Journal of Computer Applications
计算机应用, 2020, 40(2): 335 - 341
ISSN 1001-9081
CODEN JYIIDU
2020-02-10
http://www.joca.cn

文章编号:1001-9081(2020)02-0335-07
DOI:10.11772/j.issn.1001-9081.2019081405

基于DPDK并行通信的动态监控模型

李　翠,陈庆奎*
(上海理工大学 光电信息与计算机工程学院,上海 200093)
(*通信作者电子邮箱chenqingkui@usst.edu.cn)

摘　要:为了更好地发挥通信系统的性能,充分利用系统节点的资源,提高系统的可靠性与稳定性,设计了一种基于DPDK并行通信的动态监控模型。该模型结合DPDK和通信系统的高速率、大流量、强实时性等特点,面向多节点备份、数据包与控制包分离、多网口并行收发数据包、多核并行处理数据包进行设计,分析了监控对象,研究了数据采集方法,设计了二层通信协议DMPD,并对网口进行了细粒度监控,给出了网口负载信息模型。另外,将散列函数、调整函数与动态负载信息结合起来设计了更有效、更公平的基于多网口的动态负载均衡算法。实验结果表明,该监控模型能够准确检测和及时处理系统出现的异常,并且实现了多网口的动态负载均衡。

关键词:DPDK;并行通信;动态监控;多网口;负载均衡
中图分类号:TP393.1　　**文献标志码**:A

Dynamic monitoring model based on DPDK parallel communication

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

Abstract: In order to make better use of the performance of communication system, make full use of the resources of system nodes, and improve the reliability and stability of the system, a dynamic monitoring model based on DPDK (Data Plane Development Kit) parallel communication was designed. Combined with the high speed, large traffic, strong real-time characteristics of DPDK and communication system, the model was designed for multi-node backup, data packet and control packet separation, transmitting and receiving data packets in parallel by multiple network ports, and processing data packets in parallel by multiple cores. The monitoring objects were analyzed, the data acquisition methods were researched, and a 2-layer communication protocol named DMPD was designed. The fine-grained monitoring was performed on the network ports, and the network port load information model was given. Besides, a more efficient and fair dynamic load balancing algorithm based on multi-network port was designed by combining hash function, adjustment function and dynamic load information. Experimental results show that the dynamic monitoring model can accurately detect and timely deal with the abnormality appeared in the system, and achieve the load balancing of multiple network ports.

Key words: DPDK (Data Plane Development Kit); parallel communication; dynamic monitoring; multi-network port; load balancing

0　引言

随着物联网、5G、大数据和人工智能等技术的飞速发展,云计算因其资源利用率不高、可靠性低、可用性差、网络延迟大等不足逐渐满足不了用户业务的需求[1-3],边缘计算作为云计算的扩展和延伸,迅速得到了人们的关注。边缘计算就近边缘提供智能服务,融合了网络、计算、存储、应用等核心技术[4-5]。无论是云计算还是边缘计算都需要高可靠的虚拟化集群技术的支撑以应对海量数据带来的压力。

随着网络技术的发展,网络流量规模不断增大,网络传输速度逐步提高,接口带宽也从千兆发展到万兆甚至更高,但网络带宽的增长速度远远赶不上数据的增长速度。为了应对网络中数据包的高速传输,Intel设计了高性能的数据包处理框架DPDK(Data Plane Development Kit)[7]。DPDK在Intel架构的通用处理器上有效地执行数据包处理,相比Linux内核有10倍左右的性能提升[6-9]。

为了更好地发挥系统性能,充分利用节点的资源,对节点各个性能指标进行实时监控变得尤为重要。本文设计了基于DPDK并行通信的动态网络监控模型,实现了对通信系统运行时的各个指标的实时监控和监控信息的收集,如中央处理器(Central Processing Unit,CPU)负载、CPU利用率、内存利用率、网口信息、网络流量、节点状态等。若发现异常情况,能够第一时间进行处理,在用户还没有察觉之前处理完故障和异常,将损失降到最低;还可以根据采集的监控数据优化系统配置,包括配置出最优的路由路径和负载均衡策略等。

收稿日期:2019-07-31;**修回日期**:2019-09-03;**录用日期**:2019-09-19。　**基金项目**:国家自然科学基金资助项目(61572325,60970012);高等学校博士学科点专项科研博导基金资助项目(20113120110008);上海重点科技攻关项目(14511107902,16DZ1203603);上海市工程中心建设项目(GCZX14014);上海市一流学科建设项目(XTKX2012);沪江基金研究基地专项(C14001)。

作者简介:李翠(1994—),女,甘肃武威人,硕士研究生,主要研究方向:网络通信、并行计算;　陈庆奎(1966—),男,黑龙江哈尔滨人,教授,博士生导师,博士,CCF会员,主要研究方向:计算机集群、并行数据库、并行计算、物联网。

