BPCM: A Flexible High-Speed Bypass Parallel Communication Mechanism for GPU Cluster

MINGJIE WU1, QINGKUI CHEN1,2, AND JINGJUAN WANG1
1School of Management, University of Shanghai for Science and Technology, Shanghai 200093, China
2School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China

Corresponding author: Qingkui Chen (chenqingkui@usst.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61572325 and Grant 60970012, in part by the Ministry of Education Doctoral Fund of Ph.D. Supervisor of China under Grant 20113120110008, in part by the Shanghai Key Science and Technology Project in Information Technology Field under Grant 14511107902 and Grant 16DZ1203603, in part by the Shanghai Leading Academic Discipline Project under Grant GCZX14014 and Grant C14001, and in part by a Cooperation Project with the Intel Asia Pacific Research and Development Center.

ABSTRACT

With the increasing complexity of computational tasks faced by artificial intelligence technology, the scale of machine learning models continues to expand, and the data volume and frequency of parameter synchronization also increase. This will cause the communication bandwidth within the GPU cluster to become the biggest bottleneck for distributed model training. Many existing solutions cannot be widely promoted due to the need for professional equipment support, high cost, and difficulty in use. To solve this problem, this paper proposes a multi-network card bypass parallel communication mechanism based on Intel DPDK technology to increase the bandwidth within the GPU cluster at a lower cost and make full use of the idle CPU resources of the GPU server to accelerate data transmission. Firstly, we propose a data transmission model based on multiple network cards, and design a port load balancing algorithm to ensure load balancing of multiple network cards. Secondly, the model and algorithm of CPU multi-core scheduling are implemented to reduce CPU energy consumption, resource occupation, and the impact on other applications. Furthermore, for multiple application scenarios, a rate adjustment model and algorithm are designed and implemented to ensure fair use of application bandwidth. Finally, the experimental results show that this mechanism can provide high bandwidth for GPU clusters with inexpensive multi-network cards, and provide superimposed bandwidth of multi-network cards in a single connection, which has high reliability and transmission efficiency, and is simple to use and flexible to expand.

INDEX TERMS

DPDK, GPU cluster, multi-core, multi-NIC, data link layer, bypass parallel communication.

I. INTRODUCTION

In recent years, artificial intelligence technology has achieved unprecedented breakthroughs and achievements in many application fields relying on powerful computing power and massive training data. With the continuous expansion of artificial intelligence applications, the difficulty of solving problems is increasing, the computing tasks are becoming more and more complex, and the scale of machine learning models is becoming larger and larger, which propose higher requirements on the computing power of computers. It may take weeks to complete model training on a machine equipped with a single GPU, so it is a trend to extend model training to multiple GPUs [1]. The current popular single-machine multi-GPU server has a limited number of GPUs, and the cost is too high. Moreover, it cannot perform well in larger-scale model training. This has led to a distributed machine learning framework with multiple machines and multiple GPUs that is becoming a hot topic in the industry. There are many open-source distributed machine learning systems, such as Spark, PyTorch, and TensorFlow [2]. With the continuous increase of the scale of machine learning models and the rapid development of computing engines such as GPUs, not only the amount of training data required by model training has become larger, but also, the amount and frequency of parameter synchronization data has increased. The surge in the amount of interactive data among GPU nodes during model training will make network communication between...
two nodes in the GPU cluster become the biggest bottleneck that limits the expansion of distributed machine learning systems.

Although the current network equipment bandwidth has increased from the original 100Mbps to 1Gbps or even 10Gbps, the growth rate of network bandwidth is far less than the network traffic, and the high cost also limits the popularity of higher bandwidth networks. The defects of traditional network protocols and processing mechanisms also cause network bandwidth utilized at a lower rate. Driven by cost constraints and the need for high-speed transmission, improving network protocols and packet processing mechanisms, multi-NIC parallel communications is a new way out. In 2014, Intel introduced a data plane development kit DPDK which designed to cope with high-speed transmission network packets [3]. DPDK makes use of technologies such as user-space drive, CPU affinity, huge pages memory, and PMD to make the transmission and reception rate of the network card close to the line speed [4], [5]. DPDK bypasses the traditional kernel protocol stack and directly passes data at the data link layer to the user process for processing, reducing the data delivery delay in the communication process. DPDK’s support for multiple cores and multiple network ports can double the communication bandwidth [6]. The various features of DPDK can greatly improve the efficiency of network communication [7], [8].

In this context, this paper attempts to apply DPDK technology to distributed GPU clusters, aiming to provide it with a high-speed, high-bandwidth communication mechanism. The idle CPU resources of the GPU server provide the basis for high-speed communication. DPDK can make full use of the characteristics of CPU multi-core, take advantage of the characteristics of multiple network cards, and achieve exponential growth of communication bandwidth. Firstly, we propose a multi-NIC bypass parallel communication mechanism BPCM based on the DPDK technology, and provide users with a function call interface similar to traditional Socket. Secondly, the data transmission model of multi-NIC is proposed, and a port load balancing algorithm is designed for it to ensure the load balance of multiple network cards. Then, we design a CPU multi-core scheduling algorithm to reduce CPU power consumption and resource consumption, thereby reducing the performance impact on other server applications. Finally, for multiple application scenarios, a rate adjustment model and algorithm are designed and implemented to ensure fair use of multiple application bandwidths. BPCM is a bypass communication mechanism developed based on the second layer of the network, which is compatible with the traditional communication mechanism. BPCM only needs to add multiple network cards to the server and connect through the layer 2 switch, no additional professional equipment is required. Therefore, BPCM is not only inexpensive, but also simple to use and flexible to expand.

In the BPCM communication framework, the node monitoring mechanism in the cluster, the master node failure recovery mechanism, and the node dynamic expansion mechanism are designed and implemented by other members of the team. Due to the high-speed data transmission rate of BPCM, the traditional TCP reliable transmission protocol is not applicable. The BPCM reliable transmission protocol will be introduced in the next paper. This paper only discusses end-to-end unreliable data transmission at high speed.

In the next section, we will describe the background and motivation of BPCM. Then we introduce the physical topology, cluster structure and system framework of BPCM in the third section. In the fourth section, we introduce the design and implementation of BPCM in detail. We will verify multiple performances of BPCM by designing multiple sets of experiments in the fifth section, and conclude conclusion in sixth section.

II. BACKGROUND AND MOTIVATIONS

A. BACKGROUND

In a distributed machine learning system, communication time-consuming among GPU nodes is easy to become the bottleneck of the entire distributed training task [9]. To solve this problem, researchers have proposed many excellent solutions from the aspects of reducing data transmission volume, increasing communication bandwidth, improving network communication protocols, and underlying physical topology.

In the distributed training process, excellent parameter synchronization architecture or algorithms can reduce the amount of communication data and delay. At present, the parameter synchronization architecture widely used in large-scale distributed machine learning systems includes: Parameter Server (PS) architecture, Ring architecture [10] and Hierarchical Parameter Synchronization (HiPS) [11]. The parameter server (PS) architecture is widely adopted due to its advantages of simple deployment, good scalability, and robustness. Since the number of parameter servers is far less than the number of working machines, it is easy to cause parameter servers to become network bottlenecks in general. Besides, the introduction of parameter servers also increases the hardware cost of the system. Although stronger hardware devices can greatly reduce communication delays, the corresponding costs will also increase. In the Ring architecture, each node only communicates with two adjacent nodes before and after it, avoiding centralized communication bottlenecks. However, the parameters are sequentially transmitted in the communication ring, which often takes more times to complete synchronously, and the communication overhead increases [12]. Hierarchical Parameter Synchronization (HiPS) [13] divides nodes into multiple groups, and performs parameter synchronization layer by layer and in parallel, thereby reducing the impact of communication delays [14], [15]. DAIET proposed the use of programmable switches to implement In-Network Aggregation (INA) to transfer the logic of parameter aggregation from a dedicated parameter server to a switch with high throughput [16]. Parameter Hub proposed PBox, which is equipped with multiple network cards on a centralized PS to match IO performance with memory bandwidth [17].
Although the 10 Gigabit network card can provide 10 Gbps interaction among multiple GPUs in single machine.

