文章编号：2095-2163(2022)11-0026-09 中图分类号：TP393 文献标志码：A

一个并发 AI 数据流处理节点内的通信模型

黄东生，陈庆奎
（上海理工大学 光电信息与计算机工程学院，上海 200093）

摘 要：物联网智能设备产生的大量并发 AI 数据流给云端处理中心带来了巨大的挑战，为了应对这一挑战，边缘计算将海量的并发 AI 数据流以流水线加工的方式将这些并发 AI 数据流分配给边缘服务集群内的计算节点处理。如何利用计算节点内有限的计算资源、以较低的成本提高并发 AI 数据流的处理与通信效率是本文研究的目标。提出了一种能够在处理并发 AI 数据流的计算节点内使用的通信模型，该通信模型结合 DPDK 的核绑定机制为并发 AI 数据流的接收过程、计算过程、发送过程均衡地绑定 CPU 核，还加入了数据流的分类计算、网卡端口的调度策略、缓冲环和全局网卡端口的负载监控。实验分析表明，并发 AI 数据流处理节点内的通信模型能够有效制定 CPU 核的均衡绑定策略，提高流处理节点之间的并发 AI 数据流的处理效率，还实现了多网口的均衡调度策略，使网卡端口的负载达到均衡状态，不会对端口造成太大的负载，同时带宽利用率和通信速率也大大提高，并且降低了边缘集群中流处理节点的部署成本，合理利用了节点内的计算资源处理并发 AI 数据流。

关键词：并发 AI 数据流；计算资源；均衡；DPDK；流处理节点

A communication model within a concurrent AI data streams processing node

HUANG Dongsheng, CHEN Qingkui
(School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China)

【Abstract】 The large number of concurrent AI data streams generated by IoT smart devices has brought great challenges to the cloud processing center. In order to meet this challenge, edge computing processes the massive concurrent AI data streams and distributes these concurrent AI data streams to compute node processing within the edge service cluster. How to use the limited computing resources in computing nodes to improve the processing and communication efficiency of concurrent AI data streams at a lower cost is the goal of this paper. A communication model used in computing nodes that process concurrent AI data streams is proposed. Combined with the core binding mechanism of DPDK, the communication model binds CPU cores in a balanced manner for the receiving process, computing process, and sending process of concurrent AI data streams, and also adds the classification calculation of data flow, the scheduling policy of NIC ports, the buffer ring and the load monitoring of global NIC ports. Experimental analysis shows that the communication model within the concurrent AI data stream processing node can effectively formulate a balanced binding strategy for CPU cores, improve the processing efficiency of concurrent AI data streams between stream processing nodes, and achieve a balanced scheduling strategy for multiple network ports, so that the load of the network card port reaches a balanced state, which will not cause too much load on the port. At the same time, the bandwidth utilization rate and communication rate are also greatly improved, and the deployment cost of the stream processing nodes in the edge cluster is reduced, and the computing in the node is rationally utilized to handle concurrent AI data streams.

【Key words】 concurrent AI data streams; computing resources; balance; DPDK; stream processing nodes

0 引 言

随着物联网[1]的高速发展、AI 数据流的不断增加，给物联网服务器设备对于数据流的处理带来了极大的挑战。为了应对这种现状，大量研究人员和企业人员开始将目光投向了边缘计算[2]，并将其作为云计算的补充和优化，加快数据处理的速率。边缘计算可以在云的外围部署集群服务器，集群服务器中的计算节点对于数据流实施不同的处理，可以采用流水线加工的方式并行地对数据流的各个阶段进行计算处理，如进行 AI 图像并发数据流的流水线处理，前期由一组流处理节点获取到数百个图像的数据流并对数据流进行预处理，然后发送到下一组流处理节点进行图像信息分析，再将分析结果发送到下一组流处理节点进行信息汇总，这样每个流处理节点处理并发数据流的一部分，形成一条流水线加工的方式处理数据流既能减轻单个计算节点的工作压力，又方便各个计算节点功能的维护与扩展。实现边缘计算的服务器集群

基金项目：国家自然科学基金(61572325)；上海重点科技攻关项目(16DZ1203603,19DZ1208903)；上海智能家居大规模物联共性技术工程中心项目(GCZX14014)；上海市一流学科建设项目(XTKX2012)；上海市重点项目(9DZ1208903)。

作者简介：黄东生(1996-)，男，硕士研究生，主要研究方向：网络计算与并行计算、并行复杂网络算法；陈庆奎(1966-)，男，博士，教授，博士生导师，CCF 会员，主要研究方向：计算机集群、人工智能、并行理论、物联网大规模数据分析等。

收稿日期：2022-03-18

的前提是有个高性能的通信方法。有很多种实现边缘计算服务节点通信的方法，其中一方面是采购专用的网络通信设备、比如 Myrinet、ATM 和 ServerNet 等[3]，但是这些通信设备价格普遍比较高昂，无法进行大规模的普及。另一方面随着硬件技术的发展，普通的网络设备价格走低，网卡上的端口数量和通信速度在逐渐提升，CPU 核心数量也在日益增加。但是如何充分利用这些设备资源成为业界研究和讨论的问题。伴随着这些问题，Intel 在 2014 年发布了数据平面开发套件 DPDK(Data Plane Development Kit)[4]。DPDK 使用 DMA 技术和 DDIO 技术直接进行内存访问[5]，并利用大页技术，减少了中断的发生，将数据的处理从内核态转移到用户态[6-7]，绕过了传统系统的内核协议栈，不用进行多次数据的封装与拆装，大大地提高了数据的处理效率。DPDK 还能很方便有效地管理计算节点内的网卡端口和 CPU 核：实时调度端口与 CPU 核资源去执行任务和计算出端口与 CPU 核的负载程度并及时反馈给系统，为边缘计算服务集群的实现提供了方法。