1 相关工作

1.1 DPDK

DPDK是一套强大、高度优化的用户空间库和驱动程序，可以帮助用户将控制面和数据面平台进行整合[10]。DPDK绕过了Linux内核协议栈，工作在用户态；利用线程的CPU亲和绑定方式避免了线程在不同核间频繁切换；采用内存大页技术降低旁路转换缓冲(Translation Lookaside Buffer，TLB)miss，减少访存的开销[9]；使用轮询技术减少中断处理开销。DPDK允许开发者自由地修改源代码，将其应用于其他场景，包括加速包处理软件库，性能调优，为网络功能虚拟化(Network Function Virtualization，NFV)技术的发展提供平台，为网络监控提供关键技术等。

1.2 监控系统

计算机监控领域有一些开源的软件系统，近年来，很多研究想通过有效监控去提升集群性能。文献[11]以Nagios为起点，介绍了一种专为云架构设计的监控系统Rocmon，它以插件为导向，灵活性很强。文献[12]研究了Nagios用于监视和解决Apache Web服务器和Oracle数据库服务器中的问题，提出了Nagios中的自我修复功能用于自动解决Linux服务器问题。文献[13]介绍了一个基于Nagios的服务框架设计和实现，该框架能够监视物理和虚拟基础架构的资源；但Nagios的缺点也不容忽视，它配置复杂，对性能、流量等的处理也不占优势。文献[14]利用Zabbix构建了分布式的网络监控系统，实现对Windows和Linux平台上关键应用服务的监控。文献[15]设计了一种基于Zabbix的网络监控系统，能够有效监控分布在不同网络中的服务器和应用程序，具有较好的稳定性和普适性；但Zabbix缺少数据汇总功能，报警设置比较多，对于深层的监控需要二次开发。

现有监控系统大多是基于Linux内核协议栈，采用应用层传输控制协议(Transmission Control Protocol，TCP)或者用户数据报协议(User Datagram Protocol，UDP)或者简单网络管理协议(Simple Network Management Protocol，SNMP)作为传输协议，传输速度都相对比较慢，不适合高速率、大流量的并行通信系统。本文介绍的动态监控采用的是基于DPDK的数据包传输技术，绕开了复杂的内核架构，工作在用户态空间，传输协议是基于二层通信协议DMPD(Dynamic Monitor Protocol based DPDK)的，很好地保证了监控信息的实时性。

1.3 并行设计

本文基于DPDK进行了二次开发，针对局域网内多核多网口服务器节点之间的并行通信，提供了高效可靠的数据包传输、转发及处理，主要用到了以下三种并行结构。

1) 多核并行处理任务。在数据包处理领域，多核架构的处理器已经广泛应用。多核处理器内部集成了多个完整的内核，采用“横向扩展”的方法提高了系统的性能。本文模型采用了8核处理器，通过任务划分的方式，让不同的核负责专门的任务，从而提高工作效率。其中：主逻辑核负责控制线程，完成一些控制任务，包括自我监控、配合协调者完成系统监控信息同步、执行系统任务、接受用户配置等；其他逻辑核负责轮询，及时将要发送的数据包选择网卡发送或者将网卡收到的数据包过滤后分发到不同的应用管道中，以此来提高并行度。

2) 多网口并行收发数据包。单网卡单端口已经不足以应对网络中高速的流量传输，由于网卡的价格相对服务器其他配件比较便宜，因此本文采用多网口技术增加带宽，提高服务器吞吐量，使收发包速率尽可能接近线速。多网口技术还可以提高通信的可靠性，通过“一包多发”，在不采用丢包重传的情况下，降低丢包率。多网口配合多核处理器，可动态调整各网口流量，降低网口负载压力，从而保证服务器访问的稳定和流畅。

3) 多管道并行处理数据包。本文采用了多管道技术，将管道与软件结合，每个应用绑定一对双向管道；各应用之间保持相对独立，结合多核多网口技术，并行处理各自的数据包。每对管道设有优先级标志、流量阈值、缓存大小、管道状态等控制字段，用户应用可以根据自己的需求来对管道进行控制，包括设置更高的发送优先级、申请更多的缓存空间、降低带宽等，从而做到分而治之。还设计了系统反馈控制，根据系统的运行状态和监控信息动态调整共享区的资源申请上限。

1.4 并行通信结构

通信系统由多个node和一个master组成。node是一个通信和被监控的节点，一般是服务器，用node_num来唯一标识。node上运行一段程序用来采集本地节点监控数据，并提交给master，每个node都可以作为master的候选者。master是一个特殊的通信和监控节点，它是由所有node选举出来的一个node，作为通信系统的协调者，用来汇总和收集node提交的监控数据。单个节点通信系统分为通信模块和应用程序接口模块。前者实现了局域网内服务器节点间的并行通信；后者为用户应用程序提供了调用接口。通信模块包含三部分：组网、监控和通信。其中：组网部分将局域网内的服务器节点及节点信息进行初始化和组织管理，为节点间通信和监控提供基础保证；监控部分对节点的硬件信息、系统运行状态以及资源利用情况等进行监控和处理；通信部分则主要提供局域网内节点之间的通信，同时也支持三层通信框架的扩展。通信系统结构如图1所示。