Blink is also only suitable for data DGX servers, but cannot scale to ordinary multi-machine Protocol), RoCE (RDMA over Converged Ethernet), etc., technology also requires professional hardware equipment to implement, such as InfiniBand, iWarp (Internet Wide Area RDMA Protocol), RoCE (RDMA over Converged Ethernet), etc., and the price is higher. NVLink performs well on NVIDIA DGX servers, but cannot scale to ordinary multi-machine GPU clusters [28], [29]. Blink is also only suitable for data interaction among multiple GPUs in single machine. Although the 10 Gigabit network card can provide 10 Gbps bandwidth, compared to 100 Gbps bandwidth among multiple GPUs in a single machine, it still easily becomes a communication bottleneck between two nodes in the GPU cluster. Through the Linux system bonding mode to achieve multiple network card bandwidth overlay, not only a single connection cannot break the theoretical upper limit of the bandwidth of a single network card, but also requires the switch to support IEEE802.3ad, and the configuration is cumbersome.

B. MOTIVATIONS

If we want to achieve high-speed data transmission at low cost, many aspects can be used for reference in previous research. In the RDMA technology, the system kernel is bypassed, and user-space directly performs data transmission with the RNIC, which can greatly reduce the processing delay of the system protocol stack on data packets. The traditional TCP/UDP protocol has insufficient performance in congestion control and system implementation. Its transmission efficiency is not high enough which is more prominent in the local area network and needs to be improved. The idea of multi-network card bandwidth superposition proposed by Parameter Hub and BCube can not only use multiple low-cost gigabit network cards to provide the bandwidth of the 10 Gigabit network card, or multiple 10 gigabit network cards to provide 100 Gbps bandwidth, but also the price is cheaper.

Intel’s DPDK technology can provide support for the implementation of the above requirements. BPCM is based on DPDK. BPCM uses the DPDK user mode driver technology to move the processing of data packets to user-space. Not only does it bypass the system kernel protocol stack, enabling network cards to approach line-rate transmission, but also can customize communication protocols based on the data link layer. BPCM also combines the characteristics of DPDK multi-core and multi-port to achieve parallel high-speed transmission of multiple network ports, which can achieve bandwidth overlay. The advantages of BPCM are as following:

1) Ordinary network cards and Layer 2 switches can be used, and the cost is low.
2) Bypass transmission does not conflict with traditional network architecture.
3) Support multiple network cards, easy to expand network bandwidth.
4) Avoiding the traditional kernel protocol stack, the data processing speed is faster.
5) Custom protocols based on data links are implemented in user space, with high data transmission efficiency and security.
6) Give full play to the characteristics of CPU multi-core; make full use of idle CPU resources in a distributed machine learning environment.
7) Based on the Layer 2 switch topology, the cluster scale is easy to expand, simple to use, and flexible in configuration.
8) Provide a superimposed bandwidth of multiple network cards for a single connection.

III. DESIGN OF BPCM

The BPCM proposed in this paper is a high-speed transmission mechanism based on Intel’s DPDK technology. It not only designs from the underlying physical topology, cluster structure, system framework, communication model, etc., but also makes full use of the characteristics of the server CPU’s multi-core and multi-port to provide user applications with high-speed communication interfaces similar to the traditional Socket communication model. The detailed description of BPCM is given below.

A. PHYSICAL TOPOLOGY

BPCM is a bypass parallel communication mechanism based on the data link layer and combined with DPDK’s multi-NIC technology. It has no impact on the traditional network structure. It can be achieved by connecting multiple network cards of the server to the Layer 2 switch through ordinary network cables. For the expansion of the BPCM cluster nodes, there is no conflict with the traditional switch cascading technology. It is only necessary to connect multiple network cards of the server to be expanded to the Layer 2 switch through network cables. As shown in Fig. 1, node1, node2, and node3 are connected through a Layer 2 switch to realize the physical topology of BPCM without affecting the traditional network structure.

B. CLUSTER STRUCTURE

The BPCM cluster adopts a simple master-slave architecture model, as shown in Fig. 2. The master node needs to collect the real-time information of each node regularly and synchronize to each slave node to ensure that each slave node has a record of the instant information of any node in the cluster. The real-time information of these nodes includes the number of network cards, the real-time load of each network card, and the node CPU utilization. The research on high reliability and high availability in the BPCM cluster master-slave architecture is not discussed in this paper, and this task is the responsibility of the rest of the team.

C. SYSTEM FRAMEWORK

BPCM provides users with a calling interface similar to traditional Socket communication, and the underlying communication protocol is completed by BPCM. BPCM encapsulates functions based on DPDK technology. The core layer includes modules such as multi-NIC load balancing, energy consumption control, rate adjustment, and reliable transmission, as shown in Fig. 3. The port load balancing module realizes the balanced scheduling of the tasks of each network card to avoid overloading the tasks of a single network card. The energy consumption control module can dynamically schedule CPU cores for processing according to the task scale. When the task is heavy, it wakes up other cores to transmit data; when the task is light, it makes the core sleep and frees up CPU resources. The rate adjustment module can perform rate matching transmission according to the receiving capability of the destination node and the sending capability of the local machine to avoid useless data transmission. The rate adjustment module can also ensure the fairness of data transmission in the case of multi-user data transmission. The reliable transmission module implements a custom transmission protocol, which can ensure reliable delivery during high-speed transmission. Due to the high-speed parallel communication characteristics of BPCM, traditional reliable transmission protocols are not applicable and need to be re-designed. The reliable transmission of BPCM will be introduced in detail in the next paper. This paper only studies based on unreliable data packet transmission.

IV. IMPLEMENT OF BPCM

A. COMMUNICATION MODEL

BPCM aims to provide users with a simple and easy-to-use communication interface similar to traditional Socket, and the low-level communication process is transparent to users. Users only need to call the interface provided by BPCM to realize data transmission and reception, without
having to care about the underlying implementation. BPCM implements a full-duplex protocol, that is, the data sender and receiver can communicate with each other at the same time during the communication process.

1) DATA TRANSFER MODEL
Similar to the traditional Socket communication process, user applications in BPCM communicate with each other through bound ports. The user application needs to bind the receiving port, and BPCM will allocate an independent receiving buffer for data reception. The user application specifies the sending host number and the receiving port, and sends data to the common sending buffer. The bottom transmission core of the BPCM takes out the data to be transmitted from all users in turn from the public transmission buffer, and then selects the network card port for transmission according to the port load balancing policy. The BPCM bottom receiving core receives the data packet from the network port, parses it, and puts it into the corresponding receiving buffer according to the port number. The data transmission model of BPCM is shown in Fig. 4. UA_1 of Node_1 and UA_1 of Node_2 communicate with each other.

