Learning. *ACM Trans. Multimed. Comput. Commun. Appl.* **2020**, *16*, 1–21. [CrossRef]
5. Ueno, Y.; Nakamura, R.; Kuga, Y.; Esaki, H. P2PNIC: High-Speed Packet Forwarding by Direct Communication between NICs. In Proceedings of the IEEE INFOCOM 2021—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Vancouver, BC, Canada, 10–13 May 2021; pp. 1–6. [CrossRef]
6. Li, C.; Chen, Q. Dynamic monitoring model based on DPDK parallel communication. *J. Comput. Appl.* **2020**, *40*, 335–341.
7. Wu, M.; Chen, Q.; Wang, J. A Flexible High-Speed Bypass Parallel Communication Mechanism for GPU Cluster. *IEEE Access* **2020**, *8*, 103256–103272. [CrossRef]

8. Zhou, D.; Fan, B.; Lim, H.; Kaminsky, M.; Andersen, D.G. Scalable, high performance ether-net forwarding with CuckooSwitch. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13), New York, NY, USA, 9–12 December 2013; pp. 97–108. [CrossRef]
9. Hu, N.; Cen, X.; Luan, F.; Sun, L.; Wu, C. A Novel Video Transmission Optimization Mechanism Based on Reinforcement Learning and Edge Computing. *Mob. Inf. Syst.* **2021**, *2021*, 6258200. [CrossRef]
10. Qi, Z. A novel video delivery mechanism for caching-enabled networks. *Multimed. Tools Appl.* **2020**, *79*, 25535–25549.
11. Gao, Y.; Wei, X.; Kang, B.; Chen, J. Edge Intelligence Empowered Cross-Modal Streaming Transmission. *IEEE Netw.* **2020**, *35*, 236–243. [CrossRef]
12. Xu, T.; Chen, X.; Wu, C.; Wang, J.; Zheng, R.; Liu, D.; Tan, Y.; Ren, A.; Li, J. 3DS: An Efficient DPDK-based Data Distribution Service for Distributed Real-time Applications. HPCC/DSS/SmartCity/DependSys. In Proceedings of the 8th DependSys 2022, Hainan, China, 18–20 December 2022; pp. 1283–1290.
13. Tashtarian, F.; Falanji, R.; Bentaleb, A.; Erfanian, A.; Mashhadi, P.S.; Timmerer, C.; Hellwagner, H.; Zimmermann, R. Quality Optimization of Live Streaming Services over HTTP with Reinforcement Learning. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 1–6. [CrossRef]
14. Park, P.K.; Moon, S.; Hong, S.; Kim, T. Experimental Study of Zero-Copy Performance for Immersive Streaming Service in Linux. In Proceedings of the 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 19–21 October 2022; pp. 2284–2288. [CrossRef]
15. Zeng, Z.; Monis, L.; Qi, S.; Ramakrishnan, K.K. DEMO: MiddleNet: A High-Performance, Lightweight, Unified NFV &Middlebox Framework. In Proceedings of the 2022 IEEE 8th International Conference on Network Softwarization (NetSoft), Milan, Italy, 27 June 2022–1 July 2022; pp. 246–248.
16. Wu, M.; Chen, Q.; Wang, J. Toward low CPU usage and efficient DPDK communication in a cluster. *J. Supercomput.* **2022**, *78*, 1852–1884. [CrossRef]
17. Shi, J.; Pesavento, D.; Benmohamed, L. NDN-DPDK: NDN Forwarding at 100 Gbps on Commodity Hardware//ICN '20. In Proceedings of the 7th ACM Conference on Information-Centric Networking, ACM, Virtual Event, 29 September–1 October 2020.
18. Haecki, R.; Humbel, L.; Achermann, R.; Cock, D.; Schwyn, D.; Roscoe, T. CleanQ: A lightweight, uniform, formally specified interface for in-tra-machine data transfer. *arXiv* **2019**, arXiv:1911.08773.
19. Schramm, N.; Runge, T.M.; Wolfinger, B.E. The Impact of Cache Partitioning on Software-Based Packet Processing. In Proceedings of the 2019 International Conference on Networked Systems (NetSys), Munich, Germany, 18–21 March 2019.
20. Zou, P.; Ozel, O.; Subramaniam, S. Optimizing Information Freshness Through Computation-Transmission Tradeoff and Queue Management in Edge Computing. *IEEE/ACM Trans. Netw.* **2021**, *29*, 949–963. [CrossRef]
21. Intel. Intel DPDK: Programmers Guide [OL]. Available online: https://doc.DPDK.org/guides/index.html (accessed on 1 July 2022).
22. Kai, L.; Lin, Y.; Xiangzhan, Y.; Yang, H. Dynamic load balancing method for traffic based on DPDK. *Intell. Comput. Appl.* **2017**, *7*, 85–89, 91.
23. Li, C.; Song, L.; Zeng, X. An Adaptive Throughput-First Packet Scheduling Algorithm for DPDK-Based Packet Processing Systems. *Futur. Internet* **2021**, *13*, 78. [CrossRef]
24. Pandey, A.; Bargaje, G.; Krishnam, S.; Anand, T.; Monis, L.; Tahiliani, M.P. DPDK-FQM: Framework for Queue Management Algorithms in DPDK. In Proceedings of the 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Leganes, Spain, 10–12 November 2020.
25. Xi, S.; Li, F.; Wang, X. FlowValve: Packet Scheduling Offloaded on NP-based Smart NICs. In Proceedings of the 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), Bologna, Italy, 10–13 July 2022; pp. 347–358. [CrossRef]
26. Heping, F.; Shuguang, L.; Yongyi, R.; Kunhua, Z. Integrated scheduling optimization of multiple data centers based on deep reinforcement learning. *J. Comput. Appl.* **2023**, *1*, 1–11.
27. Li, X. An efficient data evacuation strategy using mul-ti-objective reinforcement learning. *Appl. Intell.* **2022**, *52*, 7498–7512. [CrossRef]
28. Yi, S.; Li, X.; Wang, H.; Qin, Y.; Yan, J. Energy-aware disaster backup among cloud datacenters using mul-ti-objective reinforcement learning in software defined network. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e6588. [CrossRef]
29. Qin, Y.; Wang, H.; Yi, S.; Li, X.; Zhai, L. An energy-aware scheduling algorithm for budget-constrained scientific workflows based on multi-objective reinforcement learning. *J. Supercomput.* **2019**, *76*, 455–480. [CrossRef]
30. Qin, Y.; Wang, H.; Yi, S.; Li, X.; Zhai, L. A mul-ti-objective reinforcement learning algorithm for time constrained scientific workflow scheduling in clouds. *Front. Comput. Sci.* **2021**, *15*, 24–35. [CrossRef]
31. Xie, J.; Miao, M.; Ren, F.; Cheng, W.; Shu, R.; Zhang, T. Overload Detecting in High Performance Network I/O Frameworks. In Proceedings of the IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Sydney, NSW, Australia, 12–14 December 2016.
32. Wiles, K. Pktgen-DPDK [EB/OL]. Available online: https://github.com/pktgen/Pktgen-DPDK (accessed on 3 July 2022).