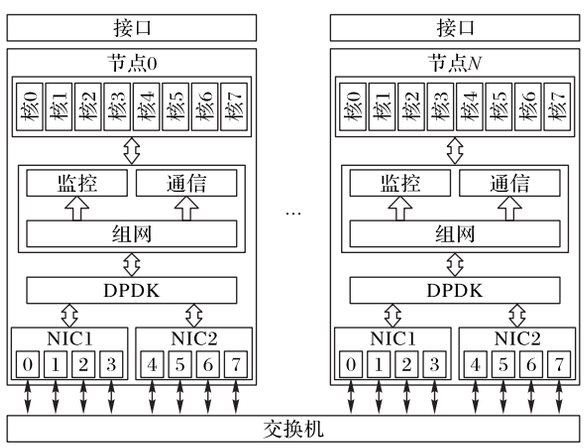

图1 节点通信系统结构

Fig. 1 Structure of node communication system

1.5 负载均衡分析

负载均衡主要是根据某种算法将网络的流量平均分配到不同的服务器、网络设备和CPU上，减轻单个设备压力。DPDK框架在负载均衡上采用了RSS(Receive Side Scaling)技术，RSS技术[16]是依据包的类型匹配相应的关键字决定收包队列。文献[17]介绍了对称RSS，将同一连接的数据包映射到同一个网卡队列，减少了线程在不同核间切换的开销；文献[18]介绍的HNLB(Hybrid NIC-offloading Load Balancer)是基于Intel提出的Flow Director技术实现的，它根据包的字段精确匹配，将其分配到某个特定队列。这两种方法都只考虑了接收端的负载均衡。

Linux内核的Bonding模块运用聚合的方法将多个物理网口绑定为一个虚拟网口,同时支持7种模式来提供负载均衡和网络冗余服务,从而实现高带宽、高可用性等目标。其中前6种模式只能处理发送数据端的负载均衡,第7种模式能够处理发送和接收数据端的负载均衡,但对于接收数据端的负载均衡只是利用地址解析协议(Address Resolution Protocol, ARP)协商机制实现,通过预先静态分配网口来实现接收端流量的负载均衡。在实际的复杂网络环境下,该模式容易出现过载现象。

DPDK的Link Bonding PMD库支持绑定相同速度和双工的rte_eth_dev端口组,允许将多个(从属)网络接口控制器(Network Interface Controller, NIC)聚合到服务器和交换机中的单个逻辑接口。然后,新的聚合的物理介质关联层(Physical Media Dependent layer, PMD)将根据指定的操作模式处理这些接口,以支持冗余链路、容错和负载均衡等功能。但PMD的实现需要更改Linux的网口驱动,使得网口不能做其他用途,并且提供的6种模式都只做了发送端网口负载均衡,没有考虑到接收端的网口压力,这样会导致接收端网口出现性能瓶颈。

2　监控模块设计

2.1　监控模型

单独的控制节点只用来对系统进行监视与控制,不参与处理任务的执行,这样的设置容易出现单点故障,使通信系统的可靠性很大程度上受到控制节点的影响。为了避免此类情况的发生,保证通信系统的可靠性和稳定性,本文介绍的通信系统的每个节点在任务处理和自我监控方面地位是对等的,控制节点只是用来收集和广播监控信息的一个载体,通信系统中每个节点都保存有整个系统的监控信息,都可以作为控制节点的备份节点。

动态监控包括自我监控和整体监控。其中:自我监控是指每个节点监控本机节点并采集监控对象数据,当本地节点出现问题时能够主动脱离集体,避免不一致的产生。整体监控由master收集保存各节点状态及资源信息,然后以心跳包DMPD协议格式广播给所有node,以便发生故障时,各节点能够做出正确的决定并及时更新系统状态。监控模型结构如图2所示。

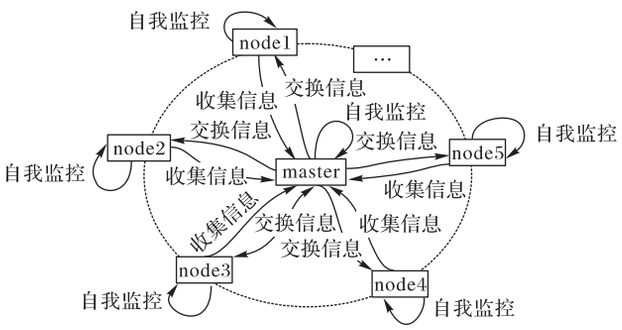

图2　监控模型结构

Fig. 2　Structure of monitoring model

2.2　监控工作原理

通信系统中每个服务器节点都配置有多个网口,其中一个网口作为外网接口,其余的用来完成基于DPDK的局域网通信。每个网口绑定一个或多个发送队列和接收队列,网口从网络上捕获监控数据包后,将数据包映射到public_ring环上,然后将public_ring环交由逻辑核去处理。本文模型选用主逻辑核解析监控数据包,并对系统监控信息进行科学准确的分析、计算和整理,针对不同的异常,做出合适、及时的处理。通信系统用public_ring环,将监控数据包与通信数据包进行分离,避免了控制数据包对常规通信数据包的干扰。

master先广播一个时间同步包,各个node收到后将采集的本机监控数据信息整理并回复给master,延时一段时间后,master将收集的所有节点的监控数据信息整理打包并广播,每一个node都会获得相关监控信息并依据这些信息分析出系统网络状况。