2) MEMORY MODEL
BPCM is implemented based on DPDK, and uses many memory objects provided by DPDK to process data packets. Fig. 5 shows the BPCM memory model. When a user sends data, first put the data to be sent into the Mbuf object space requested from the DPDK memory pool and encapsulate the header information, and then add the Mbuf pointer to the public sending buffer ring. The sending core of BPCM reads the Mbuf pointer to be sent in turn from the public sending buffer ring, parses the destination node number, calculates the source node send port and destination node receive port, fills in the Ethernet packet header MAC information, and adds it to Send_mbuf_table; When full, the DPDK port sending function is called for batch sending. The receiving core of BPCM receives data packets in batches from the network port into Recv_mbuf_table, and analyzes and processes them sequentially; adds the data packet Mbuf pointer to the corresponding receiving ring according to the port number in the data packet. The user takes the Mbuf pointer from the corresponding private receiving buffer ring and reads the data. Relying on pointers for data transfer throughout the process reduces the number of memory-copy and reduces processing delays.

3) MULTIPLE PHYSICAL CHANNELS
With the increase in the number of computers in the BPCM cluster and the expansion of the network scale, a single switch cannot meet the access requirements of multiple network card servers. Moreover, the failure of a single switch will render the entire BPCM unusable. Therefore, BPCM cluster needs to interconnect multiple switches to replace a single switch. To achieve the high availability of the BPCM cluster, a reasonable network connection topology is necessary. As shown in Fig. 6, multiple network cards of a server can be connected to multiple switches to achieve redundancy of physical links, and the connection is still available when the switch fails. For example, when the Layer 2 switch 2 in Fig. 6 fails, node 1 and other nodes can communicate with each other through several other switches, which is the solid line in Fig. 6. For multiple switches, users can choose cascading, stacking, and clustering methods according to their needs.

B. PORT LOAD BALANCING
Because BPCM is multi-port parallel communication, port load balancing is essential. Traditional port distribution
strategies include random methods, polling methods, balanced round-robin methods, and weighted polling methods. Because it only considers the fairness of data transmission and reception, it does not take into account the actual operating environment (such as receiver port load, port performance, etc.), which is prone to port congestion and a large number of Packet loss problems. As each node in the BPCM can obtain the receiver’s instant information, this paper combines the port load information of the sender and receiver to propose a dynamic load balancing mechanism to ensure the stability of data transmission.

1) PORT LOAD BALANCING MODEL

Only efficient port load balancing algorithms can guarantee high-speed data transmission, and overly complex algorithms are not suitable for BPCM. The dynamic load balancing model in this paper is shown in Fig. 7. The sending core takes the mbuf to be sent from the public sending buffer and extracts the five-tuple information Quintuple 
\[(s\_node, d\_node, s\_pump, d\_pump, ptype)\], where s\_node represents the source node number, d\_node represents the destination node number, and s\_pump represents the source port Number, d\_pump indicates the destination port number, and ptype indicates the packet type. The extracted five-tuple information is calculated by using a Microsoft Toppliz algorithm to calculate a hash value, and then performing a modulo operation with the total number of the sender and receiver sides to select a sending network port and a receiving network port. According to the load of the selected network port, decide whether to call the adjustment function to reselect a suitable network port. Finally, an optimal quad (Quad 
\[(s\_port, s\_mac, d\_port, d\_mac)\]) is obtained. After the corresponding information is filled in the Ethernet packet header, the DPDK port is called to send data.

2) PORT LOAD BALANCING ALGORITHM

In this section, the network port load information model is established based on the network card dynamic parameters obtained by the network port statistics function provided by DPDK. Each symbol is described in detail in Table 1.

| Symbol          | Description                                      |
|-----------------|--------------------------------------------------|
| port\(_i\)      | The i-th NIC                                     |
| \(N\)          | Total number of NICs                             |
| \(T\)          | Data collection cycle                            |
| \(r\)          | Adjustment factor, usually set to 0.8            |
| \(\delta\)     | Load balancing threshold, user setting value     |
| \(N\_ip\_port\_i\) | Total number of successfully received packets    |
| \(N\_op\_port\_i\) | Total number of successfully transmitted packets |
| \(N\_ib\_port\_i\) | Total number of successfully received bytes       |
| \(N\_ob\_port\_i\) | Total number of successfully transmitted bytes   |
| \(N\_imp\_port\_i\) | Total number of dropped packets                  |
| \(N\_eop\_port\_i\) | Total number of erroneously transmitted packets  |
| \(V\_ip\_port\_i\) | Theoretical maximum transfer rate (bps)          |
| \(V\_op\_port\_i\) | The rate (bps) of successfully received packets  |
| \(V\_ib\_port\_i\) | The rate (bps) of successfully transmitted packets|
| \(V\_ob\_port\_i\) | The rate (bps) of successfully received bytes    |
| \(V\_imp\_port\_i\) | The rate (bps) of dropped packets                |
| \(V\_eop\_port\_i\) | The rate (bps) of erroneously transmitted packets|
| \(V\_mb\_port\_i\) | The rate (bps) of dropped bytes                  |
| \(V\_eb\_port\_i\) | The rate (bps) of erroneously received bytes     |
| \(V\_eob\_port\_i\) | The rate (bps) of erroneously transmitted bytes  |
| \(P\_ib\_port\_i\) | The average packet size(byte) of received bytes  |
| \(P\_ob\_port\_i\) | The average packet size(byte) of transmitted bytes|
| \(L\_i\_port\_i\) | The receive load of the i-th NIC                 |
| \(L\_i\_port\_i\) | The transmit load of the i-th NIC                |

For the universal full-duplex network card, we can get six basic dynamic parameters, \(N\_ip\_port\_i\), \(N\_op\_port\_i\), \(N\_ib\_port\_i\), \(N\_ob\_port\_i\), \(N\_imp\_port\_i\), \(N\_eop\_port\_i\). By collecting through the period T, the rate values of the parameters \(V\_ip\_port\_i\), \(V\_op\_port\_i\), \(V\_ib\_port\_i\), \(V\_ob\_port\_i\), \(V\_imp\_port\_i\), \(V\_eop\_port\_i\) of the network port during the T period can be calculated. The calculation methods are shown in (1)-(7).

\[V\_ip\_port\_i(t) = \frac{N\_ip\_port\_i(t) - N\_ip\_port\_i(t - 1)}{T}\]  
\[V\_op\_port\_i(t) = \frac{N\_op\_port\_i(t) - N\_op\_port\_i(t - 1)}{T}\]
In the period $T$, the average size $PS_{ib\_port}$ of received packets processed by network port $port_i$ and the average size $PS_{ob\_port}$ of sent packets processed by network port $port_i$ can be calculated according to (8) and (9). Packets that are discarded because the NIC queue is full and error packets should be included in the total number of packets received by the NIC. The receiving load factor of the NIC is calculated as (10), where $V_{imb\_port}$ is the number of bytes that are discarded because the queue is full, $V_{iep\_port}$ is the number of bytes discarded for reception errors. Packets that are sent incorrectly should also be included in the total number of packets sent by the network card. The calculation method of the load rate of the network card is (14), where $V_{oeb\_port}$ is the number of bytes dropped due to the transmission error.

