Modeling and Simulation 建模与仿真, 2025, 14(5), 1139-1152
Published Online May 2025 in Hans. https://www.hanspub.org/journal/mos
https://doi.org/10.12677/mos.2025.145464

Hans汉斯

基于DPDK的集群内并发数据流传输机制研究

沈非凡，陈庆奎

上海理工大学光电信息与计算机工程学院，上海

收稿日期：2025年4月30日；录用日期：2025年5月23日；发布日期：2025年5月31日

摘　要

针对人工智能和物联网技术发展带来的大规模AI数据流传输需求，文章基于DPDK框架，设计了一种高性能的并发数据流传输机制，以优化集群内节点间的数据通信效率。并提出了高性能传输策略，结合端口分配策略以及双链路聚合调度策略，实现了大数据流的多端口并行传输。通过多网卡绑定、多核分配技术，以及反向流量控制机制，系统能够动态调整传输路径，优化带宽利用率，显著降低传输延迟和丢包率。实验表明，该方案在吞吐量、延迟和丢包率方面相较传统方法具有显著优势，在高并发场景下展现出优异的性能扩展性。本研究为大规模AI数据流的高效传输提供了一种低成本、高效率的解决方案，可为未来大数据和人工智能领域的分布式计算系统设计提供重要参考。

关键词

DPDK，高性能通信，AI数据流，负载均衡，集群

Research on Concurrent Data Stream Transmission Mechanism in Clusters Based on DPDK

Feifan Shen, Qingkui Chen

School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai

Received: Apr. 30th, 2025; accepted: May 23rd, 2025; published: May 31st, 2025

Abstract

To address the demand for large-scale AI data stream transmission driven by the development of artificial intelligence and the Internet of Things, this study designs a high-performance concurrent data streaming mechanism based on the DPDK framework to optimize the data communication

文章引用: 沈非凡, 陈庆奎. 基于 DPDK 的集群内并发数据流传输机制研究[J]. 建模与仿真, 2025, 14(5): 1139-1152.
DOI: 10.12677/mos.2025.145464

efficiency between nodes in the cluster. In this study, a high-performance transmission strategy is proposed, which combines the port allocation strategy and the dual-link aggregation scheduling strategy to realize the multi-port parallel transmission of large data streams. Through multi-NIC binding, multi-core distribution technology, and reverse flow control mechanism, the system can dynamically adjust the transmission path, optimize bandwidth utilization, and significantly reduce transmission delay and packet loss rate. Experiments show that the proposed scheme has significant advantages over traditional methods in terms of throughput, latency, and packet loss rate, and shows excellent performance scalability in high-concurrency scenarios. This paper provides a low-cost and high-efficiency solution for the efficient transmission of large-scale AI data streams, which can provide an important reference for the design of distributed computing systems in the field of big data and artificial intelligence in the future.

Keywords

DPDK, High-Performance Communication, AI Data Stream, Load Balancing, Cluster

1. 引言

随着人工智能和物联网时代的蓬勃发展，网络中的数据流量开始呈现爆发式几何增长。据国际数据公司(International Data Corporation, IDC)的测算，预计到 2025 年，仅中国产生的数据总量将达到 48.6 ZB，占比全球总量的 27.8% [1]。庞大的数据规模给数据处理产生了巨大的负担，但传统的服务器系统对于大规模数据读写请求的到来往往处理得不尽人意[2] [3]。网络技术在不断的发展，网卡从从前的 1G 到现在的 100 G，中央处理器(Central Processing Unit, CPU)从单核到多核到如今的多 CPU。虽然服务器的单机能力在不断的增强，然而数据处理能力却不能和硬件门当户对。利用计算机中有限的资源去提高网络中的数据处理能力和数据传输速度成为了当今学术界大数据处理的热点。

传统的报文传输都要采用硬中断来做通信，每次硬中断大约需要 100 微秒，传输过程操作系统必须在内核态和用户态之间来回切换拷贝，产生大量的 CPU 消耗，期间还会产生 Cache Miss，导致需要回内存再次读取数据，产成大量的计算资源浪费[4]。为了应对报文传输通信过程中的资源消耗问题，Intel 公司设计开发出了数据平面开发套件(Data Plane Development Kit, DPDK) [5]。DPDK 通过对原本的数据包处理流程进行了优化，绕过了 Linux 内核驱动的协议栈，直接让网卡接管用户态的数据，从而大大减少了传统报文传输过程中内核态和用户态之间切换带来的开销，提高了内存访问效率[6]。

边缘计算成为缓解数据中心计算压力、提升数据处理效率的重要手段。然而，随着智能设备的普及，边缘计算集群中的终端设备(如智能摄像头、自动驾驶传感器、工业监控系统)每天都会产生海量的 AI 数据流[7]，并需要实时传输至计算集群进行分析处理。以智能监控系统为例，高清摄像头每秒可采集上千张图片，用于人脸识别、行为分析等 AI 任务[8]。然而，由于图像内容的复杂度不同(如光照、运动模糊、场景变化等)，即便采集频率固定，每张图片的大小也有所不同，导致 AI 数据流的动态变化。这种数据的不确定性给高并发数据传输与计算调度带来了极大的挑战。

因此，如何高效地组织、传输和处理海量 AI 数据流，成为 AI 计算集群优化的重要研究方向。尤其是在 GPU 计算集群中，AI 计算任务通常采用 SPMD (Single Program Multiple Data)模型[9]进行并行计算，

要求同一时间片内的数据尽可能均衡输入至 GPU，以最大化计算资源利用率[10]。然而，传统的单网卡通信架构在面对高并发、多节点的大规模数据流时，往往难以满足低延迟、高吞吐的需求，导致系统计算效率下降。因此，提高数据流传输效率，充分发挥 GPU 计算能力，优化集群内数据通信架构，成为本研究的核心问题。