2.3　监控对象

通信系统中节点越多、资源越分散,系统管理就越困难。为了保证通信系统的正常运行,需要对系统各节点的运行状况和资源状况信息进行全面收集和统一管理。影响系统正常运行的因素有很多,包括节点的CPU、内存、网络、网卡等多个组件的相关信息,Linux系统已经内置了很多监控信息并且提供相应的接口,本文主要针对网卡端口和管道进行更细粒度的监控,以更好地应对局域网内高速率流量传输。主要监控内容如表1所示。

表1　监控对象

Tab. 1　Objects of monitoring

组件	监控的内容
CPU	CPU的温度、核数目、核利用率、利用率、负载、进程数和线程数
内存	内存大小、内存利用率
网口	网口的数量、流量、丢包率、状态、负载和利用率
电源	工作电压、工作电流
I/O	I/O利用率
网络	网络流量、网络负载
管道	管道的数目、大小、状态和优先级

网口流量$F_{\text{port}}(t)$(单位:b)计算公式如下:

$$F_{\text{port}}(t)=\sum_{i=1}^{n}(RX_i+TX_i)\times 1024\times 8 \tag{1}$$

其中:n表示网卡总的收发队列,i表示第i个收发队列,RX表示收包环形队列,TX表示发包环形队列。

网口丢包率$R_{\text{loss}}(t)$计算公式如下:

$$R_{\text{loss}}(t)=\left(1-T_{\text{recv}}/T_{\text{send}}\right)*100\% \tag{2}$$

其中:T_{recv}表示所有收到的包数量,T_{send}表示一共发送的包数量。

网口利用率$U_{\text{port}}(t)$计算公式如下:

$$U_{\text{port}}(t)=\frac{F_{\text{port}}(t)-F_{\text{port}}(t-1)}{\Delta\tau}*\frac{1}{BW}*100\% \tag{3}$$

其中:$F_{\text{port}}(t)$表示当前的流量值,$F_{\text{port}}(t-1)$表示上次收集时的流量值,$\Delta\tau$表示两次收集时间差;BW表示网口带宽,取值为1 Gb/s。

2.4　监控数据采集

采集的数据主要来自监控对象。通过在每个网口设置原子计数器,来记录网口流量值、收发包数量、错误包数量、丢包数量以及队列溢出次数,并且记录了网口一段时间的历史值;每个核设置原子计数器和空闲计数器,分别用来记录核的收发包数量和空轮询的次数;通过读取配置文件可以获取大页大小、管道数目、节点网口数量等静态信息;通过调用Linux系统函数可以获取CPU负载、内存大小、内存利用率、CPU温度、CPU核数目、工作电压等系统信息。

采集的信息采用DMPD协议格式进行封装,以提高信息

的规范性，简化信息的解析。封装好的信息通过基于 send/recv 模式的心跳机制在通信节点间进行消息交换。本文介绍的系统监控的心跳机制是将传统的心跳信息附加到监控信息中一起发送，减少了额外的宽带和流量消耗。

心跳数据包的传输用到了额外的传输协议 DMPD，它是基于 DPDK 开发的二层传输协议，包括心跳包协议和响应心跳包协议。心跳数据包是 master 通过广播，将收集保存的 node 信息发送到所有 node 的每一个网卡；响应心跳数据包是 node 收到心跳数据包后回复给 master 的本地节点的监控信息及对 master 的监控信息。协议格式如图 3 所示。其中：m 前缀代表网口信息，p 前缀代表管道信息，n 前缀代表 node 信息，SCORE 表示核数目，UCPU 表示 CPU 利用率，LCPU 表示 CPU 负载，LIO 表示 IO 负载，SMEM 表示内存大小，UMEM 表示内存利用率。