为了提升边缘计算服务集群的通信能力与计算节点内对数据流的处理速度与通信速度，满足大规模数据流的实时处理需求，本文提出了并发 AI 数据流处理节点内的通信模型，并制作相关系统来验证本文方法。该系统利用 DPDK 的绑定机制与线程亲和性等技术，根据 CPU 核资源的负载情况结合线性规划算法和排序算法均衡地为资源分配模型的接收过程、计算过程和发送过程绑定 CPU 核，提高核的利用率；还实现了基于 DPDK 的高效数据流接收；根据数据流的 id 类型进行分类排序再计算；实时监控集群内网卡端口的负载情况；依据网卡端口的负载情况制定各端口的调度策略，计算出各个端口的数据负载量，以提高系统性能，提高系统资源利用率。

1　相关工作

针对边缘计算服务集群的构建，如何在资源受限的集群计算节点上处理数据流已经成为研究的热门方向。文献[8]构建了基于 Kafka 的预警数据汇集分发系统架构，说明了 Kafka 集群原生负载均衡存在的问题，并提出了一种动态负载均衡算法，利用采集各代理节点运行时的负载指标计算负载值，但 Kafka 主要是针对应用层通信进行加速，对于系统的底层通信并不能提供很好的支持。文献[9]提出一种多网卡带宽叠加方案，其原理是所定义的端口负载均衡模型来对数据进行收发，达到端口的均衡利用，以实现数据传输的稳定性和高效性，但是主要研究的是网卡端口的通信加速，并没有针对 CPU 核的数据处理进行优化研究。

传统网络通信方案制约着相关行业的发展，经业界行业内人员的不断研究，目前已出现了如 netmap、DPDK 等高性能网络数据包处理框架。其中，netmap 基于共享内存的思想，提供了一套用户态的库函数来访问共享内存，绕过系统内核的数据包处理操作，实现了用户态和网卡之间的数据包高性能传递[10]，但是 netmap 框架仍然需要依赖中断机制来进行数据包的发送与接收，未完全解决通信瓶颈。DPDK 结合了 netmap 共享内存技术，并采用轮询模式与混合中断轮询模式，在数据包收发时，减少了中断处理的开销。近年来，因 DPDK 部署起来比较简单、社区参与人员较多、技术发展快而被广大业内人员使用。文献[11]提出一种基于 DPDK 的捕获数据包的方法，使用 DPDK 提出的 RSS 数据分发算法充分发挥了多端口通信性能。但 RSS 的分发是基于五元组，就是数据包必须包含源 IP 地址、源端口、目的 IP 地址、目的端口和使用的协议才能进行数据包的分发，主要针对网络层及其以上的协议，若只在同一局域网内通信，RSS 则不适用，还有就是若不考虑网卡端口的负载情况而使用 RSS 分发算法，容易造成端口的拥塞。文献[12]则是将 DPDK 应用在网络虚拟化，将 SRIVO 与 DPDK 结合，利用 DPDK 底层的高速数据处理性能，来提升云计算网络中密集型数据在转发方面的通信性能，但是这对硬件的要求比较高，部署成本和难度较大。

上述研究成果，有的只是针对网卡端口设计了相关的调度策略，但并没有充分利用 CPU 核资源[9-12]；还有些是没有设计任何端口的调度策略，直接利用网卡进行数据的收发[8]，没有针对计算节点中的多核多网卡设计相关并发数据流的调度策略。本文在计算节点内部根据网卡端口与 CPU 核的负载情况结合相关算法计算出网卡端口和 CPU 核的有效调度策略，均衡各个过程的 CPU 核绑定，使计算节点处理并发数据流的速率更优。将计算完成的数据发送给服务器集群中的下一个计算节点进行下一步计算，计算节点之间存在着一对一、一对多和多对多的数据流通信方式，为了提高集群整体的性能，研究了针对计算节点中的计算资源进行均衡利用的划分方法。

2　基本定义

定义 1　流处理节点　指为了减轻单个流处理

节点(计算节点)处理数据的压力而采用流水线加工的方式来处理并发 AI 数据流。每个流处理节点都会对并发 AI 数据流进行接收、存储、计算和发送处理,这里给出并发 AI 数据流的串行处理过程如图 1 所示。图 1 中,流处理节点 *A* 接收图像信息数据,进行图像预处理,如简单去除图像、图像标点和形成 AI 数据流的通信单元等操作,然后将预处理完成的数据并发地传输到流处理节点 *B* 结合 CPU 或 GPU 进行图像分析计算,将分析结果再传输到流处理节点 *C* 进行数据汇总等操作,这样一套流水加工的方式,不用将所有操作交给一个流处理节点处理,减轻了单个流处理节点的压力,提高效率。