\[
\begin{align*}
V_{ib\_port}(t) &= \frac{N_{ib\_port}(t) - N_{ib\_port}(t-1)}{T} \\
V_{ob\_port}(t) &= \frac{N_{ob\_port}(t) - N_{ob\_port}(t-1)}{T} \\
V_{imp\_port}(t) &= \frac{N_{imp\_port}(t) - N_{imp\_port}(t-1)}{T} \\
V_{iep\_port}(t) &= \frac{N_{iep\_port}(t) - N_{iep\_port}(t-1)}{T} \\
V_{oep\_port}(t) &= \frac{N_{oep\_port}(t) - N_{oep\_port}(t-1)}{T}
\end{align*}
\]

\[
\begin{align*}
\overline{PS}_{ib\_port}(t) &= \frac{V_{ib\_port}(t) \times T}{V_{ip\_port}(t) \times T} = \frac{V_{ib\_port}(t)}{V_{ip\_port}(t)} \\
\overline{PS}_{ob\_port}(t) &= \frac{V_{ob\_port}(t) \times T}{V_{op\_port}(t) \times T} = \frac{V_{ob\_port}(t)}{V_{op\_port}(t)} \\
Li_{port}(t) &= \frac{V_{ib\_port}(t) + V_{imb\_port}(t) + V_{iep\_port}(t)}{V_{max\_b\_port}} \times 100\% \\
V_{imb\_port}(t) &= \frac{V_{imb\_port}(t) \times \overline{PS}_{ib\_port}(t)}{V_{ib\_port}(t)} \\
V_{iep\_port}(t) &= \frac{V_{iep\_port}(t) \times \overline{PS}_{ib\_port}(t)}{V_{iep\_port}(t)} \\
V_{oep\_port}(t) &= \frac{V_{oep\_port}(t) \times \overline{PS}_{ob\_port}(t)}{V_{oep\_port}(t)} \\
Lo_{port}(t) &= \frac{V_{ob\_port}(t) + V_{oep\_port}(t)}{V_{max\_b\_port}} \times 100\%
\end{align*}
\]

\[
Li^*_\text{port}(t) = r \times Li_{\text{port}}(t) + (1-r) \times Li_{\text{port}}(t-1)
\]

\[
Lo^*_\text{port}(t) = r \times Lo_{\text{port}}(t) + (1-r) \times Lo_{\text{port}}(t-1)
\]

To measure whether the current network port load rate is too high compared to other network port load rates, the load balancing degree of network port is proposed. The load balancing degree of network port can reflect the difference in load rate among network ports, and has reference value for whether to call the adjustment function. The receiving load balancing degree and sending load balancing degree of network port are defined by (19) and (20). For the selection of the sending network port, when the sending load balancing degree of the network port exceeds the user-defined load balancing degree threshold $\delta$, the adjustment function is called to select a suitable sending network port again; for the selection of the receiving network port, when the receiving load balancing degree of the network port exceeds the user. When the load balancing degree threshold $\delta$ is defined, the adjustment function is called to select a proper receiving network port again. Finally, the best network port information quad Quad (s_port, s_mac, d_port, d_mac) is returned.

\[
\overline{Li^*}_{\text{port}}(t) = \frac{1}{m} \sum_{i=1}^{m} Li^*_\text{port}(t)
\]

\[
\overline{Lo^*}_{\text{port}}(t) = \frac{1}{m} \sum_{i=1}^{m} Lo^*_\text{port}(t)
\]

\[
Wi_{\text{port}}(t) = \frac{Li^*_\text{port}(t) - \overline{Li^*}_{\text{port}}(t)}{\overline{Li^*}_{\text{port}}(t)} \times 100\%
\]

\[
Wo_{\text{port}}(t) = \frac{Lo^*_\text{port}(t) - \overline{Lo^*}_{\text{port}}(t)}{\overline{Lo^*}_{\text{port}}(t)} \times 100\%
\]

### C. ENERGY CONSUMPTION CONTROL

Since the polling work of the DPDK is continuous, the CPU core utilization can reach 100%. When there is no data to send or receive, this will seriously consume CPU resources and cause unnecessary energy consumption. Although DPDK itself provides power management functions, including the user’s dynamic adjustment of the CPU frequency and the CPU core’s hibernation at different depths based on adaptive algorithms, it is not suitable for the research scenario in this paper. Reducing the CPU frequency on the GPU server will affect the work of other service processes. Moreover, the adaptive sleep algorithm cannot cope with sudden traffic conditions. Therefore, for this application scenario, we designed a unique energy control module.

1) **LCore Model**

In the energy consumption control module, we divide the CPU core into a master core and multiple slave cores, as shown in Fig. 8. The master core is responsible for scheduling the slave core to work or sleep. The core’s working state is divided into transmit and receive mode, receive mode and transmit mode, and the sleep states are divided into long sleep, short sleep and user-defined sleep.
Throughout the entire operating state, we divide the core state into three parts: a work area, a light sleep area, and a deep sleep area, as shown in Fig. 9. Sleep time at the core of the shallow sleep zone is in milliseconds, and sleep time at the core of the deep sleep zone is in seconds. At the same time, a core can only exist in one area. The master core always exists in the work area, and is in the receiving and sending mode, and monitors the traffic status in real-time, and schedules the slave core. At any time when all cores are not fully operational, there is at least one slave core in the shallow sleep zone. When the master core changes the state of the slave core, a slave core can only sequentially switch from the work area to the light sleep area, and then to the deep sleep area, or from the deep sleep area to the light sleep area, and then to the work area. The conversion process is unable to directly switch from the work area to the deep sleep area or from the deep sleep area to the work area. This is to avoid the wake-up delay from the core and better cope with traffic bursts. The core in the shallow sleep zone will be automatically scheduled to the deep sleep zone if it exceeds a certain time. The core switching among other sleep zones requires the master core to schedule it.

2) LCORE SCHEDULING ALGORITHM

This section builds an energy consumption control model based on the dynamic parameters of each network card obtained previously and monitors the flow parameters of each logical core to dynamically and adaptively schedule the logical core work. Each symbol is described in detail in Table 2.

In the whole logic core scheduling process, since the working state of the logic core can only be in one of three regions, (21) always exists. The changes in sent and received traffic play a decisive role in logical core scheduling, so this paper models the sending and receiving traffic. For data transmission, the logic core needs to take out the data to be transmitted from the public transmission buffer for transmission, and the load of the public buffer can be calculated by (24). When the load of the public buffer exceeds the user-set $\beta$ value, all cores need to be called for sending.

$$n = k + l + d$$  

(21)

| TABLE 2. Description of Lcore scheduling Algorithm symbols. |
|---------------------------------|---------|
| Symbol                          | Description |
| $n$                             | Total number of lcores |
| $k$                             | Total number of working lcores |
| $l$                             | Total number of light sleeping lcores |
| $d$                             | Total number of deep sleeping lcores |
| $PS_{ib\_port}$                 | The average packet size(byte) of received bytes from all NICs |
| $PS_{ob\_port}$                 | The average packet size(byte) of transmitted bytes from all NICs |
| $N_{total\_psb}$               | The total size of the public send buffer |
| $N_{used\_psb}$                | The used size of the public send buffer |
| $V_{send\_p\_psb}$             | The used growth rate(pps) of the public send buffer |
| $V_{send\_b\_send}$            | The send packet rate(pps) of all users |
| $V_{send\_b\_psb}$             | The send byte rate(bps) of all users |
| $L_{psb}$                      | The load of the public send buffer |
| $\beta$                        | The load threshold of the public send buffer, user setting value |
| $N_{ip\_lcore}$                | Total number of received packets from the i-th lcore |
| $N_{op\_lcore}$                | Total number of transmitted packets from the i-th lcore |
| $V_{ip\_lcore}$                | The rate (pps) of received packets from the i-th lcore |
| $V_{op\_lcore}$                | The rate (pps) of transmitted packets from the i-th lcore |
| $V_{ib\_lcore}$                | The rate (bps) of received bytes from all working lcores |
| $V_{ob\_lcore}$                | The rate (bps) of transmitted bytes from all working lcores |
| $V_{ib\_port}$                 | The rate (bps) of successfully received bytes from all NICs |
| $V_{imb\_port}$                | The rate (bps) of dropped bytes from all NICs |
| $V_{iob\_port}$                | The rate (bps) of erroneous received bytes from all NICs |
| $V_{*\_ib\_port}$              | The rate (bps) of received bytes from all NICs |
| $N_{core\_required\_recv}$     | The number of cores required to send |
| $N_{core\_need\_recv}$         | The number of cores needed to receive |