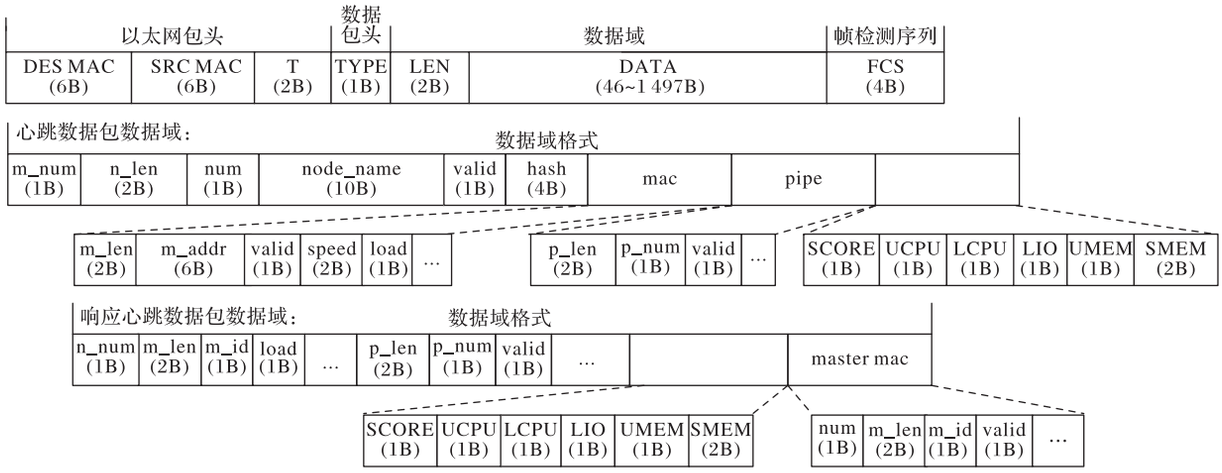

图 3 DMPD 协议示意图

Fig. 3 Schematic diagram of DMPD protocol

3 负载均衡

3.1 多网口负载均衡模型

本文针对 DPDK 的 Link Bonding PMD 库存在的缺陷与不足，设计了一种基于 DPDK 多网口的负载均衡结构，并实现了动态负载均衡算法。

当收到数据包时，将定制的关键字元组，即五元组（Snode_num，Dnode_num，Spump，Dpump，Ptype）根据微软托普利兹算法[19]计算散列值，并通过取模操作来选择发送网口和接收网口，然后根据散列表中对应网口的负载，决定是否调用调整函数，选择当前最优网口进行发包或者收包，还采用心跳机制动态更新网口负载，保证网口选择的可靠性。其中：Snode_num 表示源节点编号，Dnode_num 表示目的节点编号，Spump 表示源节点管道，Dpump 表示目的节点管道，Ptype 表示包类型。负载均衡框架如图 4 所示。

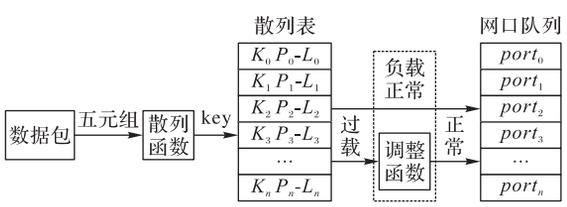

图 4 负载均衡框架

Fig. 4 Load balancing framework

3.2 动态负载均衡算法

本节给出了网口负载信息模型，设有 n 个网口，每个网口有 m 个影响网口负载的因素，每个因素占用的权重系数为 λ，λ 的取值范围为[0,1]，且权重系数满足式(4)：

$$\sum_{k=1}^{m}\lambda_k = 1 \tag{4}$$

则第 $i(i=1,2,\cdots,n)$ 个网口 t 时刻的负载为式(5)：

$$L_{\text{port\_}i}(t) = \sum_{k=1}^{m}\lambda_k p_k \tag{5}$$

本文选取的影响网口负载的因素分别为网口利用率、网口丢包率和队列溢出率，通过反复实验，选出效果比较好的权重系数分别为 0.84、0.09 和 0.07。由于距离、延迟或突发的网络故障等原因，会让采集到的监控信息产生误差，为了防止负载信息出现突变，提高负载信息准确性，本文选择使用自动调节公式平滑过渡。设上次采集负载为 $L_{\text{port\_}i}(t-1)$，当前采集负载值为 $L_{\text{port\_}i}'(t)$，i 表示第 i 个网口，则当前负载值为式(6)：

$$L_{\text{port\_}i}(t) = \gamma * L_{\text{port\_}i}'(t) + (1-\gamma) * L_{\text{port\_}i}(t-1) \tag{6}$$

其中：γ 表示调整因子，取值为 0.8。

为了对多网口的负载进行总体统筹，准确调用调整函数，本文采用了负载均衡度对各网口负载进行衡量。负载均衡度反映了多网口数据包收发处理的差异程度，它的值越小，表明各网口的数据包分配就越均匀。式(7)是网口 t 时刻的平均负载，式(8)为负载均衡度。

$$\bar{L}_{\text{port}}(t) = \frac{1}{n}\sum_{i=1}^{n}L_{\text{port\_}i}(t) \tag{7}$$

$$WL(t) = \sqrt{n} * \frac{\left|\bar{L}_{\text{port}}(t) - \mu\right|}{\sqrt{\dfrac{1}{n-1}\sum_{i=1}^{n}(L_{\text{port\_}i}(t) - \bar{L}_{\text{port}}(t))^2}} \tag{8}$$

其中：n 为网口的数量，$L_{\text{port\_}i}(t)$ 表示在时刻 t 第 $i(i=1,2,\cdots,n)$ 个网口的负载。这里的 μ 值是期望负载，可以调整，在当前网口负载高于平均负载且负载均衡度的值大于 1.86 时，就会调用调整函数选出当前最优网卡，调整函数采用小根堆结构，经过一次堆调整找出当前负载最小的网口。动态负载均衡算法流程步骤描述如下：

步骤 1 收到将要发送的数据包后，解析包头，得到关键字元组（Snode_num，Dnode_num，Spump，Dpump，Ptype）。

步骤2　通过散列函数得到散列结果,在散列表中找到发送网口对应项。

步骤3　查看发送网口状态值,如果为不可用,则执行步骤4,否则计算当前平均负载和负载均衡度;如果负载均衡度没有超过阈值,则选择该网口作为发送网口,填充源MAC(Media Access Control)地址,跳至步骤5,否则执行步骤4。