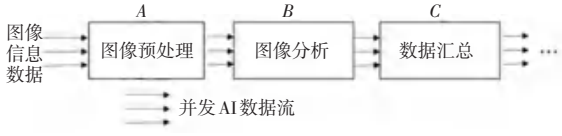

图 1 并发 AI 数据流的串行处理过程

Fig. 1 Serial processing of concurrent AI data streams

定义 2 AI 数据流与通信单元 通信单元是流处理节点(计算节点)的接收、存储、计算和发送的基本单元;经过智能终端的 AI 计算得出的中间结果称为 AI 数据流,AI 数据流(AIStream)是由一个或多个通信单元按序组成的一条连续单元序列。

AI 数据流与通信单元如图 2 所示。由图 2 可知,一条 AI 数据流可由多个通信单元 (*U*) 组成,每个 *U* 都是向 DPDK 创建的内存池[13-14](mempool)申请而来, *U* 的主要部分结构可表示为 *U*(*head*, *data*),其中 *head* 为头部区域,主要包括的字段有 *des* 目的地址、*src* 源地址和 *type* 通信单元类型,本文主要介绍2种类型:*FT_TYPE* 类型,用于更新流转发表端口信息;*DATA_TYPE* 类型, 用于存储 AI 数据流信息使各个流节点进行读取、计算和传输, 不同的类型所对应的 *data* 数据域的结果也不同,当然, *head* 部分的字段格式对所有类型的通信单元是一样的,详细情况见数据传输协议。

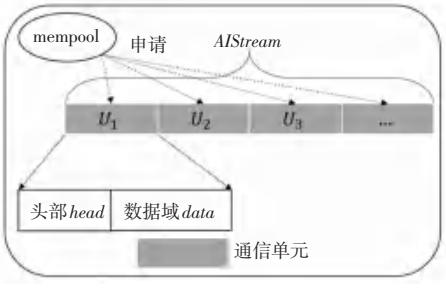

图 2 AI 数据流与通信单元

Fig. 2 AI data flow and communication unit

定义 3 通信速度 指计算节点的网卡端口单位时间内发送和接收通信单元的数量。

定义 4 处理速度 指计算节点的 CPU 核单位时间内处理通信单元的数量。

3 系统模型与实现

流处理节点内的并发 AI 数据流的通信过程如图 3 所示。图 3 中,系统首先通过 DPDK 绑定网卡端口以便并行地接收边缘服务集群中其它流处理节点(计算节点)传输来的 AI 数据流(AIStream)中的通信单元;接收过程从网卡端口获取通信单元并根据通信单元的 *id* 进行分类,继而保存至对应的接收环 (*RR*) 中;每个 *RR* 都有一个与之对应的计算线程 (*CTP*),*CTP* 将 *RR* 中的通信单元按序组合成一条单元序列进行计算操作,将计算结果存储到对应的发送环(*SR*) 中,图 3 中的 RR_1 中的通信单元由 CTP_1 处理,再将处理后的 *AI* 数据转存至 SR_1;发送过程根据流转发表(*FT*) 并结合通信单元调度策略将发送环 *SR* 中的通信单元并发地传输到下一个流处理节点进行下一步 *AI* 数据流的处理。端口监控(*PM*) 模块主要是实时监控集群中各个计算节点的网卡端口的负载情况, 将负载情况及时更新到流转发表中,为通信单元调度策略与 AI 数据流的转发提供数据支持。

3.1 数据传输协议

数据传输协议的类型由协议头部 (*head*) 中的 *type* 字段控制, 主要有 2 种传输协议类型: *DATA_TYPE* 类型和 *FT_TYPE* 类型。*AI* 数据流 *DATA_TYPE* 类型通信单元格式如图 4 所示。图 4 中,*type* 字段为 *DATA_TYPE* 的通信单元格式,其中 *des* 为接收通信单元的计算节点网卡端口的 *MAC* 地址,即为目的 *MAC*,占 6 字节;*src* 为发送通信单元的计算节点网卡端口的 *MAC* 地址,即为源 *MAC*,占 6 字节; *node* 为计算节点编号,主要是用于判断通信单元的来源与转发,占 2 字节;*id* 为通信单元编号,主要用于分类,属于同一数据流的通信单元 *id* 都是一样的,是用于通信单元分类计算的重要属性,占 2 字节;*seq* 为通信单元的序号,序号是连续编排的,方便最终数据按序组合成完整的数据流, 占 4 字节;*size* 为当前通信单元中实际需要被处理的数据大小,占 4 字节;*data* 为需要计算处理的实际数据。