$$PS_{ib\_port}(t) = \frac{1}{m} \sum_{i=1}^{m} PS_{ib\_port}(t)$$  

(22)

$$PS_{ob\_port}(t) = \frac{1}{m} \sum_{i=1}^{m} PS_{ob\_port}(t)$$  

(23)

$$L_{psb}(t) = \frac{N_{used\_psb}(t)}{N_{total\_psb}} \times 100\%$$  

(24)

$$V_{used\_p\_psb}(t) = \frac{N_{used\_psb}(t) - N_{used\_psb}(t - 1)}{T}$$  

(25)

$$V_{used\_b\_psb}(t) = V_{used\_p\_psb}(t) \times PS_{ob\_port}(t)$$  

(26)

$$V_{user\_b\_send}(t) = V_{user\_p\_send}(t) \times PS_{ob\_port}(t)$$  

(27)
rate $V_{used\_b\_psb}$ of the public send buffer is greater than zero, indicating that the core needs to be increased for transmission; when $V_{user\_b\_send}$ is equal to $V_{ib\_lcore}$, $V_{used\_b\_psb}$ is equal to zero, indicating the current working core is just able to complete the sending task; when $V_{user\_b\_send}$ is less than $V_{ib\_lcore}$, $V_{used\_b\_psb}$ is less than zero, indicating that the working core has excess sending capacity.

The variable $V_{used\_b\_psb}$ is instructive for evaluating the size of sent traffic. The byte rate received by all working cores is equal to the byte rate successfully received by all network cards, as shown in (33). It is not accurate to estimate the size of the access traffic using the byte rates received by all working cores. We use $V^{*\_ib\_port}$ calculation, which includes not only the successful byte rate of the network card, but also the discarded and incorrectly received byte rate.

$$V_{ip\_lcore}(t) = \frac{N_{ip\_lcore}(t) - N_{ip\_lcore}(t - 1)}{T}$$

$$V_{op\_lcore}(t) = \frac{N_{op\_lcore}(t) - N_{op\_lcore}(t - 1)}{T}$$

$$V_{ip\_lcore}(t) = \sum_{i=1}^{k} V_{ip\_lcore}(t)$$

$$V_{op\_lcore}(t) = \sum_{i=1}^{k} V_{op\_lcore}(t)$$

$$V_{ib\_lcore}(t) = V_{ip\_lcore}(t) \times F_{ib\_port}$$

$$V_{ib\_lcore}(t) = V_{ib\_port}(t)$$

$$V_{ob\_lcore}(t) = V_{op\_lcore}(t) \times F_{ob\_port}$$

$$V_{ib\_port}(t) = \sum_{i=1}^{m} V_{ib\_port}(t)$$

$$V_{imb\_port}(t) = \sum_{i=1}^{m} V_{imb\_port}(t)$$

$$V^{*\_ib\_port}(t) = V_{ib\_port}(t) + V_{imb\_port}(t) + V_{ieb\_port}(t)$$

The maximum transmission and reception rate that a single core can complete is the theoretical limit rate of a single network card. Therefore, to send current-scale traffic, the number of cores $N_{core\_need\_send}$ needs to be increased is rounded up by (39), where $V_{max\_b\_port}$ is the theoretical limit of a single network card speed. The number of cores $C_{N_{core\_required\_recv}}$ required to receive current-scale traffic is calculated by rounding up from (40), and the number of cores to be increased $N_{core\_need\_recv}$ is calculated from (41), where $k$ is the number of cores that have been working.

$$N_{core\_required\_recv}(t) = \left[ V_{used\_b\_psb}(t) / V_{max\_b\_port} \right]$$

$$N_{core\_need\_recv} = N_{lcore\_need\_recv} - k$$

According to the positive and negative values of $N_{core\_need\_recv}$ and $N_{core\_need\_recv}$, it is judged whether the sending data and the receiving data are the wake-up core or the sleep core. Wake-up or hibernate cores can be comprehensively determined according to the transmitted data and received data, and the number of cores to be awakened or hibernated, as well as the number of cores that are awakened or hibernated, such as (44), as shown at the bottom of the next page.

When waking up the cores, the number of cores to be woken up $N_{core\_need\_wakeup}$ exceeds the number of cores in the light sleep area, then all cores $l$ in the light sleep area are woken up to work, and then the max $\{1, N_{core\_need\_wakeup} - l\}$ cores in the deep sleep area are scheduled to enter the light sleep area. For the working mode of waking up the core, if both sending and receiving data need to wake up the core, the maximum number of cores is used to wake up, and the core is in the sending and receiving mode; if one of the sending and receiving data needs to wake up the core, it wakes up the required number of cores, and the cores are in their corresponding modes; if the data transmission and data reception are both sleeping cores, the smallest number of cores is selected for sleeping, and the sleeping cores enter the shallow sleep area. If there are not enough cores for the main core to wake up, that is, most of the cores are in the working area, all working cores need to be switched to the transmit and receive mode.

D. RATE ADJUSTMENT

When multi-user applications send data, to ensure the fairness of bandwidth, it is necessary to limit the transmission bandwidth of each user application. Moreover, to reduce failed transmissions, the receiving capability of the receiver must also be considered. When the user application sends data, it can dynamically adjust the real-time sending rate according to the calculated value of the rate adjustment module.

1) RATE ADJUSTMENT MODEL

In a BPCM cluster, the master node periodically synchronizes real-time information such as the number of current user applications monitored by each node, and the current transmission rate of each network port to all nodes in the cluster. Each node in the cluster receives the synchronization information and sends it to the main core for processing through a private channel. The main core extracts synchronization information from the private pipeline, parses related fields and updates them to the local cache table, and at the same time shares the related information of the rate adjustment module to the global monitoring information table for use by each user application. When each user application sends
data, it can obtain the current user application number of the node and the receiving node, and the current transmission rate of each network port from the global monitoring information table, and calculate the sending rate of the application according to the rate adjustment algorithm. As shown in Fig. 10.

2) RATE ADJUSTMENT ALGORITHM

This section establishes a user application rate adjustment model based on the dynamic parameters of each network card of the sending server and receiving server. The symbols are described in detail in Table 3.

Through the global monitoring information, we can obtain the maximum theoretical total bandwidth \( V_{\text{max}} \_ib \_port \) and available total bandwidth \( V_{\text{available}} \_ib \_port \) received by all network cards of the sending node and the receiving node, and the maximum theoretical total bandwidth \( V_{\text{max}} \_ob \_port \) and total available bandwidth \( V_{\text{available}} \_ob \_port \) that can be sent, the number of applications of the receiving node \( N_{\text{app}} \_recv \) and the number of applications of the sending node \( N_{\text{app}} \_send \). For the maximum sending rate supported by the sending node user sending application, as shown in (51), we choose the maximum value between the average sending bandwidth and the available sending bandwidth; for the maximum receiving rate supported by the user receiving application at receiving node, as shown in (52), we choose the maximum value between the average receiving bandwidth and the available receiving bandwidth. The final sending bandwidth \( V_{\text{max}} \_b \_send \) of the sending application selects the minimum of the maximum sending rate \( V_{\text{max}} \_b \_send \) supported by the user sending application and the maximum receiving rate \( V_{\text{max}} \_b \_recv \) supported by the user receiving application at receiving node, as shown in (53). Finally, we can get the sending rate \( V_{\text{max}} \_p \_send \) of the current user sending application according to (54). Equations (51) and (52) reflect our idea of ensuring that each user uses the bandwidth fairly while making full use of it. Equation (53) can minimize the possibility of transmission on the premise of making full use of the bandwidth.