步骤4　当网口不可用或者网口过载时,通过调整函数找出当前可用发送网口中负载最小的作为发送网口,填充源MAC地址。

步骤5　通过调整函数找出当前可用接收网口中负载最小的作为接收网口,填写目的MAC地址。

步骤6　发送数据包至发送网口的TX(Transport)队列。

动态负载均衡算法的伪代码如下:

```
Algorithm of load balancing
while(pthread_quit!=0)
  Receive packets to be sent
  Parse the header of packets and get the key_touple
  Get the hash result
  Find the sending port by key
  if(port_state==0)
    Find the minimum load sending port from the available ports
    Fill source MAC address
  else
    Compute current Lport(t) and WL(t)
    if(WL(t) > threshold)
      Find the minimum load sending port from the available ports
      Fill source MAC address
    end if
  end if
  Find the minimum load receiving port from the available ports
  Fill destination MAC address
  Send packets to the TX
end while
```

4　实验分析

4.1　实验环境

在Linux环境下实现了基于DPDK的数据密集型并行通信系统,它由20 000行C语言代码组成,是基于DPDK18.02版本开发的,动态监控是它的一个组成部分。为了验证动态网络监控的有效性,搭建了千兆网络的局域网通信系统,由一台二层交换机和10台服务器组成,系统拓扑结构如图5所示,服务器配置信息则如表2所示。

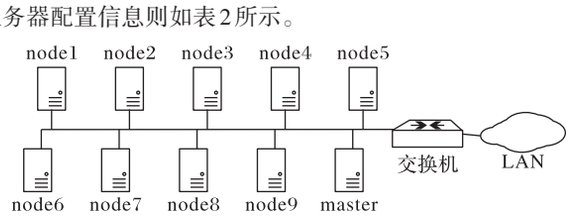

图5　系统拓扑结构

Fig. 5　System topology

在测试中,使用DPDK提供的流量发生器Pktgen[20]作为数据包生成工具来产生大小不同的数据包。局域网10台服务器的应用连接数目各不一样,不同节点应用之间相互通信,接收端服务器收到这些报文后会进行过滤解析,将其分发到目的应用管道中。整个过程中,监控模块一直在动态地监控通信系统的资源与节点的状态信息,为节点通信提供可靠性保证。

表2　服务器配置信息

Tab. 2　Configuration information of server

配置项	配置信息
CPU	Intel Core i7-4790 CPU @3.60 GHz/ Intel Xeon CPU E3-1230 V2@3.30 GHz 8核
操作系统	Centos7.0
内核版本	Linux version 3.10.0-862.14.4.el7.x86_64
网卡	4端口,i350
内存	16 GB/24 GB/32 GB,DDR3
大页	4 GB/8 GB

4.2　实验结果

在上述的实验环境中,不同节点之间相互通信,发送大小不同的数据包,其中:node1、node2发送64 B的数据包,node3、node4发送128 B的数据包,node9发送256 B的数据包,node5、node6发送512 B的数据包,node7、node8发送1 024 B的数据包,master发送1 024 B大小以上的数据包。

4.2.1　监控性能测试

master、node1、node3、node6、node7配置8个网口,其余节点配置4个网口,每隔5 s对系统资源及节点状态进行采集,表3是一段时间某一时刻系统的监控信息统计结果,其中CPU负载一列展示的是最近1 min、5 min、15 min的负载值。表4是网口监控信息统计,其中:网口状态一列每一位表示一个网口的状态,1表示正常状态,0表示不可用状态;网口数据流速率一列每个值表示一个网口单位时间的流量;网口利用率一列每个值表示一个网口收发包速率和带宽的百分比。选取master、node1、node3、node6这4个节点30 s内的8个网口总的收包、丢包、数据包处理情况进行分析计算,结果分别如图6~8所示。