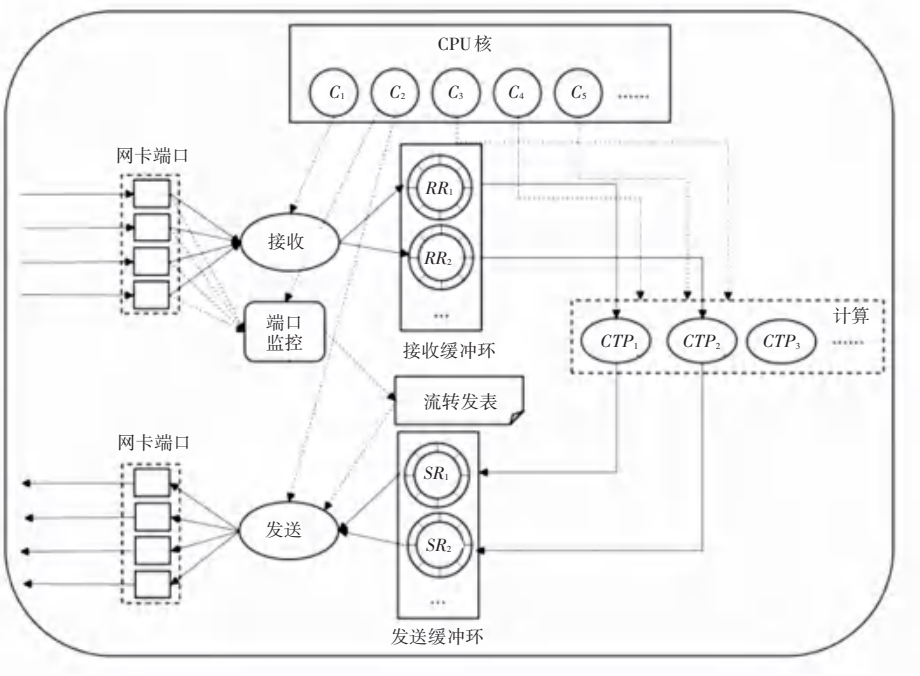

图 3　流处理节点内的并发 AI 数据流通信过程

Fig. 3　Concurrent AI data stream communication process within stream processing nodes

协议头部			协议数据域					
des (6B)	*src* (6B)	*type* (2B)	*node* (2B)	*id* (2B)	*seq* (4B)	*size* (2B)	*data* (size)	*FCS* (4B)

图 4　AI 数据流 *DATA_TYPE* 类型通信单元格式

Fig. 4　AI data stream *DATA _ TYPE* type communication unit format

AI 数据流 *FT_TYPE* 类型通信单元格式如图 5 所示。图 5 中，*type* 字段为 *FT_TYPE* 的通信单元格式，当网卡端口收到此类型的通信单元时，则由端口监控模块将 *FT_TYPE* 类型的通信单元中的数据信息更新到流转发表中。其中，协议头部与 *DATA_TYPE* 类型的通信单元格式一样，*node* 字段所表达的内容也为计算节点编号；*cnt* 为该 *node* 编号所对应的流处理节点的网卡端口个数；*p_mac* 为网卡端口的 *MAC* 地址，编号为 0 ~ *cnt*；*p_load* 为网卡端口所对应的端口负载并与端口 *MAC* 地址一一对应。

协议头部			协议数据域				
des (6B)	*src* (6B)	*type* (2B)	*node* (2B)	*cnt* (2B)	p_mac_0 (6B)	p_mac_1 (6B)	… (6B)
					p_load_0 (6B)	p_load_1 (6B)	… (6B)
					FCS (4B)		

图 5　AI 数据流 *FT_TYPE* 类型通信单元格式

Fig. 5　AI data stream *FT_TYPE* type communication unit format

3.2　接收过程

通过 DPDK 的核绑定技术为接收过程绑定若干 CPU 核、简称 Recv 核，并通过 Recv 核将网卡端口接收的通信单元分类转存至接收环 *RR* 中，这一过程称为接收过程。此时网卡端口的通信速度与 Recv 核的处理速度存在一定限制，若 Recv 核过多，总处理速度太快，则可能会造成模型中其它过程因 CPU 核的分配不均衡而使任务执行速度慢下来。下面就是系统通信模型结合线性规划算法为接收过程均衡地绑定 CPU 核的理论描述过程。

计算节点中网卡端口总数量为 n，通信速度分别为 $D=\{u_1,u_2,\cdots,u_n\}$，CPU 核总数量为 m，处理速度分别为 $V=\{v_1,v_2,\cdots,v_m\}$，一般情况下核的处理速度大于端口的通信速度，但是若 CPU 核在同时处理多个任务时，分配给单个任务的处理速度可能比通信速度小。此种情况下，在接收过程中，设 Recv 核的数量为 a，因为网卡端口总的通信速度大于 CPU 核总的处理速度会造成 Recv 核来不及分类和转存网卡端口接收的通信单元而导致数据的丢失，所以需要求出 a 的值以及与其对应 Recv 核的处理速度 $V^{rv}=\{v_1^{rv},v_2^{rv},\cdots,v_a^{rv}\}$，需要满足如下条件：

$$\begin{cases}\sum_{i=1}^{n} u_i \leqslant \sum_{j=1}^{a} v_j^{rv}\\ u_1,u_2,\cdots,u_n \in D\\ v_1^{rv},v_2^{rv},\cdots,v_a^{rv} \in V^{rv} \text{ 且 } V^{rv} \subset V\\ 0 < a \leqslant m\end{cases} \tag{1}$$

满足式(1)的条件下，目标函数为：

$$Z^{rv} = \min\left(\sum_{j=1}^{a} v_j^{rv}\right) \tag{2}$$

通过规划论中线性规划算法,在式(1)的约束下求出式(2)目标函数中 Z^{rv} 的最小取值,此时可以求出 Recv 核的数量 a 以及在 V 中选择的对应 Recv 核的处理速度 $\{v_1^{rv}, v_2^{rv}, \cdots, v_a^{rv}\}$,此时为接收过程均衡配置的 CPU 核,既能及时处理网卡端口接收的通信单元,又能提高 Recv 核的利用率。