针对上述问题，本文提出了一种基于 DPDK 的集群内高并发 AI 数据流传输机制，并设计了一系列优化策略，以提升数据流的传输效率和计算集群的资源利用率。

1) 多端口并行传输：采用 DPDK 的多网卡绑定技术，通过多端口并行传输，增加传输带宽，提高集群内数据流的吞吐能力，减少单一网卡的负载瓶颈。

2) 反向流量控制机制：提出基于丢包率、带宽利用率和端口负载状态的反向流量控制策略，动态调整数据流传输速率，降低丢包率，提高数据传输的稳定性。

3) 高性能传输策略：结合端口带宽聚合策略以及双链路聚合调度策略，实现了大数据流的多端口并行传输，解决了 AI 数据流的传输问题。

2. 系统设计及核心思想

2.1. 相关概念定义

在高并发 AI 数据流的传输和处理过程中，数据的层次化组织是实现高效通信与计算的关键。为了更直观地描述系统的数据流结构和处理逻辑，本文定义了 AI 数据流、时间片数据批和数据批元三个概念，构建起由宏观到微观的逐层细化模型。图 1 展示了 n 个数据流当前数据单元的结构，其中每个数据流的当前数据单元大小为 $S_i\,(i=1,2,\ldots n)$，单位为字节(Bytes)。则该时间片内所有数据流的总通讯容量需求 M 为：

$$M=\sum_{i=1}^{n}S_i$$

假设每个数据单元的传输必须在处理周期 T 内完成(单位为秒)，则所需的总带宽 B_{total} 为：

$$B_{total}=\frac{M}{T}$$

其中，T 的值由系统性能需求决定，例如 $T=1$s 或更小的时间片长度。

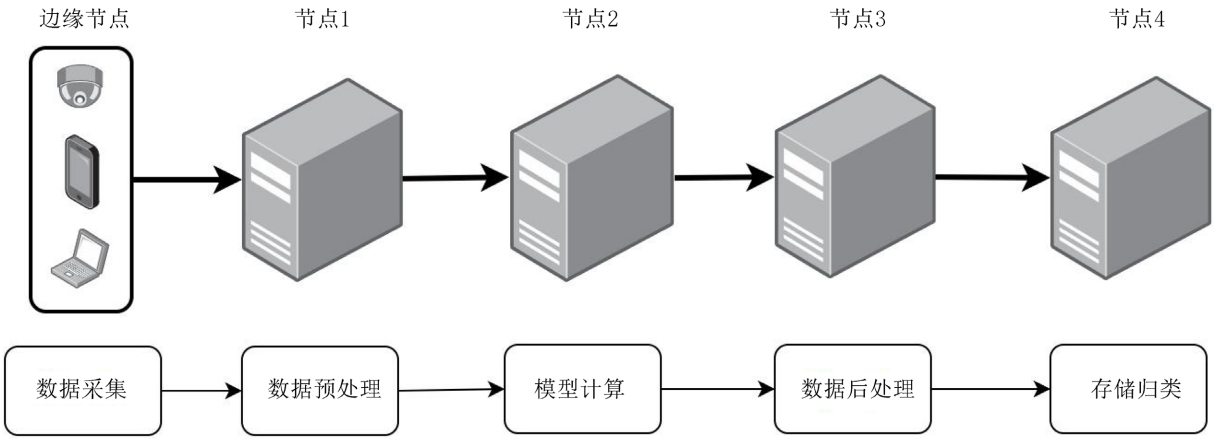

Figure 1. Distributed AI data stream node transmission model
图 1. 分布式 AI 数据流节点传输模型

2.1.1. AI 数据流

在模型计算推演过程中产生的中间数据被称为 AI 数据流，在人工智能与物联网的深度融合背景下，智能终端设备(如摄像头、无人机、传感器等)持续产生大量的 AI 数据流。为了提高并发 AI 数据流处理的效率并减轻单个计算节点的压力，采用了流水线式的分布式处理架构。计算集群由多个计算节点(服务器)组成，每个节点有多个 GPU。数据不会集中在单个节点上，而是被分布式处理，每个计算节点负责特定的任务去对接收到的 AI 数据流进行处理。图 2 给出了一个基于图像识别的简单分布式 AI 数据流的内存架构。

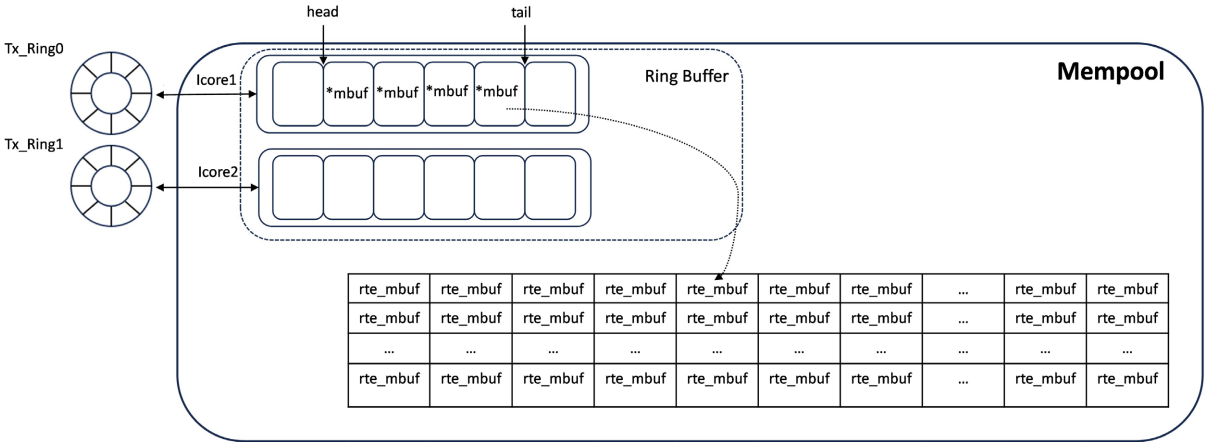

Figure 2. System memory architecture diagram
图 2. 系统内存架构图

2.1.2. 时间片数据批

AI 数据流通常以时间片为单位进行划分，并以数据批次的形式提交给计算集群以提高计算效率，充分利用 GPU 的并行计算能力，实现 SPMD 的计算模式，以减小计算开销。时间片是指在某个固定时间窗口(如 1 s)内收集的数据单位。数据批次是指在当前时间片内采集的一组数据，例如：摄像头 1 秒内采集到的数千张图片可视为一个数据批次。在计算集群内，时间片数据批次的组织方式直接影响系统的吞吐量和计算性能。终端设备在每个时间片内采集 AI 数据流，并通过网络传输至计算集群。各计算节点按照时间片划分数据批次，并将其传输至 GPU 进行计算。GPU 按照 SPMD 模型并行计算数据批次，提高处理速度并确保任务均匀分配。