\[
V_{\text{available}} \_ib \_port(t) = V_{\text{max}} \_ib \_port_i - V \_ib \_port_i(t)
\]

\[
N_{\text{core}} \_need \_send \geq 0 \quad \text{send} \_wakeup(N_{\text{core}} \_need \_send)
\]

\[
N_{\text{core}} \_need \_send < 0 \quad \text{send} \_sleep(N_{\text{core}} \_need \_send)
\]

\[
N_{\text{core}} \_need \_recv \geq 0 \quad \text{recv} \_wakeup(N_{\text{core}} \_need \_recv)
\]

\[
N_{\text{core}} \_need \_recv < 0 \quad \text{recv} \_sleep(N_{\text{core}} \_need \_recv)
\]

\[
\begin{align*}
&[\text{send} \_wakeup(N_{\text{core}} \_need \_send), \text{recv} \_wakeup(N_{\text{core}} \_need \_recv)] \quad \text{wakeup} \text{max} \left\{ N_{\text{core}} \_need \_send, N_{\text{core}} \_need \_recv \right\} \\
&[\text{send} \_wakeup(N_{\text{core}} \_need \_send), \text{recv} \_sleep(N_{\text{core}} \_need \_recv)] \quad \text{wakeup}(N_{\text{core}} \_need \_send) \\
&[\text{send} \_sleep(N_{\text{core}} \_need \_send), \text{recv} \_wakeup(N_{\text{core}} \_need \_recv)] \quad \text{wakeup}(N_{\text{core}} \_need \_recv) \\
&[\text{send} \_sleep(N_{\text{core}} \_need \_send), \text{recv} \_sleep(N_{\text{core}} \_need \_recv)] \quad \text{sleep} \text{min} \left\{ N_{\text{core}} \_need \_send, N_{\text{core}} \_need \_recv \right\}
\end{align*}
\]
TABLE 4. Server configuration information.

| Items     | Type                  | Quantity |
|-----------|-----------------------|----------|
| CPU       | Intel(R) Xeon(R) E3-1230 V2@3.3Ghz | 8-core   |
| Memory    | DDR3-1600MHz          | 32G      |
| System    | CentOS7.0             | -        |
| OS Kernel | Linux version 3.10.0-862.14.4.el7.x86_64 | -        |
| NIC       | I350-T4-4-port 1Gbps each port PCI-e X4 | 8-port(2*4) |
| DPDK      | DPDK-17.11.3          | -        |

\[ V_{\text{available}_ob\_port}(t) = V_{\text{max}_ob\_port} - V_{ob\_port}(t) \]  
\[ V_{\text{available}_ib\_port}(t) = \sum_{i=1}^{m} V_{\text{available}_ib\_port}(t) \]  
\[ V_{\text{available}_ob\_port}(t) = \sum_{i=1}^{m} V_{\text{available}_ob\_port}(t) \]  
\[ V_{\text{max}_ib\_port} = \sum_{i=1}^{m} V_{\text{max}_ib\_port} \]  
\[ V_{\text{max}_ob\_port} = \sum_{i=1}^{m} V_{\text{max}_ob\_port} \]  
\[ V_{\text{max}_b\_send}(t) = \max \left\{ \frac{\text{send\_node}(V_{\text{max}_ob\_port})}{N_{\text{app\_send}(t)}}, \frac{\text{send\_node}(V_{\text{available}_ob\_port}(t))}{N_{\text{app\_send}(t)}} \right\} \]  
\[ V_{\text{max}_b\_recv}(t) = \max \left\{ \frac{\text{recv\_node}(V_{\text{max}_ib\_port})}{N_{\text{app\_recv}(t)}}, \frac{\text{recv\_node}(V_{\text{available}_ib\_port}(t))}{N_{\text{app\_recv}(t)}} \right\} \]  
\[ V_{\text{max}_b\_send}(t) = \min \left\{ V_{\text{max}_b\_send}(t), V_{\text{max}_b\_recv}(t) \right\} \]  
\[ V_{\text{max}_p\_send}(t) = \frac{V_{\text{max}_b\_send}(t)}{PS_{b\_send}} \]

V. EVALUATION

Below we have experimentally verified the algorithm of each module of BPCM. We use 6 servers and a 48-port Gigabit Layer 2 switch to build a BPCM cluster. The server configuration details are shown in Table 4. Each server CPU is an 8-core Intel Xeon E3-1230 V2@3.3Ghz, two four-port Intel-I350-T4 gigabit network cards, and a total of 8 network ports. The BPCM system is developed based on the DPDK-17.11.3 version. It is configured with 8G huge page memory, one port receiving queue and one sending queue, and the size is 1024, which can be adjusted.

A. TRANSMISSION PERFORMANCE EVALUATION

To verify the transmission performance of BPCM, we designed several sets of experiments. Because this paper only studies the performance of BPCM based on unreliable data packet transmission, traditional UDP transmission is selected for comparison experiments. We discussed the transmission performance of UDP and BPCM in the case of a single network card and four network cards. By sending 5GB of data multiple times, and selecting the data packet size as 64B, 128B, 256B, 512B, 1024B, the transmission rate, packet loss rate, and transmission delay are counted.

Fig. 11 shows the statistics results of UDP and BPCM data transmission between the sending server and the receiving server when using a Gigabit network card. It can be seen from Fig. 11 (a) that in the case of a single network card, BPCM benefits from the DPDK user mode driving feature, which reduces the processing time of the traditional kernel protocol stack. When the data packet is small, the transmission rate is significantly better than UDP. The reduction of processing time also causes BPCM to be significantly lower in packet loss rate than UDP. Fig. 11 (b) shows that the BPCM transmission delay is lower than UDP.

The Linux system is not friendly to the parallel transmission of multiple network cards. Although a bonding module with multiple network cards is provided in the kernel, this module is mainly used for network load balancing and redundancy. To increase bandwidth, the switch also needs to support (802.3ad)IEEE 802.3ad Dynamic link aggregation. Although the bonding module can increase the total bandwidth, it cannot break the bandwidth limit of a single connection (the theoretical rate of a Gigabit network card). To achieve the total bandwidth transmission rate, the cooperation of multi-connection and multi-threading technology is also required. To compare the performance of parallel transmission of multiple network cards, we bind the four network cards of the sending server and the receiving server to mode four through the bonding module of the Linux system, and enable the LACP function of the switch in the LAN, and establish four connections through four threads for UDP data transmission. Compare the transmission performance of the UDP single connection and multiple connections with BPCM, as shown in Fig. 12.

In Fig. 12 (a) single connection transmission test, it can be seen that UDP in bonding mode cannot break the limit of the gigabit rate of a single connection network card, and BPCM single connection can make full use of the total bandwidth of four network cards. Due to the influence of the underlying bonding mode in UDP in bonding mode, the packet loss rate increases significantly. Compared with the BPCM of a single network card, the packet loss rate of the four network cards BPCMs increases slightly, but it is significantly lower than the UDP packet loss rate in the bonding mode. Fig. 12 (b) shows that although UDP in bonding mode can make full use of the bandwidth of four network cards through multithreading and multiple connections, it is still lower than the transmission rate of BPCM, and the packet loss rate has increased significantly. In the case of BPCM with multiple connections, the packet loss rate does not change much.

Through the above experiments, we can see that no matter whether there is a single network card or multiple network
cards, BPCM is superior to traditional UDP in terms of transmission performance. Due to the tedious configuration of the bonding module of the Linux system and the bandwidth limitation of a single connection, the utilization of multihybrid cards is not high. The programming model of multithread and multi-connection requires high technical staff. BPCM can double the bandwidth in a single connection, making it easier to use.