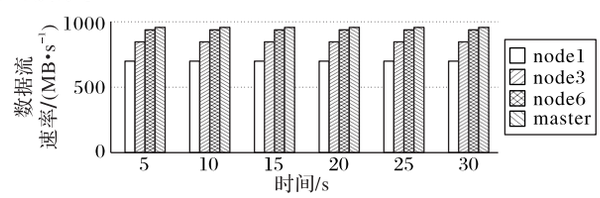

图6　不同包大小的节点的数据流速率

Fig. 6　Data flow rate of nodes with different package size

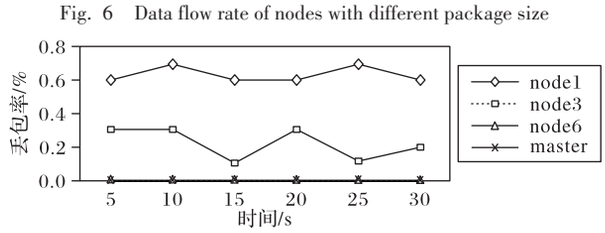

图7　不同包大小的节点的丢包率

Fig. 7　Packet loss rate of nodes with different package size

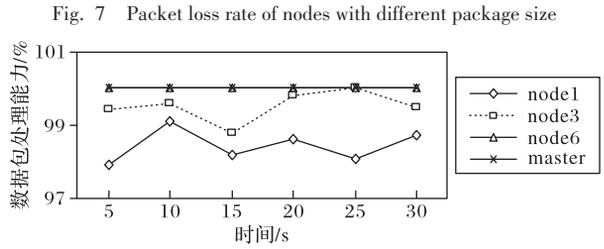

图8　不同包大小的节点的数据包处理能力

Fig. 8　Packet handling capability of nodes with different package size

表3 系统监控对象数据

Tab. 3 Data of monitoring objects of system

监控对象	数据包大小/B	核数目	CPU利用率/%	CPU平均负载值			内存/GB	内存占用率/%
				1 min	5 min	15 min		
master	>1 024	8	100	8.51	8.22	6.78	16	59
node1	64	8	100	8.28	8.30	6.01	16	57
node2	64	8	100	8.92	8.24	6.85	24	42
node3	128	8	100	7.82	7.50	5.38	24	38
node4	128	8	100	7.87	7.57	5.66	16	56
node9	256	8	100	7.85	7.56	5.42	16	57
node5	512	8	100	8.21	7.92	6.78	16	56
node6	512	8	100	8.16	7.89	6.66	16	58
node7	1 024	8	100	7.98	7.78	5.86	16	57
node8	1 024	8	100	8.07	8.01	6.23	24	37

表4 网口监控对象数据

Tab. 4 Data of monitoring objects of network port

监控对象	网口数量	网口状态	网口数据流速率/(MB·s^{-1})	网口利用率/%
master	8	1,1,1,1, 1,1,1,1	114,123,124,108, 117,124,103,120	91,98, 99, 86, 93, 99, 82, 96
node1	8	1,0,1,1, 1,1,1,1	90, 0, 77, 90 , 84,91,71,85	72 ,0, 62, 72, 67,72, 56, 68
node2	4	1,1,1,0	91,103,83,0	72, 82, 66, 0
node3	8	1,1,1,1, 1,1,1,1	108, 100, 107,108, 118, 103, 100, 96	86, 80, 86, 86, 94, 82, 80, 77
node4	4	1,1,1,1	106, 113 ,99, 107	85, 90, 79, 86
node9	4	1,1,1,1	120, 101, 113 ,116	96, 81, 90, 93
node5	4	1,1,1,1	101, 116, 108,121	81, 93, 86, 97
node6	8	1,1,0,1, 1,1,1,1	110, 121, 0,102, 118, 106 ,120,101	88, 97, 0, 82, 94, 85, 96, 81
node7	8	1,1,1,1, 1,1,1,1	105,120, 113 ,118, 124,114 ,117 ,122	84, 96, 90, 94, 99, 91, 94, 97
node8	4	1,1,1,1	111,119, 104 ,123	89, 95, 83, 98

由图6~8可以看出,随着数据包的增大,节点丢包率逐渐下降到0,收发包速率(节点数据流速率)逐渐向线速逼近,数据包处理能力逐渐增强至能够及时处理所有收到的数据包。由表3、4数据可看出,系统各个节点的各项指标相对一致,系统状态与预期一致,由此可说明动态监控模型能够正确有效地监控各个节点的各项指标,保证通信系统处在一个相对稳定的状态。

4.2.2 负载均衡测试

在上述测试的基础上,流量限制为4 Gb/s,每隔5 s,对node4(4网口,8核心)和master(8网口,8核心)的监控信息进行采集(5 s内的平均值),计算各网口的利用率,采用负载均衡前后的节点网口利用率情况如图9、10所示。

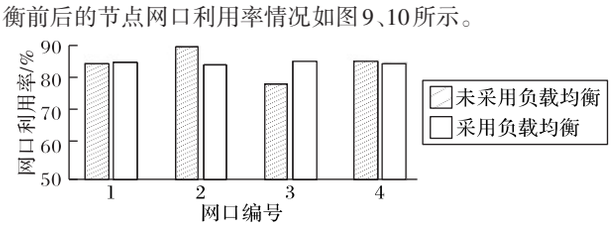

图9 node4网口利用率

Fig. 9 Network port utilization of node4

由图9和图10可以看出,未采用负载均衡时,各网口利用率层次不齐,不同网口之间差距很大,采用负载均衡方法后,各网口利用率变得相对比较平均,各网口之间相互分担负载,避免了瓶颈的发生,提高了服务器的性能。

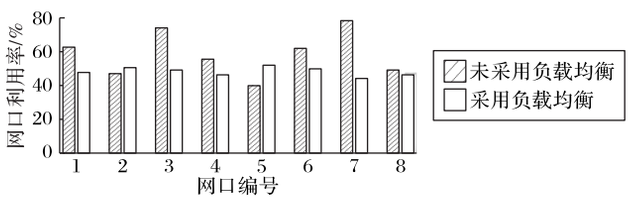

图10 master网口利用率

Fig. 10 Network port utilization of master

5 结语

本文首先对DPDK、并行通信和动态监控进行了研究分析,设计实现了基于DPDK并行通信的动态监控模型;然后对监控对象进行了分析,针对不同对象设置了合适的探测点,通过DMPD协议进行监控数据采集,并且根据监控信息设计了一种基于多网口的动态负载均衡算法;最后通过实验验证了动态网络监控及负载均衡算法的性能。接下来我们将对功耗控制、广播包过滤功能做进一步的探讨和研究。

参考文献 (References)

[1] BONOMI F, MILITO R, ZHU J, et al. Fog computing and its role in the Internet of things[C]// Proceedings of the 1st Edition of the MCC Workshop on Mobile Cloud Computing. New York: ACM, 2012: 13-16.