3.3 发送过程

发送过程并发地从各个 *SR* 中获取计算完成后的通信单元,并采用端口调度策略规划每个网卡端口传输通信单元的数量,再将分类后的通信单元序列(数据流)传输到下一组对应的流处理节点继续进行处理。若发送过程传递给网卡端口的速度大于网卡端口总的通信速度,则可能会造成网口来不及发送通信单元而丢失数据,所以发送过程绑定的 CPU 核有一定的限制,网卡端口的通信速度在 3.1 节已求出,需要给出发送过程绑定的 CPU 核(Send 核)的数量 f, 以及在 V 中选择的 Send 核的处理速度 $V^{send} = \{v_1^{send}, v_2^{send}, \cdots, v_f^{send}\}$, 则需要满足 Send 核的处理应该恒定小于网卡端口总的通信速度,即 $\sum_{i=1}^{n} u_i \geqslant \sum_{j=1}^{f} v_j^{send}$ 并且 Send 核尽可能与 CLA 核不重合,即 $V^{cla} \cap V^{send} \approx \emptyset$, 求出合适的 f 以及对应的 $\{v_1^{send}, v_2^{send}, \cdots, v_f^{send}\}$,设 y 初始值为无限大,x 为-1,循环以下步骤:

(1)如果 f 不变,找出所有 $\sum_{i=1}^{n} u_i \geqslant \sum_{j=1}^{f} v_j^{send}$ 且 $V^{rv} \cap V^{send} \approx \emptyset$ 的 $\{v_1^{send}, v_2^{send}, \cdots, v_f^{send}\}$ 组合,并判断每个组合与 $\sum_{i=1}^{n} u_i$ 相减的大小,即 $\sum_{i=1}^{n} u_i - \sum_{j=1}^{f} v_j^{send} < y$, 则 $y = \sum_{i=1}^{n} u_i - \sum_{j=1}^{f} v_j^{send}$, $x = f$, 记录当下的 $\{v_1^{send}, v_2^{send}, \cdots, v_f^{send}\}$, 此后 f 加 1,继续执行(1)。

(2)如果在 f 任意组合的情况都有 $\sum_{i=1}^{n} u_i < \sum_{j=1}^{f} v_j^{send}$, 跳出循环。

(2)如果最终结果 x 不为-1,则表示分配 $f = x$ 个 CPU 核给发送过程并且核的处理速度为 $\{v_1^{send}, v_2^{send}, \cdots, v_f^{send}\}$。若最终 $x = -1$, 则 $f = 1$ 表明剩下的 CPU 核中任意一个 CPU 核的处理速度都大于网卡端口总的通信速度之和,则 $\min V^{send} > \sum_{i=1}^{n} u_i$, 这时计算资源均衡分配模型应给 Send 核绑定一个处理速度最小的 CPU 核,即 $v^{send} = \min V^{send}$, 并且需要给 Send 核设置一个延迟时间,避免 Send 核的传输数据的速度太快导致网卡端口来不及发送,延迟时间为 $\frac{v^{send} - \sum_{i=1}^{n} u_i}{v^{send}}$。其中, $v^{send} - \sum_{i=1}^{n} u_i$ 表示最小核每秒处理通信单元的数量大于总网卡端口传输通信单元的数量, v^{send} 为最小核处理速度。

3.4 计算过程

计算过程中的每个计算线程(CTP)从对应的接收环 *RR* 中获取通信单元,并会根据字段 *seq* 进行按序计算,这一过程称为计算过程。计算过程中一般会对数据进行大量的复杂计算,如 GPU 计算、图像识别计算和数据故障分析或耗时的系统调度等。系统为计算过程均衡地绑定多个 CPU 核、简称 CALC 核,并使用 DPDK 技术为每个 CALC 核绑定一个或多个计算线程(CTP)。并发 AI 数据流的大部分处理时间都在计算过程中,所以这个过程需要配置处理速度较快的 CALC 核来加快计算过程。因为并发数据流的大部分处理时间都在 CALC 过程中,所以这个过程需要配置处理速度较快的 CALC 核来加快计算过程,为 CALC 过程绑定的 CPU 核的数量为 e, 以及在 V 中为计算过程选择的 CPU 核的处理速度 $V^{calc} = \{v_1^{calc}, v_2^{calc}, \cdots, v_e^{calc}\}$, 此时应将剩余的 CPU 核全部分配给 CALC 过程,则 $e = m - a - b - f$, 其中 m 为 CPU 核总数, a 为 Recv 核数, b 为 CLA 核数, f 为 Send 核数,与之对应的 V^{calc} 为 $V^{calc} \subset V$ 且 $V^{calc} \cap V^{rv} \approx \emptyset$, $V^{calc} \cap V^{cla} \approx \emptyset$, $V^{calc} \cap V^{send} \approx \emptyset$, 表示 V^{calc} 所包含的计算核尽量不与其它过程所取的核有交集。

在 CPU 核数量不足的情况下,如某系统只有 2 个 CPU 核可用,应尽可能使计算过程有一个独立的核可用,其它过程共用另一个 CPU 核,因为一般情况下计算数据所花费的时间较多,耗能较大,所以应多分配 CPU 核到计算过程中,但是具体情况则视实际情况而定。

3.5 缓冲环

缓冲环是利用 DPDK 技术创建的 Ring 来缓存通信单元,其具有先进先出,可以设置最大空间,指针存储在表中,多消费者或单消费者(这里,消费者是指数据对象出队机制),多生产者或单生产者入队(这里,生产者是指数据对象入队机制)等特点,优点是数据交换速度快,使用简单,还可用于巨型数据的入队和出队操作。本文将缓冲环分为 2 种。一种是用于接收初始通信单元分类的接收缓冲环 RR,另一种是缓