2.1.3. 数据批元

在 AI 数据流的计算过程中，数据批次通常需要进一步拆分为更细粒度的数据批元，以适应 GPU 计算、网络传输和存储管理的需求。数据批元是数据批次中的最小计算单元，例如在图像识别任务中每张图片可看作一个数据批元，在视频处理任务中，每一帧画面可视为一个数据批元。

2.2. 内存模型

在 DPDK 框架中，内存模型是其高性能数据包处理的核心技术之一。通过对传统数据包处理流程的优化，DPDK 实现了数据从接收到发送的高效传递，降低数据传输时的延迟以及资源开销。本系统基于 DPDK 的内存管理机制，设计了一种高效的内存模型，主要包括 Memory Pool (内存池)、Mbuf (数据包缓冲区)和 Ring Buffer (环形缓冲区)三大组件，结合零拷贝和 NUMA 优化技术，满足高并发 AI 数据流传输的性能需求。基于 DPDK 的多核多网卡集群节点通信模型如图 3 所示，DPDK 在数据传输过

程中将接收到的 AI 数据流分成 Mbuf 块存储在内存池 Memory Pool 中，通过 Ring Buffer 调度存储在 Mempool 中的 AI 数据流。为避免内存泄漏和资源浪费，在数据传输完成后 Mbuf 会归还至内存池进行内存回收。

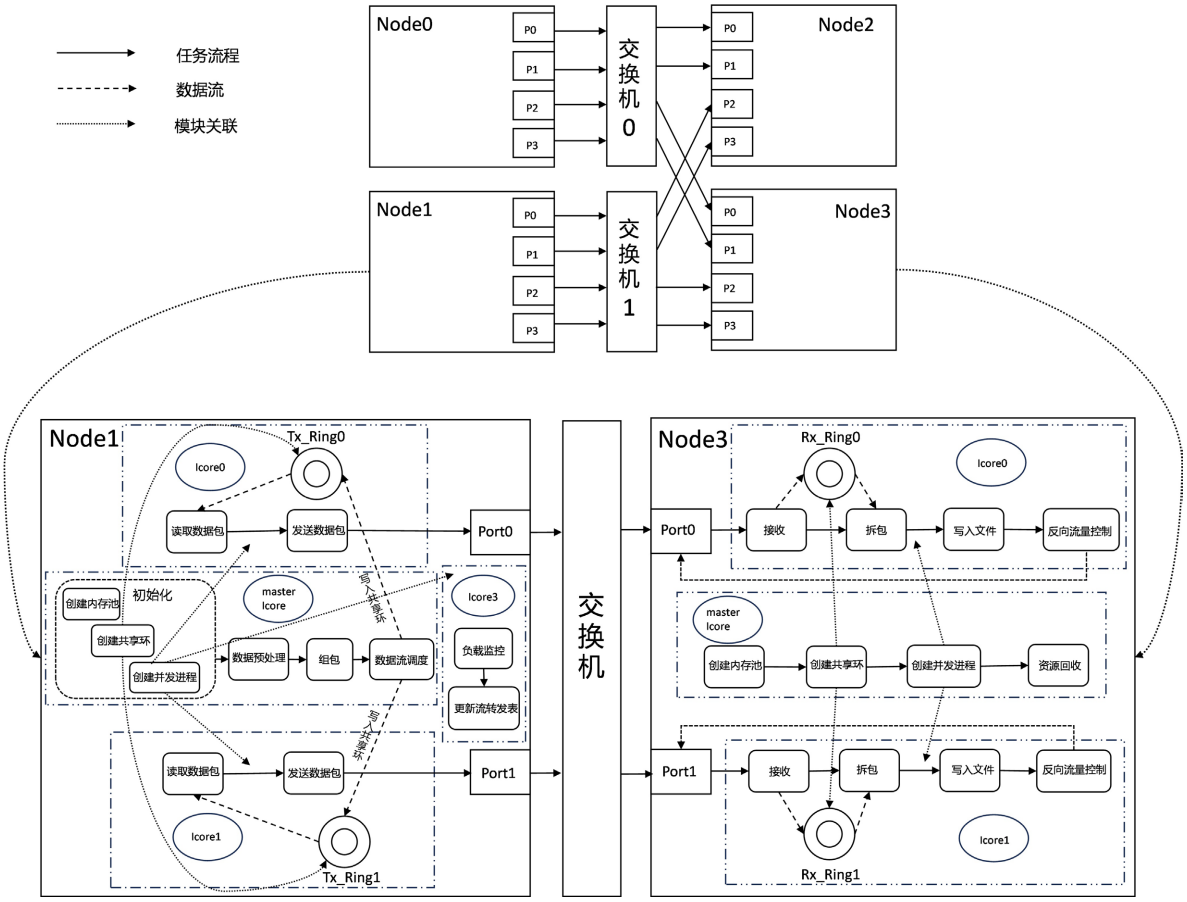

Figure 3. Multi-core and multi-NIC cluster node communication model based on DPDK
图 3. 基于 DPDK 的多核多网卡集群节点通信模型

2.3. 通信模型

本系统的通信架构如图 4 所示，在并发数据流传输中，单一交换机的带宽可能无法满足大规模数据流的传输需求。为了增强网络带宽，系统采用了双交换机架构，通过链路捆绑增加带宽。发送节点和接收节点均连接到两台交换机，通过负载均衡算法将数据流分配到两个网络中，实现带宽聚合和链路冗余提升传输效率并降低单一链路的负载压力。假设两台交换机的总带宽为：

$$B_{switch} = B_1 + B_2$$

其中 B_1 和 B_2 分别表示两个交换机的带宽。本系统通过动态调度算法将数据流分配到两条链路中，设每条链路分配的数据流数量分别为 n_1 和 n_2，则由于交换机带宽限制，需要满足：

$$\begin{cases} \sum_{i=1}^{n_1} s_i / T \le B_1 \\ \sum_{i=1}^{n_2} s_i / T \le B_2 \end{cases}$$