[2] NINAGAWA C, IWAHARA T, SUZUKI K. Enhancement of OpenADR communication for flexible fast ADR aggregation using TRAP mechanism of IEEE1888 protocol[C]// Proceedings of the 2015 IEEE International Conference on Industrial Technology. Piscataway: IEEE, 2015: 2450-2454.

[3] BELOGLAZOV A, ABAWAJY J, BUYYA R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing[J]. Future Generation Computer Systems, 2012, 28(5):755-768.

[4] SHI W, CAO J, ZHANG Q, et al. Edge computing: vision and challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646.

[5] HU Y C, PATEL M, SABELLA D, et al. Mobile edge computing — a key technology towards 5G[J]. ETSI White Paper, 2015, 11(11): 1-16.

[6] FD. io. How does fd. io relate to DPDK?[EB/OL]. [2019-05-16]. https://fd. io/news/faqs/how-does-fdio-relate-dpdk.

[7] Intel. Data plane development kit[EB/OL]. [2019-05-16]. http://dpdk. org/.

[8] Intel DPDK. Programmer's guide [EB/OL]. [2019-05-12]. http://doc. dpdk. org/guides/prog_guide/index. html.

[9] LUO T, WANG X, HU J, et al. Improving TLB performance by increasing hugepage ratio[C]// Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Piscataway: IEEE, 2015: 1139-1142.

[10] Intel DPDK. Project charter [EB/OL]. [2019-05-12]. https://www. dpdk. org/charter/.

[11] CIUFFOLETTI A. Beyond Nagios-design of a cloud monitoring system[C]// Proceedings of the 6th International Conference on Cloud Computing and Services Science. Setúbal: SciTePress, 2016: 363-370.

[12] ANUSAS-AMORNKUL T, SANGRAT S. Linux server monitoring and self-healing system using Nagios[C]// Proceedings of the 14th International Conference on Mobile Web and Intelligent Information Systems, LNCS 10486. Cham: Springer, 2017: 290-302.

[13] KATSAROS G, KÜBERT R, GALLIZO G. Building a service-oriented monitoring framework with REST and Nagios[C]// Proceedings of the 2011 IEEE International Conference on Services Computing. Piscataway: IEEE, 2011: 426-431.

[14] 郭晓慧,李润知,张茜,等. 基于Zabbix的分布式服务器监控应用研究[J]. 通信学报,2013,34(Z2):94-98. (GUO X H, LI R Z, ZHANG Q, et al. Application research on distributed Zabbix network monitoring system [J]. Journal on Communications, 2013, 34 (Z2): 94-98)

[15] 赵哲,谭海波,赵赫,等. 基于Zabbix的网络监控系统[J]. 计算机技术与发展,2018,28(1):144-149. (ZHAO Z, TAN H B, ZHAO H, et al. Network monitoring system based on Zabbix[J]. Computer Technology and Development, 2018, 28 (1): 144-149)

[16] CERRATO I, ANNARUMMA M, RISSO F. Supporting fine-grained network functions through Intel DPDK[C]// Proceedings of the 3rd European Workshop on Software Defined Networks. Piscataway: IEEE, 2014: 1-6.

[17] 令瑞林,李峻峰,李丹. 基于多核平台的高速网络流量实时捕获方法[J]. 计算机研究与发展,2017, 54(6):1300-1313. (LING R L, LI J F, LI D. Realtime capture of high-speed traffic on multi-core platform [J]. Journal of Computer Research and Development, 2017, 54 (6): 1300-1313

[18] DURNER R, VARASTEH A, STEPHAN M, et al. HNLB: utilizing hardware matching capabilities of NICs for offloading stateful load balancers [C]// Proceedings of the 2019 IEEE International Conference on Communications. Piscataway: IEEE, 2019: 1-7

[19] Mircosoft. Receive Side Scaling (RSS)[EB/OL]. [2019-05-24]. http//msdn. Microsoft. com/en-us/library/windows/hardware/ff567236 (v=vs. 85). aspx.

[20] WILES K. Pktgen-DPDK [EB/OL]. [2019-05-15]. https://github. com/pktgen/Pktgen-DPDK.

This work is partially supported by the National Natural Science Foundation of China (61572325, 60970012), the Research Fund for the Doctoral Program of Higher Education of China (20113120110008), the Key Science and Technology Research Project of Shanghai (14511107902, 16DZ1203603), the Construction Project of Shanghai Engineering Center (GCZX14014), the Shanghai Top-class Discipline Construction Project (XTKX2012), the Hujiang Foundation of China (C14001).

LI Cui, born in 1994, M. S. candidate. Her research interests include network communication, parallel computing.

CHEN Qingkui, born in 1966, Ph. D., professor. His research interests include computer cluster, parallel database, parallel computing, Internet of things.