存计算后等待发送的通信单元的发送缓冲环 SR。

3.6　端口监控与流转发表

端口监控(PM)模块绑定一个空闲的或有空余负载的 CPU 核、简称 PM 核，用于监控服务器集群中计算节点的端口通信负载情况并及时更新到流转发表 (*FT*) 中，PM 模块还需要将其所在计算节点的网卡端口负载情况每隔一个时间段就发布给服务器集群内的其它计算节点，以便服务器集群中的计算节点能实时掌控全局网卡端口负载情况，有助于转发数据时填充目的 MAC。

FT 主要作为一个流 *id* 与流处理节点的映射表，指引 AI 数据流转发到下一个流处理节点。

流转发表的结构 $FT(id,node(mac,load))$ 见表 1。表 1 中，展示了当前流处理节点中第 i 条 AI 数据流在计算后应转发到下一跳节点 $node_i$，其中 mac 为 $node_i$ 的 *MAC* 地址，用于通信单元目的地址填充；$load$ 为 $node_i$ 所对应流处理节点的各个端口负载，主要用于端口调度策略的计算。在整个计算资源均衡分配模型中 PM 核的工作负载较小，因此可利用 Recv 核、CLA 核或 Send 核的空余负载并发地执行 PM 模块的功能，若还有剩余的 CPU 核则可以分配给 PM 模块或计算过程。

表 1　流转发表结构

Tab. 1　Flow forwarding table structure

流 id	下一跳节点	*mac*	*load*
id_i	$node_i$	MAC_0	$LOAD_0$
		MAC_1	$LOAD_1$
		…	…

3.7　端口调度策略

将 *SR* 中的通信单元序列(计算完成的 AI 数据流)通过网卡端口并行地传输给下一组计算节点，均衡各个网卡端口的通信负载需要进一步考虑每个网卡端口发送通信单元的数量，制定端口的调度策略，均衡网卡端口的通信负载。当前网卡端口总发送的通信单元数量为 N，需要求出为各网卡端口分配的通信单元的数量 $\{s_1,s_2,\cdots,s_n\}$，则调度分配策略如下：

$$\begin{cases} N = SR.size \\ \sum_{i=1}^{n} s_i = N \\ s_1,s_2,\cdots,s_n \geqslant 0 \\ N \geqslant 0 \end{cases} \tag{3}$$

令每个端口发送通信单元的时间为 $t_i = \dfrac{s_i}{u_i}$，这里，$u_i \in D$，u_i 为第 i 个网卡端口的通信速度，则平均时间为 $\bar{t} = \dfrac{\sum_{i=1}^{n} t_i}{n}$，根据最小二乘法可知：

$$Z^t = \min\left(\sum_{i=1}^{n} (t_i - \bar{t})^2\right) \tag{4}$$

当 Z^t 取最小值时，可达到最优的端口调度，此时可求出每个端口的通信单元发送量 $\{s_1,s_2,\cdots,s_n\}$。

在计算出各端口的调度策略后，发送过程需要根据发送网卡端口先填充每个待发送的通信单元 U 的 $U.src$，$U.src$ 为发送该通信单元的网卡端口地址，再根据每个通信单元的 $U.id$ 到流转发表中找到对应计算节点的目的网卡端口信息，$U.des$ 填充如图 6 所示。若对应计算节点的 $FT.node.load$ 还有空余负载，可以将 $FT.node.load$ 对应的 $FT.node.mac$ 填入 $U.des$，若 $FT.node.load$ 没有空余负载，则继续扫描下一个 $FT.node.load$。

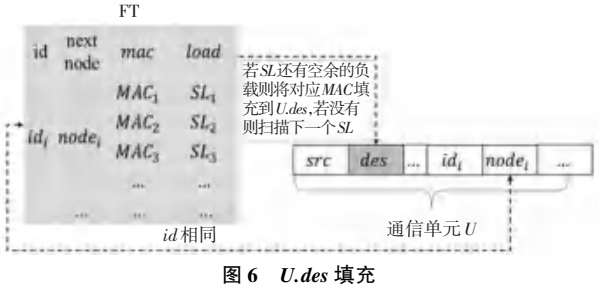

图 6　*U.des* 填充

Fig. 6　*U.des* filling

4　实验分析

4.1　实验环境

为了验证所设计的计算资源均衡分配模型的性能，研究使用了 5 台服务器作为边缘服务集群中的计算节点来模拟将 avi 格式的视频数据转换成 mp4 格式的视频数据。其中，2 台服务器设备产生视频数据，1 台服务器对视频数据进行预处理，1 台服务器对视频数据流进行 avi 到 mp4 格式的转码操作，一台服务器对最后的视频数据进行整合汇总。

本次实验中的各个服务器的布局情况如图 7 所示。图 7 中，每个计算节点(服务器、流处理节点)会先安装 DPDK 进行网卡端口的绑定和系统的初始化操作，接着由 2 个计算节点 *A* 和 *B* 生成视频数据，再并发地将视频数据传输给计算节点 *C* 进行预处理：将每个视频进行 *id* 编号和切分等操作，形成视频数据流后发送给计算节点 *D* 进行 avi 格式到 mp4 视频格式的转码操作，再将转码后的视频数据流发送给计算节点 *E* 进行视频数据单元的组合，汇总形成完整的 mp4 格式的视频。