其中 S_i (i = 1, 2, ... n)为每个数据流的当前数据单元大小，T 为传输周期。

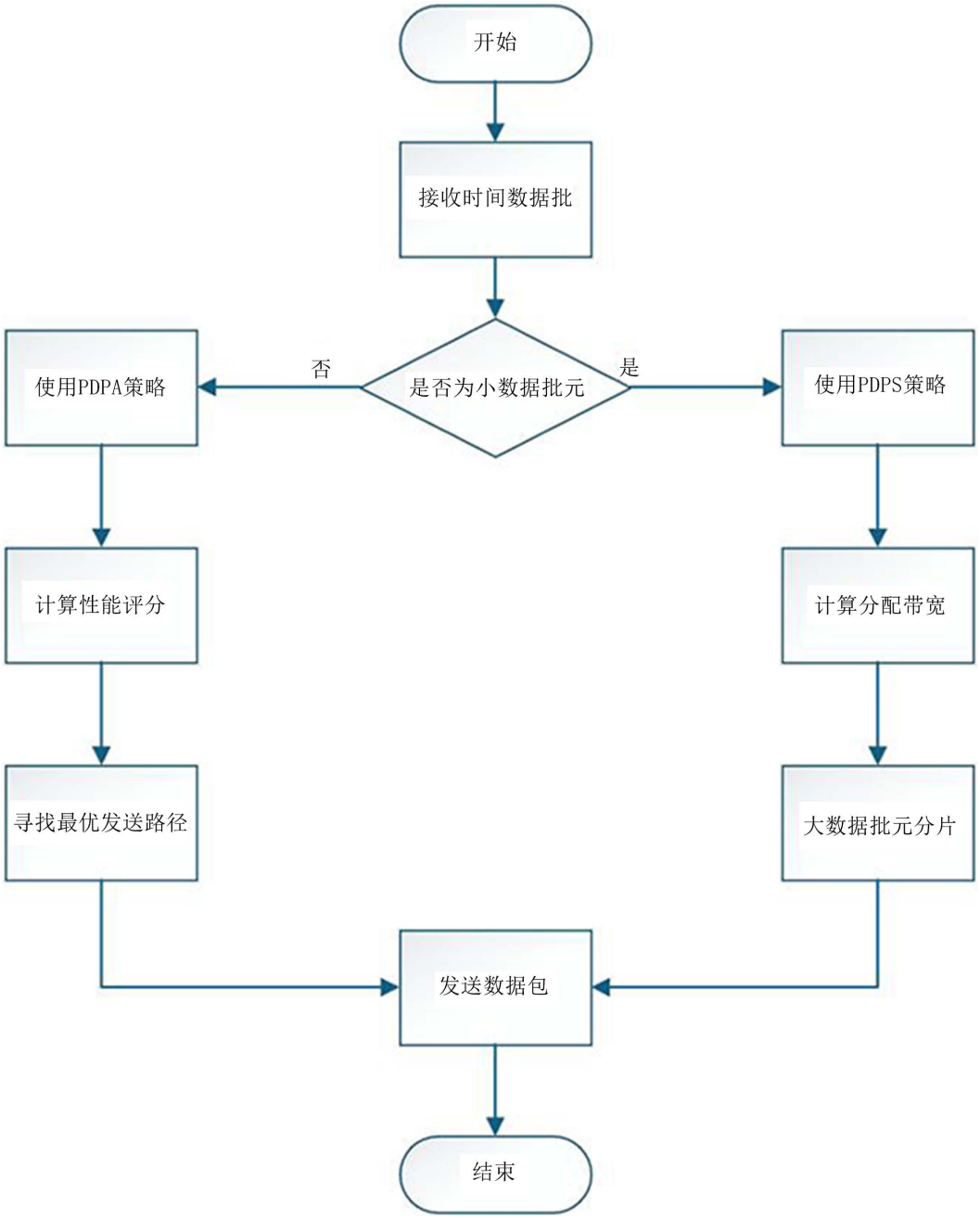

Figure 4. DPPDS scheduling strategy flow chart
图 4. DPPDS 调度策略流程图

3. 系统实现

3.1. 数据帧设计

在网络通信中，数据帧的高效传输和管理是确保系统性能和可靠性的关键。本系统基于 DPDK 框架，设计并实现了一种高效的数据帧结构和内存管理机制。该设计旨在满足高性能、低延迟和高可靠性的需

求，适用于高速网络环境中的实时数据传输。表 1 对此数据帧的组成结构以及关键字段进行了详细的说明。

Table 1. Data frame field table
表 1. 数据帧字段表

字段名称	长度	概述
目的 MAC 地址	6 字节	接收方的 MAC 地址
源 MAC 地址	6 字节	发送方的 MAC 地址
帧标识符	1 字节	数据帧的唯一标识
数据批元编号	4 字节	标识该数据包属于哪个数据批元
序列号	4 字节	数据包在整个数据批元中的顺序编号
时间戳	8 字节	记录数据发送时间，用于计算网络延迟
数据长度	2 字节	数据域的实际长度
结束标识符	1 字节	标识是否为最后一个数据包
数据域	46~1500 字节	实际数据内容
帧校验序列(FCS)	4 字节	验证数据完整性

3.2. 反向流量控制

在集群内并发数据流的高效传输过程中，负载均衡是保证系统性能稳定、资源利用率高、延迟低的重要环节。然而，在高并发数据流传输环境下，网络负载的不均衡可能导致链路拥塞，进一步造成数据丢失、传输延迟增加以及系统吞吐量下降。因此，本系统引入了反向流量控制机制，通过实时监测接收端的负载情况，动态调整数据流的传输速率，以优化传输稳定性、提高带宽利用率以及减少丢包率。