B. PORT LOAD BALANCING PERFORMANCE EVALUATION

This section verifies the performance of the port load balancing algorithm proposed in this paper and compares it with a random algorithm. In the case where the number of network ports is equal or not, 5GB of data is sent multiple times, and the packet sizes are selected to be 64B, 128B, 256B, 512B, and 1024B. The load rate and packet loss rate of each network port of the sending server and receiving server were calculated.

Fig. 13 shows the load of each network card with different network port selection algorithms when the sending server and the receiving server use four Gigabit network cards for transmission at the same time. As shown in Fig. 13 (a), the load balancing network port selection algorithm in the sending server makes the load rate of each network card more uniform than the random network port selection algorithm. The load rate of each network port is almost the same. The receiving server in Fig. 13 (b) shows the same result. The load-balanced network port selection algorithm has a more uniform load rate on the sending server than on the receiving server. This is because there is a certain delay in synchronizing the load information of each network port to the sending server. As a result, the sending server has an error in selecting the receiving port.

Fig. 14 shows the load of each network card with different network port selection algorithms when the number of network cards of the sending server and the receiving server are not equal. The sending server uses four Gigabit network cards, and the receiving server uses two Gigabit network cards. Fig. 14 (a) and Fig. 14 (b) also show that the load balancing network port selection algorithm in the sending server makes the load rate of each network card more uniform than the random network port selection algorithm. The load rate of the network port is almost the same. The receiving server in Fig. 13 (b) shows the same result. The load-balanced network port selection algorithm has a more uniform load rate on the sending server than on the receiving server. This is because there is a certain delay in synchronizing the load information of each network port on the receiving server to the sending server. As a result, the sending server has an error in selecting the receiving port.

Fig. 14 shows the load of each network card with different network port selection algorithms when the number of network cards of the sending server and the receiving server are not equal. The sending server uses four Gigabit network cards, and the receiving server uses two Gigabit network cards. Fig. 14 (a) and Fig. 14 (b) also show that the load balancing network port selection algorithm in the sending server makes the load rate of each network card more uniform than the random network port selection algorithm. The load rate of the network port is almost the same. The receiving server in Fig. 13 (b) shows the same result. The load-balanced network port selection algorithm has a more uniform load rate on the sending server than on the receiving server. This is because there is a certain delay in synchronizing the load information of each network port on the receiving server to the sending server. As a result, the sending server has an error in selecting the receiving port.

Fig. 14 shows the load of each network card with different network port selection algorithms when the number of network cards of the sending server and the receiving server are not equal. The sending server uses four Gigabit network cards, and the receiving server uses two Gigabit network cards. Fig. 14 (a) and Fig. 14 (b) also show that the load balancing network port selection algorithm in the sending server makes the load rate of each network card more uniform than the random network port selection algorithm. Due to the bandwidth limitation of the two network cards of the receiving server, the load rate of the network port of the sending server is also limited. This is due to the effect of the rate adjustment module of the BPCM.

Similarly, in the case where the number of network ports of the sending server and the receiving server is equal or
unequal, we have also calculated the packet loss rate of the two network port selection algorithms with different packet sizes. As shown in Fig. 15, in both cases, the packet loss rate of the random network port selection algorithm is higher than that of the load-balanced network port selection algorithm, and the packet loss rate of the load balancing network port selection algorithm does not change much in the two cases.

C. ENERGY CONSUMPTION EVALUATION

This section experimentally verifies the effectiveness of the logical core scheduling algorithm of the BPCM energy control module. Tests were repeated with a total of 5GB and 64B packets. In the process of sending data, the CPU utilization of the sending server and the receiving server and the working state of each logical core will be recorded at each time. In the case where the number of network ports of the sending server and the receiving server is equal or unequal, discussions are made separately.

In Fig. 16, the number of network ports on the sending server and the receiving server is the same. Both are four Gigabit network ports. The CPU utilization on the sending server and the receiving server is always consistent with
the changing trend of the number of working cores. The light sleep area always maintains at least one core number. When the sending server starts to send data, there will be jitter in the number of working cores. This is because the user sends data into the common sending buffer, which is a high-speed memory operation, which will affect the core scheduling.

In Fig. 17, the sending server is four Gigabit Ethernet ports, and the receiving server is two Gigabit Ethernet ports. The overall change trend of the parameters of the sending server and the receiving server is consistent with Fig. 16. However, due to the limitation of the receiving server’s receiving bandwidth, the number of working cores of the sending server and the receiving server both decreased. This is because the BPCM rate adjustment module limits the sending rate of the sending server.

D. RATE ADJUSTMENT EVALUATION

The BPCM rate adjustment module is designed to ensure bandwidth fairness and bandwidth matching for multi-user applications. In the experiment, we used five user applications to send 5GB of data sequentially, with a packet size of 64B. When the number of network ports of the sending server and the receiving server is equal or unequal, the sending bandwidth of each user application at each time is recorded. Finally, the user application bandwidth is compared with the BPCM data transmission without the rate adjustment module.

Fig. 18 shows the change in user application bandwidth when the number of server network ports is the same or different under the role of the rate adjustment module. With the same number of network ports on the server, the total bandwidth is almost fully utilized, and each user application distributes the bandwidth evenly, as shown in Fig. 18 (a). When the number of network ports of the server is different, the total bandwidth of the sending server is limited by the total bandwidth of the receiving server. However, each user application still distributes the bandwidth evenly, as shown in Fig. 18 (b).

When the number of network ports of the server is the same or different, we monitor the final allocated bandwidth and packet loss rate of each user application without using the rate adjustment module or using the rate adjustment module, as shown in Fig. 19 and Fig. 20. In both cases, users who do not use the rate adjustment module have extremely uneven application bandwidth allocation, and the packet loss rate has
significantly increased, which is more prominent when the number of network ports is not equal.

E. ANALYSIS OF COMMUNICATION RELIABILITY OF BPCM

BPCM, as a high-speed parallel communication system, involves not only software design, but also the underlying physical equipment. To achieve reliable and durable high-speed communication of the BPCM system, a lot of work has been done in the design of the BPCM system, which will be described in detail below.

1) HARDWARE RELIABILITY

The underlying physical devices of BPCM include switches, network cards, and network cables. On one hand, the reliability of the physical device itself is one of the determinants of reliable communication. For physical devices, indicators that measure their reliability include MTBF (Mean Time Between Failure), MTTR (Mean Time To Repair), and MTTF (Mean Time To Failure), where MTTF = MTBF + MTTR. For today’s electronic products, under normal working conditions, its MTTF value can reach 10 years. On the other hand, achieving reliability through multiple devices is another way. As described in Section 4.1.3, BPCM uses multiple network cards, multiple switches, and multiple lines to achieve connectivity reliability on the physical lines among nodes. Even if a device or line fails, nodes can still communicate through other channels. High reliability at the physical level is the basis of BPCM’s high reliability.
2) SOFTWARE RELIABILITY
Software level reliability is also critical for BPCM systems. Firstly, the master-slave architecture of the BPCM cluster enables the master node to monitor the connectivity of the network cards and links of each node in the cluster through the heartbeat mechanism, and synchronize to each node in real-time to ensure that each node can quickly shield faulty network cards and links (This part of the work is done by other team members). Secondly, the port load balancing module can avoid overloading each network port and link or failing. Finally, reliable transmission protocols are essential. Because BPCM provides a higher transmission rate, the traditional TCP protocol is not suitable. Therefore, it is necessary to redesign the reliable transmission protocol. This part of the work will be introduced in the next article.