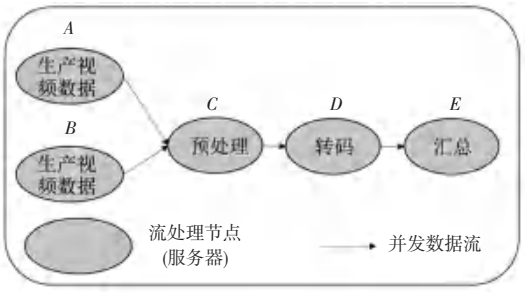

图 7 实验中服务器的布局

Fig. 7 The layout of the server in the experiment

实验环境中每个服务器节点上都有一个 4 口千兆网卡,用来进行并发数据流的接收和发送,还有 16 个 CPU 核,并通过上述各个过程的算法为各过程均衡绑定 CPU 核,服务器详细配置信息见表 2。

表 2 服务器配置信息

Tab. 2 Server configuration information

配置项	配置信息
CPU	Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz Intel(R) Xeon(R) CPU E3-1245 v3 @ 3.40 GHz
操作系统	Centos7.0
内核版本	3.10.0-123.el7.x86_64
网卡	I350-T4-4-port 1Gb/s PCI-e X4
内存	32 G/24 G
DPDK	DPDK-17.11.3

4.2 性能评估指标

在对计算资源均衡分配模型进行评估时,选取网卡端口带宽利用率、丢包率来对整个系统的性能进行评估、并发数据流的流量大小对模型中各个过程 CPU 核绑定的情况进行分析和计算任务复杂度的变化对 CALC 过程的核绑定的影响,因本文还未实现可靠传输,所以采用传统 UDP 通信来与本文方案做对比实验。网卡端口带宽利用率 $R_{pt}(t)$ 计算公式如下:

$$R_{pt}(t) = \frac{F_{pt}(t) - F_{pt}(t-1)}{\Delta t} * \frac{1}{BW} * 100\% \quad (5)$$

其中,$F_{pt}(t)$ 表示当前时间端口通过的通信单元数量;$F_{pt}(t-1)$ 表示上次采集的值;Δt 表示 2 次收集的数据量的时间差;BW 表示网卡端口的带宽、即端口的通信速度。

端口丢包率 $R_{loss}(t)$ 计算公式如下:

$$R_{loss}(t) = \left(\frac{T_{sed} - T_{rv}}{T_{sed}}\right) * 100\% \quad (6)$$

其中,T_{sed} 表示当前网卡端口输入的通信单元数量,T_{rv} 表示当前网卡端口输出的通信单元数量。

4.3 实验过程与分析

4.3.1 通信性能和丢包率

实验中的计算节点都在同一局域网下,在计算节点 C 上将每个通信单元的大小从 64 字节到 1 024 字节进行调整,并传输给计算机节点 D,数据流数量为 240 条。通信性能测试,主要在计算机节点 D 上测试本文单个端口的带宽与不采用本文模型的系统内核 UDP 通信做对比实验,在同样的数据包和数据量情况下,通信性能与丢包率如图 8 所示。

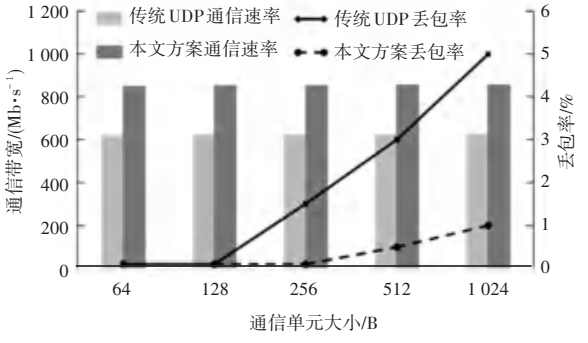

图 8 通信性能和丢包率测试

Fig. 8 Communication performance and packet loss rate testing

由图 7 可知,通信单元的大小对通信带宽的影响不大,而对于丢包率会有影响。柱状图为通信速率,由此可知,本文方案明显优于传统 UDP 通信速率;由折线图可以看出传统 UDP 的丢包率在 0.01% 到 0.05% 之间,而本文方案的丢包率在0.001% 到 0.01%之间,比传统 UDP 通信的丢包率低 10 倍左右。这 2 个结果得益于 DPDK 的高性能数据包处理机制,加速了数据包的处理,提高了通信带宽,降低了丢包率。

4.3.2 计算核数量与通信核数量的变化

随着并发数据流的增大,对各个通信过程(接收过程和发送过程)与计算过程绑定核的数量变化,与此同时在进行视频转码时,会使用到 GPU 来参与计算,所以在实验过程中也记录了 GPU 负载的变化,如图 9 所示。

由图 9 可以看出,随着并发数据流的增加,通信核的个数也逐渐增加,在总核数不变的情况下,分配给计算部分的计算核有所下降,但是一起参与计算的 GPU 使用率逐渐增加,来弥补 CPU 核计算能力不足的问题。