动态负载均衡将接收端充当反向控制端，监控每个发送端的传输状态，并对数据流的实时状态进行统计分析。当某个接收端的负载过高时，控制机制将动态调整数据流的传输速率。系统接收端通过统计接收到的数据包数量并与发送端预期的包数量进行对比，实时计算丢包率，如果发现丢包率超过设定的阈值，则向发送端反馈拥塞信号。发送端根据接收端的反馈信息调整传输策略，通过降低发送速率逐步缓解网络负载，避免完全暂停发送导致的传输中断。经过预设的时间间隔后，发送端逐步恢复发送速率。如果接收端未再次发送拥塞信号，发送端最终恢复到原始发送速率。发送端在暂缓期间，通过增加数据包发送间隔来控制速率。发送间隔公式如下：

$$T_{send\_interval} = T_{base\_interval} * \left(1 + k * R_{drop} * 100\right)$$

其中：

$T_{base\_interval}$ 为基础发送间隔，代表发送端预设发送数据包的间隔；

k 为控制时间间隔增幅的参数；

R_{drop} 为此端口当前时间段的丢包率。

3.3. 端口分配策略

在集群内高并发数据流的传输过程中，端口分配的合理性对系统的传输性能至关重要。通过对端口状态的实时监控，并结合数据预处理操作，本系统提出了一种动态端口分配策略。通过分析当前所有端口实时的关键指标，包括带宽利用率、队列长度、丢包率等，制定相关评分策略，选择性能最优的路径传输数据。本系统各个端口的性能评分公式如下：

$$Score_i = w_1 * \left(1 - R_{bandwidth}(i)\right) + w_2 * \left(1 - R_{buffer}(i)\right) + w_3 * \left(1 - R_{drop}(i)\right)$$

其中：

$Score_i$ 为端口 i 的性能评分结果；

$R_{bandwidth}(i)$为端口 i 的带宽利用率；

$R_{buffer}(i)$为端口 i 的环形队列填充率；

$R_{drop}(i)$为端口 i 的丢包率；

w_1、w_2、w_3 为各项指标的权重，满足 $w_1 + w_2 + w_3 = 1$。

w_1 (带宽利用率权重)：表示传输路径选择中对带宽利用率的关注程度。权重越高，系统越倾向于选择带宽利用率低的端口。

w_2 (环形队列填充率权重)：表示传输路径选择中对端口负载的关注程度。权重越高，系统越倾向于选择负载较低的端口。

w_3 (丢包率权重)：表示传输路径选择中对丢包率的关注程度。权重越高，系统越倾向于选择丢包率较低的端口。

该策略监控核心实时监控各端口的状态，并计算出各端口的性能评分，持续更新路径监控表。发送端则根据性能评分选择当前数据包的发送路径。

3.4. 端口监控模块设计

端口监控模块是系统的核心组成部分，其主要功能是实时采集和更新每个端口的状态信息，存储在端口路径表中，为端口分配决策以及反向流量控制提供支持。以下是端口监控模块的详细设计，包括端口路径表的结构定义、关键状态指标的计算方法，以及模块的整体实现流程。

3.4.1. 端口路径表结构

端口路径表是一个记录每个端口状态的核心数据结构，表中每一条记录对应一个端口。表 2 是端口路径表的字段设计。

Table 2. Port path table fields
表 2. 端口路径表字段

字段名称	数据类型	字段描述
port_id	uint16_t	端口编号，用于唯一标识端口
dest_mac	char (18)	目标 MAC 地址
Icore_id	uint16_t	端口绑定的 CPU 核心编号
tx_rate	float	包转发速率，表示当前端口的发送速率
bandwidth_rate	float	当前端口的带宽利用率
drop_rate	float	当前时间片内端口的丢包率
buffer_occupancy	float	当前端口环形队列缓冲区负载程度
port_score	float	路径评分，用于调度策略选择最优路径，评分越高优先级越高
send_interval	uint32_t	数据包发送间隔，用于反向流量控制，动态调整发送速率
updated_time	uint64_t	最近一次更新的时间戳，记录表更新时间

3.4.2. 关键指标计算方法

1) 包转发速率计算

包转发速率表示每秒通过该端口发送的平均数据包数量，用于评估端口的实际负载，由当前发送数据包数量和上一个时间片记录的发送数据包数量的差值与时间间隔的比值得出。

$$tx\_rate(T)=\frac{N_{sent}(T)-N_{sent}(T-t)}{t}*100\%$$

其中 $tx_rate(T)$表示 T 时刻的包转发速率，单位为 pps，即每秒发送的包数，$N_{sent}(T)$表示 T 时刻已发送数据数量，t 表示与上一次记录数据包数量的时间差值。

2) 带宽利用率计算

带宽利用率表示当前实际使用带宽与理论最大带宽的比值。计算公式如下：

$$bandwidth\_rate(i)=\frac{R_{actual\_data}(T)}{\mathrm{Max}\_bandwidth(i)}*100\%$$

其中 $bandwidth_rate(i)$表示端口 i 的带宽利用率，$\mathrm{Max}_bandwidth(i)$表示该端口的最大理论带宽，其最大理论带宽由网卡的规格参数决定。$Ractual_data(T)$表示 T 时刻实际的发送速率，$Ractual_data(T)$通过 tx_rate 和数据包大小计算。计算公式如下：

$$R_{actual\_data}(T)=tx\_rate(T)*packet\_size*8$$

计算得到的发送速率单位为 bps。

3) 丢包率计算

丢包率表示在当前时间片内，发送端发送的总数据包与接收端未接收到的数据包的比值：

$$drop\_rate=\left(1-\frac{\mathrm{P_{recv}}}{\mathrm{P_{send}}}\right)*100\%$$

其中 $\mathrm{P_{send}}$ 表示发送的数据包数量，$\mathrm{P_{recv}}$ 表示实际接收到的数据包数量。

4) 环形队列负载计算

环形队列负载由该核心绑定的环形队列中未处理的数据包数量与该环形队列总容量的比值得出，表示该端口发送队列的负载程度。

$$buffer\_occupancy(T)=\frac{N_{ring}(T)}{N_{capacity}}*100\%$$

其中 $buffer_occupancy(T)$表示 T 时刻该环形队列的负载率，$N_{ring}(T)$表示 T 时刻队列中未处理数据包的数量，$N_{capacity}$ 表示环形队列的总容量。