VI. CONCLUSION
As a hardware platform of distributed machine learning system, GPU cluster plays a decisive role in the speed of machine learning model training. The ever-expanding scale and parameters of machine learning models place higher demands on the network bandwidth within GPU clusters. In small-scale GPU clusters, existing solutions cannot be applied due to the need for professional equipment support and high costs. To address this issue, this paper first proposes a multi-NIC bypass parallel communication mechanism based on DPDK, making full use of the idle CPU resources of the GPU server to accelerate data transmission. Then it proposes port load balancing algorithm and CPU multi-core scheduling algorithm for multi-NIC and CPU multi-core to ensure load balance of network card and low CPU energy consumption. In addition, rate adjustment algorithms are proposed to ensure fair use of bandwidth in multi-user scenarios. Finally, the experiment shows that BPCM can make full use of idle CPU resources, take advantage of the characteristics of multiple network cards, and achieve a multiplied increase in communication bandwidth. Moreover, the properties of BPCM based on the second layer of the network determine its simple to use and flexible to expand. The high bandwidth, high efficiency and low cost of BPCM show that it has certain application prospects.

Future work will focus on the support of BPCM for 10 Gigabit network cards and the optimization of data transmission efficiency. On this basis, we will design a special reliable transmission protocol and upper layer communication protocol for distributed model training.

REFERENCES
[1] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, “Multi-tenant GPU clusters for deep learning workloads: Analysis and implications,” Microsoft Res., Albuquerque, NM, USA, Tech. Rep. MSR-TR-2018-13, 2018.
[2] A. Sergeev and M. Del Balso, “Horovod: Fast and easy distributed deep learning in TensorFlow,” 2018, arXiv:1802.05799. [Online]. Available: http://arxiv.org/abs/1802.05799
[3] Intel DPDK. (2014). Data Plane Development Kit Project Page. Accessed: Sep. 12, 2016. [Online]. Available: http://www.dpdk.org
[4] W. Zhu, P. Li, B. Luo, H. Xu, and Y. Zhang, “Research and implementation of high performance traffic processing based on intel DPDK,” in Proc. 9th Int. Symp. Parallel Archit. Algorithms Program. (PAAP), Dec. 2018, pp. 62–68.
[5] H. Wippel, “DPDK-based implementation of application-tailored network software on end user nodes,” in Proc. Int. Conf. Workshop Netw. Future (NOF), Dec. 2014, pp. 1–5.
[6] F. Hijazi, B. Kahne, P. Wilson, and O. Khan, “Efficient parallel packet processing using a shared memory many-core processor with hardware support to accelerate communication,” in Proc. IEEE Int. Conf. Netw. (ICN), Aug. 2015, pp. 122–129.
[7] M. Alam and A. K. Varshney, “A new approach of dynamic load balancing scheduling algorithm for homogeneous multiprocessor system,” Int. J. Appl. Evol. Comput., vol. 7, no. 2, pp. 61–75, Apr. 2016.
[8] I. Cerrato, M. Annarumma, and F. Risso, “Supporting fine-grained network functions through intel DPDK,” in Proc. 3rd Eur. Workshop Sofware Defined Netw., Sep. 2014, pp. 1–6.
[9] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and P. E. Xing, “Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters,” in Proc. USENIX Annu. Tech. Conf. (USENIX ATC), 2017, pp. 181–193.
[10] S. Wang, D. Li, Y. Cheng, J. Geng, Y. Wang, S. Wang, S. Xia, and J. Wu, “BML: A high-performance, low-cost gradient synchronization algorithm for DML training,” in Proc. Adv. Neural Inf. Process. Syst. (NIPS) 2019, pp. 4238–4248.
[11] J. Geng, D. Li, Y. Cheng, S. Wang, and J. Li, “HiPS: Hierarchical parameter synchronization in large-scale distributed machine learning,” in Proc. Workshop Netw. Meets AI ML Netw., 2018, pp. 1–7.
[12] S. Shi, X. Chu, and B. Li, “MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms,” in Proc. IEEE INFOCOM Conf. Comput. Commun., Apr. 2019, pp. 172–180.
[13] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, “Highly scalable deep learning training system with mixed-precision: Training ImageNet in four minutes,” in Proc. 2019, arXiv:1807.11205. [Online]. Available: http://arxiv.org/abs/1807.11205
[14] H. Mikami, H. Suganuma, P. U-chupala, Y. Tanaka, and Y. Kageyama, “Massively distributed SGD: ImageNet/ResNet-50 training in a flash,” 2018, arXiv:1811.05233. [Online]. Available: http://arxiv.org/abs/1811.05233
[15] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, “Image classification at supercomputer scale,” 2018, arXiv:1811.06992. [Online]. Available: http://arxiv.org/abs/1811.06992
[16] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, “In-network computation is a dumb idea whose time has come,” in Proc. 16th ACM Workshop Hot Topics Netw., Nov. 2017, pp. 150–156.
[17] L. Luo, J. Nelson, L. Ceze, A. Phanishayee, and A. Krishnamurthy, “Parameter hub: A rack-scale parameter server for distributed deep neural network training,” in Proc. ACM Symp. Cloud Comput., Oct. 2018, pp. 51–54.
[18] B. Yi, J. Xia, L. Chen, and K. Chen, “Towards zero copy Dataflows using RDMA,” in Proc. SIGCOMM Posters Demo, Aug. 2017, pp. 28–30.
[19] P. Patarasuk and X. Yuan, “Bandwidth optimal all-reduce algorithms for clusters of workstations,” J. Parallel Distrib. Comput., vol. 69, no. 2, pp. 117–124, Feb. 2009.
[20] G. Wang, U. Finkler, D. Kong, and H. Hunter, “Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy,” in Proc. sysML 2019, pp. 1–11.
[21] (2017). NVIDIA NVLINK. [Online]. Available: http://www.nvidia.com/object/nvlink.html
[22] (2018). NVIDIA NVSWITCH. [Online]. Available: http://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf
[23] (2018). PCI Express: An Overview of the PCI Express Standard. [Online]. Available: http://www.nvidia.com/object/nvlink.html
[24] (2019). Massively Scale Your Deep Learning Training with NCCL 2.4. [Online]. Available: https://bit.ly/2FwFQ4.
[25] Pieter Noordhuis. (2017). Accelerating Machine Learning for Computer Vision. [Online]. Available: https://github.com/facebookincubator/gloo
[26] G. Wang, S. Venkataraman, A. Phanishayee, J. Thelin, N. Devanur, and I. Stoica, “Blink: Fast and generic collectives for distributed ML,” 2019, arXiv:1910.04940. [Online]. Available: http://arxiv.org/abs/1910.04940

M. Wu et al.: BPCM: A Flexible High-Speed Bypass Parallel Communication Mechanism for GPU Cluster
MINGJIE WU is currently pursuing the Ph.D. degree in system analysis and integration with the University of Shanghai for Science and Technology. His current research interests include high performance parallel computing, cluster communication, edge computing, and so on. He recently works on low-cost and high-speed parallel communication solutions in GPU clusters.

QINGKUI CHEN received the Ph.D. degree. He is currently a Professor and the Doctoral Tutor of computer science with the University of Shanghai for Science and Technology. His current research interests include the Internet of Things, parallel computing, high performance computing, GPU architecture and artificial intelligence, and so on. He is also a member of the Chinese Computer Society. At the same time, he is also a Computer Science Leader and a Reviewer of leading journals and conferences.

MINGJIE WU

JINGJUAN WANG

MINING JUAN WANG is currently pursuing the Ph.D. degree in system analysis and integration with the University of Shanghai for Science and Technology. Her current research interests include spatiotemporal data mining, deep learning and traffic prediction. She recently works on traffic prediction model based on spatiotemporal attention network.

***