4.3.3 网卡端口均衡带宽利用率测试

为测试端口调度策略的可行性,数据流源源不断地从计算节点 A 和 B 传输给计算节点 C、D 和 E,

需要观测计算节点 C、D、E 的网卡端口是否处在均衡使用状态。各计算节点端口均衡使用情况如图 10 所示。

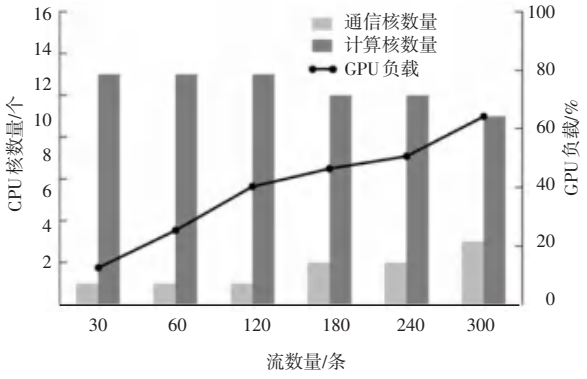

图 9　通信核数量与计算核数量和 GPU 负载的变化

Fig. 9　Changes in the number of communication cores, the number of computing cores and GPU load

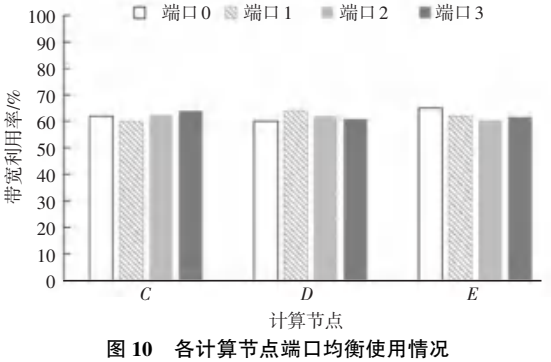

图 10　各计算节点端口均衡使用情况

Fig. 10　Balanced usage of ports on each computing node

此次测试是在 120 条数据流的情况下进行的，通过图 10 可以发现，计算节点对网卡端口的使用相对均衡，各个端口带宽利用率稳定在 60%左右，证明端口调度策略是可行的。

下面测试数据流大小对计算节点内网卡端口利用率的影响。

数据流大小对带宽利用率的影响如图 11 所示。由图 11 可知，随着并发数据流逐渐增大，计算节点的平均带宽利用率也在上升，最终稳定在 90%左右。

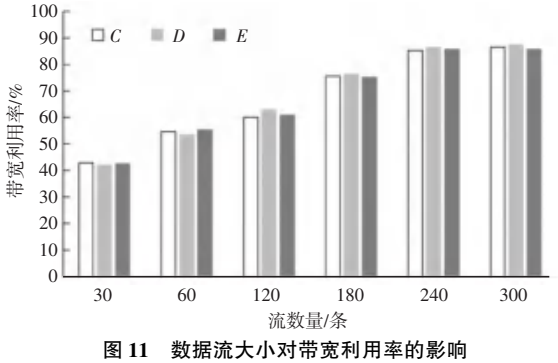

图 11　数据流大小对带宽利用率的影响

Fig. 11　The effect of data stream size on bandwidth utilization

5　结束语

本文研究与分析了并行通信、DPDK 和计算节点内的通信资源与计算资源，在边缘服务集群中以流水加工串行的方式处理并发 AI 数据流，以减轻各个计算节点的计算压力与通信压力，设计实现了一个并发 AI 数据流处理节点内的通信模型；然后对资源分配模型中的各个过程进行资源分析，并为每个过程均衡地分配 CPU 核，以提高 CPU 核的利用率，同时还设计了端口调度策略用来均衡各端口的带宽利用率，还加入了端口监控模块和流转发表实时监控服务器集群中的端口负载情况，将缓冲队列中的数据转发给下一个计算节点；最后，通过实验验证了计算资源均衡分配算法和端口调度算法的可行性，实现了计算资源的均衡分配，有效降低了边缘服务集群中计算节点的部署成本，提高了计算节点的计算效率与通信速率。接下来为完善边缘服务集群整体性能，将对模型的可靠性、能耗、CPU 核的并发处理能力进一步优化降低通信核的数量等方面做更深入系统的探索与研究。

参考文献

[1] ASHTON K.That′ Internet of Things′ thing [J]. Radio Frequency Identification Journal,2009,22(7): 97-114.

[2] GUSEV M, DUSTDAR S. Going back to the roots-the evolution of edge computing, an IoT perspective [J]. IEEE Internet Computing,2018,22(2): 5-15.

[3] BUYYAR. High Performance Cluster Computing Architectures and Systems(Volume 1) [M]. Beijing: Electronic Industry Press, 2001.

[4] ZHU Heqing, LIANG Cunming, HU Xue. In-depth understanding of DPDK [M]. Beijing: Mechanical Industry Press,2016.

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

[6] WIPPEL H. DPDK-based implementation of application-tailored networks on end user nodes [C]//2014 International Conference and Workshop on the Network of the Future (NOF). Paris, France:IEEE ,2014: 1-5.

[7] HIJAZ F, KAHNE B, WILSON P, et al. Efficient parallel packet processing using a shared memory many-core processor with hardware support to accelerate communication [C]//2015 IEEE International Conference on Networking, Architecture and Storage (NAS). Boston, MA, USA: IEEE,2015: 122-129.

[8] ALAM M, VARSHNEY A K. A new approach of dynamic load balancing scheduling algorithm for homogeneous multiprocessor system [J]. International Journal of Applied Evolutionary Computation,2017,7(2): 61-75.

[9] 车思阳. 基于 Kafka 的大容量实时预警数据汇集分发技术研究[D]. 成都：电子科技大学，2021.

(下转第 40 页)