4. 实验结果

4.1. 实验环境

为验证本系统的可行性和有效性，实验使用了四台普通主机作为集群节点以及两台千兆以太网交换机进行实机测试，以模拟一个千兆网络集群通信的传输过程。集群的拓扑结构如图 5 所示，其中节点 0 和节点 1 作为发送节点，模拟集群内并发数据流传输的发送端，节点 2 和节点 3 则作为接收节点，模拟集群内数据流的接收端。

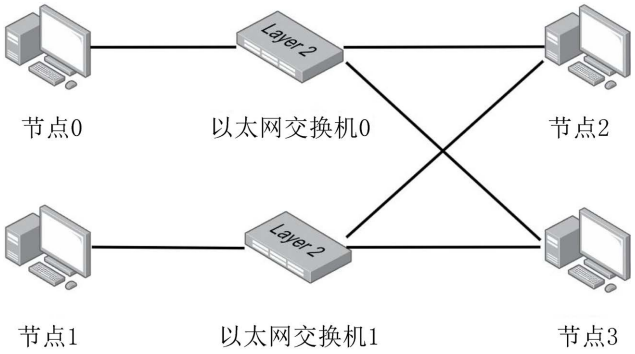

Figure 5. Cluster topology diagram
图 5. 集群拓扑结构

实验为每一个集群节点都配置了一张四端口的多队列网卡，并配备了两台 48 端口的以太网交换机，具体的系统配置信息如表 3 所示。

Table 3. System configuration information
表 3. 系统配置信息

节点 0	
网卡	Intel Ethernet I350-T4 1 GB/s 4 端口
内存	DDR3 1333Mhz 4G × 2
CPU	Intel (R) Core (TM) i7-870 @2.93 GHz
操作系统	Centos 7.0
内核版本	3.10.0-123.el7.x86_64
DPDK 版本	DPDK-stable-17.11.3
节点 1	
网卡	Intel Ethernet I350-T4 1 GB/s 4 端口
内存	DDR3 1333 Mhz 8 G
CPU	Intel (R) Core (TM) i3-2130 @3.40 GHz
操作系统	Centos 7.0
内核版本	3.10.0-123.el7.x86_64
DPDK 版本	DPDK-stable-17.11.3
节点 2	
网卡	Intel Ethernet I350-T4 1GB/s 4 端口
内存	DDR3 1333Mhz 4G × 4
CPU	Intel (R) Xeon (R) E31230 @3.20 GHz
操作系统	Centos 7.0
内核版本	3.10.0-123.el7.x86_64
DPDK 版本	DPDK-stable-17.11.3
节点 3	
网卡	Intel Ethernet I350-T4 1GB/s 4 端口

续表

内存	DDR3 1333 Mhz 4G × 2 + 8G × 2
CPU	Intel (R) Xeon (R) E3-1231 V3 @3.40 GHz
操作系统	Centos 7.0
内核版本	3.10.0-123.el7.x86_64
DPDK 版本	DPDK-stable-17.11.3
交换机	
交换机 0	H3C S5500 Serise (48 个以太网端口)
交换机 1	D-Link DES-1250G (48 个以太网端口)

4.2. 实验分析

4.2.1. 基于 DPDK 的多端口并行传输

为测试基于 DPDK 的多端口并发传输能力，实验模拟时间片数据批的传输过程，将 512 张不同的图片作为 512 条 AI 数据流当前时间片的数据批元进行传输，每张图片约 220 KB 至 310 KB，总数据量约为 137.6 MB。分别进行单网口传输、双网口传输、四网口传输以及将两个发送节点进行绑定实现八网口传输。

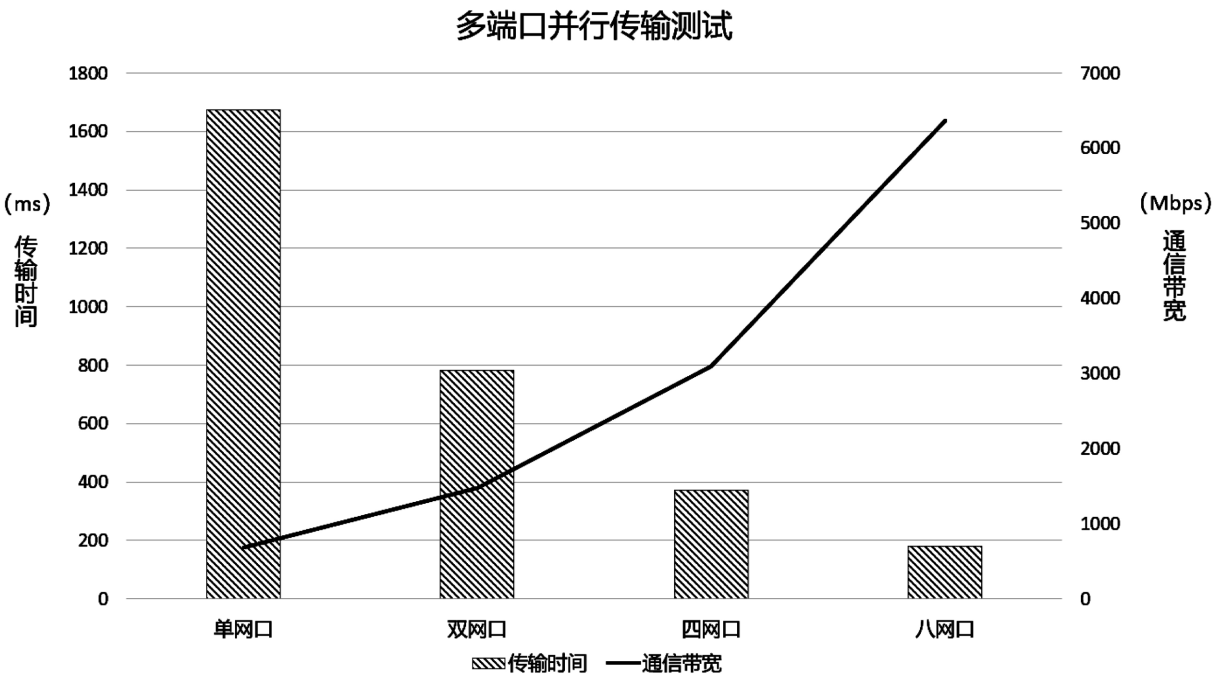

Figure 6. Multi-port parallel transmission test
图 6. 多端口并行传输测试

初始实验中，三项指标权重均设置为相等($w_1 = w_2 = w_3 = 0.33$)，并通过小规模数据流传输测试观察系统性能。在实验中发现，带宽利用率对吞吐量的影响更为显著，因此适当提高了 w_1 的权重；同时，环形队列填充率对传输延迟和端口负载均衡的影响较大，w_2 的权重次之；而丢包率则在高并发情况下才会显著影响性能，w_3 的权重相对较低。通过多次实验调整，最终将权重设置为 $w_1 = 0.57$，$w_2 = 0.31$，$w_3 =$

0.12。这一参数配置在吞吐量、延迟和丢包率三方面达到了较好的平衡，并将反向流量控制的 $T_{base\_interval}$ 设置为 1.5 微秒，增幅参数 k 取 22.5，丢包率阈值设置为 0.2%，实验结果如图 6 所示。

实验表明，随着网口的增加，传输同一时间片数据批所用的处理周期 T 是有显著降低的，同一时间片数据批使用八个网口进行传输的处理周期 T 仅为单网口传输处理周期的 10.84%。其带宽利用率分别为 68.82%、73.62%、77.21%以及 79.47%，由于反向流量控制会降低负载较高端口的发送速率，因此多端口的带宽利用率比单端口更高，以单网口作为基准，带宽加速比分别为 1.00、2.13、4.49、9.23，表明随着网口的增加，系统的通信带宽有近似线性增长的显著提升。

4.2.2. 反向流量控制有效性验证

在千兆网卡的环境下，如果数据包发送速率超过网卡最大吞吐量，则网卡可能会直接丢弃部分数据包，导致数据传输过程的丢包率增加，因此需要主动控制数据包发送时的速率。在实验中，发送端充分利用多口网卡的优势，进行多次实验，利用四个端口并行发送 64 个至 2048 个数据批元，单个数据批元为一张 170 KB 的图片，并分别使用静态控制发送速率和本系统的反向流量控制发送速率两种方式进行测试，静态速率控制的控制参数为 8 微秒，即每个数据包发送后需要 8 微秒的缓冲时间，每个数据包的大小为 1024 字节。实验参数设置如同章节 4.2.1，实验结果如图 7 所示。

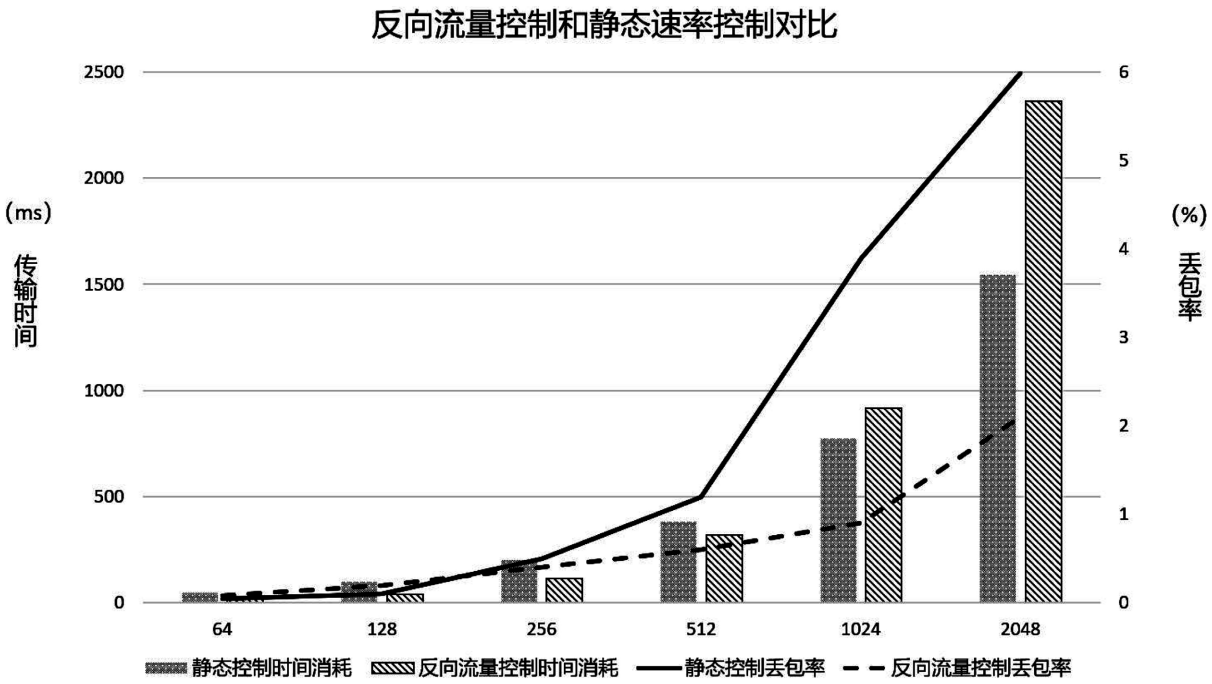

Figure 7. Comparison results of reverse flow control and static rate control
图 7. 反向流量控制和静态速率控制对比结果

实验表明当时间片数据批较少的时候，利用反向流量控制去动态控制发送速率可以减少数据传输的时间，提高传输效率，随着数据规模的增大，时间片数据批的传输周期 T 变大，丢包率增加。反向流量控制在数据量较大的传输情况下由于设置了较低的丢包率阈值，会频繁触发反向流量控制，导致传输时间增加，但是丢包率得到了明显的控制，在传输 2048 个数据批元的过程中，反向流量控制的丢包率为静态控制速率方法丢包率的 21.91%，证明在大数据量的传输情况下，反向流量控制的负载均衡方法可以有效减少数据传输的丢包率。

4.2.3. 双链路聚合测试

为测试双交换机组成的双链路通信网络的传输性能，将两个发送节点和接收节点连接于两台交换机上，形成双链路通信网络，与单交换机传输进行对比。实验使用 8 个传输端口，传输 256 个至 4096 个数据批元，每个数据批元大小为 220 KB 至 310 KB，其余参数设置如同章节 4.2.1，实验结果如图 8 所示。

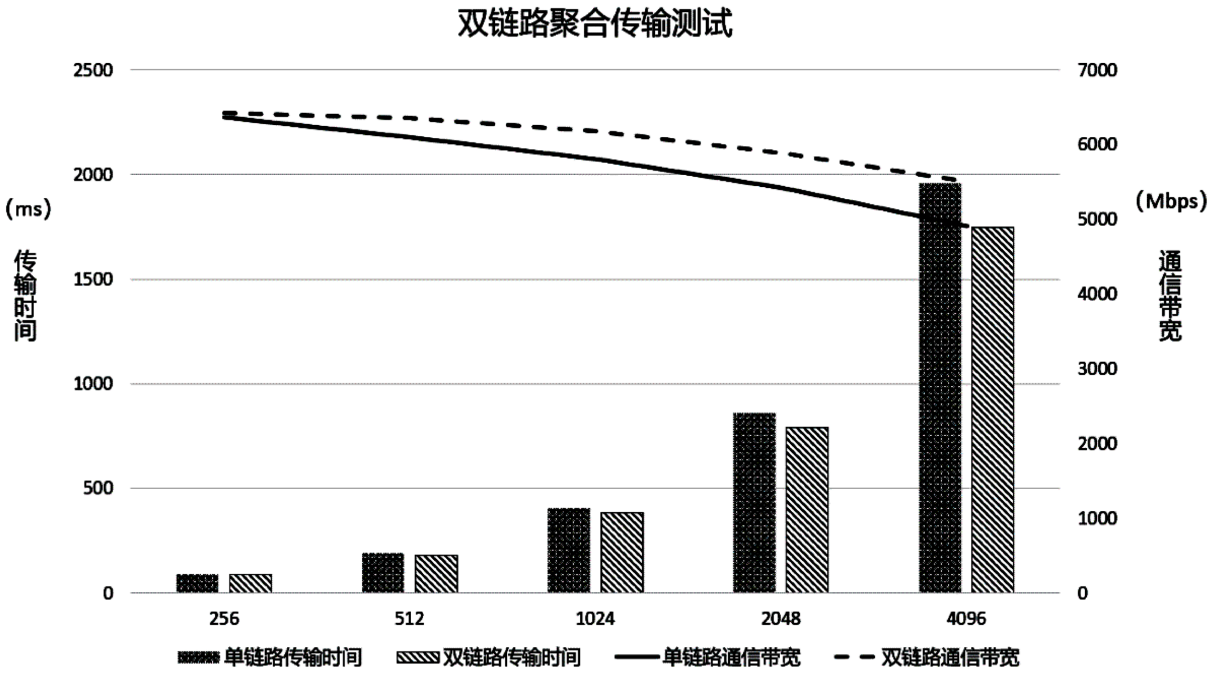

Figure 8. Dual-link aggregation transmission test
图 8. 双链路聚合传输测试

实验表明双链路通信网络的传输时间优于单链路通信网络的传输时间，在数据量较多的时候，双链路通信带宽对比单链路有明显提升，且数据量越多提升越大，在传输 4096 个数据批元(1113.4 MB)的实验中，双链路通信带宽比单链路提升了 12.02%。

5. 结论

本文基于 DPDK 框架，提出了一种面向集群内并发 AI 数据流的高性能传输机制，以解决大规模 AI 数据流传输中常见的延迟高、丢包率高和带宽利用率低等问题。并通过多网卡绑定、多核分配技术、反向流量控制机制以及双链路聚合策略，优化了集群内数据通信的效率。实验表明，本研究提出的传输机制在吞吐量、延迟和丢包率方面均优于传统方法，并在高并发场景下展现了良好的性能扩展性。

参考文献

[1] 绯樱. 2022 中国 IDC 数据中心 TOP30 [J]. 互联网周刊, 2023(16): 11.

[2] 宫学庆, 金澈清, 王晓玲, 等. 数据密集型科学与工程: 需求和挑战[J]. 计算机学报, 2012, 35(8): 1563-1578.

[3] 黄詠, 易晓东, 李姗姗, 等. 面向高性能计算机的海量数据处理平台实现与评测[J]. 计算机研究与发展, 2012, 49(S1): 357-361.

[4] 张郁. 基于 DPDK 实现企业网络性能优化的研究与设计[D]: [硕士学位论文]. 郑州: 郑州大学, 2018.

[5] 朱河清, 梁存铭, 胡雪焜. 深入浅出 DPDK [M]. 北京: 机械工业出版社, 2016: 10-34.

[6] Intel (2023) Programmer's Guide. https://doc.dpdk.org/guides/prog_guide/index.html

[7] 袁旭初, 付国, 毕继泽, 等. 分布式数据流计算系统的数据缓存技术综述[J]. 大数据, 2020, 6(3): 101-116.

[8] da Silva Veith, A., Dias de Assunção, M. and Lefèvre, L. (2023) Latency-Aware Strategies for Deploying Data Stream Processing Applications on Large Cloud-Edge Infrastructure. *IEEE Transactions on Cloud Computing*, **11**, 445-456. https://doi.org/10.1109/tcc.2021.3097879

[9] Muresano, R., Meyer, H., Rexachs, D. and Luque, E. (2017) An Approach for an Efficient Execution of SPMD Applications on Multi-Core Environments. *Future Generation Computer Systems*, **66**, 11-26. https://doi.org/10.1016/j.future.2016.06.016

[10] 周勇, 王皓, 程春田, 等. 基于 GPU 的多数据流相关系数并行计算方法研究[J]. 计算机应用研究, 2010, 27(4): 1232-1235.