计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2018.20170923
55(4):689-703, 2018

面向网络功能虚拟化的高性能负载均衡机制

王煜炜[1,2] 刘 敏[1] 马 诚[1,2] 李鹏飞[1,2]
[1](中国科学院计算技术研究所 北京 100190)
[2](中国科学院大学 北京 100049)
(wangyuwei@ict.ac.cn)

High Performance Load Balancing Mechanism for Network Function Virtualization

Wang Yuwei[1,2], Liu Min[1], Ma Cheng[1,2], and Li Pengfei[1,2]
[1](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)
[2](*University of Chinese Academy of Sciences, Beijing* 100049)

Abstract Network load balancer is an important middle box component for NFV(network function virtualization) which can realize load distribution accurately, through which telecom operators can improve the business ability and extend capacity flexibly and efficiently. In this paper, we design and implement a high performance NFV-oriented network load balancing mechanism and system based on DPDK framework which is capable of running on common virtualization platforms. HVLB is designed based on the SDN (software defined networking) principle which splits the data forwarding and scheduling strategy effectively. It realizes the multi-cores and multi-queues based data processing architecture in user space, and ensures the data access isolation and task handling balance meanwhile. It uses the comprehensive abilities including network transmitting and computing ability to select the objective NF and forward the packet, which can guarantee the network performance and deliver the data packet accurately at the same time. Evaluations based on a prototype of KVM platform show that our mechanism significantly improves the performance of packet processing and forwarding compared with the LVS, and it also obtains the line speed processing with 64 byte UDP packet.

Key words load balance; network function virtualization (NFV); software defined networking (SDN); high performance packet processing; comprehensive abilities

摘 要 网络负载均衡器是实现网络功能虚拟化(network function virtualization, NFV)的重要功能组件,通过其准确的业务负载分担,运营商可以灵活、高效地实现业务能力增强和容量扩展.基于DPDK技术框架设计实现了面向NFV的高性能4层网络负载均衡机制及系统HVLB,能够灵活部署运行于主流虚拟化平台.HVLB整体采用软件定义网络(software defined networking, SDN)思想设计,实现调度策略制订和数据转发的有效分离.在转发端基于用户空间实现多核多队列高效数据处理架构,同时保证各处理队列间的数据访问隔离和任务处理均衡;在控制端基于网络链路和计算相结合的综合能力作为NF选择和转发策略的制订依据,在实施业务数据准确分发的基础上保障了网络性能.基于KVM的原型系统实验结果表明:与现有LVS系统相比,HVLB极大地提升了数据包处理与转发性能,并实现了64字节UDP数据包的线速转发.

关键词 负载均衡;网络功能虚拟化;软件定义网络;高效数据包处理;综合能力

中图法分类号 TP302

收稿日期:2017-12-04;**修回日期**:2018-02-05

基金项目:国家自然科学基金项目(61472404,61732017,61472402,61501125,61502457)

This work was supported by the National Natural Science Foundation of China (61472404, 61732017, 61472402, 61501125, 61502457).

在传统的电信业务系统中,丰富多彩的业务由各类专有硬件设备承载.伴随着新技术和新服务的蓬勃发展,设备的生命周期越来越短且运营成本大幅增加,使得传统电信业务应对来自互联网企业OTT(over the top)的服务竞争中逐渐处于不利地位.

针对上述问题,由全球各大运营商牵头,欧洲电信标准化协会(ETSI)在2012年首次提出了网络功能虚拟化(network function vitalization, NFV)的概念,其核心思想在于将网络功能与专用硬件设备解耦,进而将其以软件形式部署于数据中心通用硬件平台,从而为运营商提供强大的业务整合与创新能力[1-2].基于NFV技术框架,通过特定的管理与编排策略,使得虚拟网络功能(network function, NF)以中间件的形式串接形成业务服务链(service function chain, SFC)(简称业务链)来提供服务,NF类型和顺序决定了业务链的功能内容[3].然而,部署电信业务所带来的庞大用户请求和数据流量给NFV业务系统带来了巨大压力,从而引发时延增大和吞吐量下降等严重的性能问题[4],此时单一NF功能处理单元往往不能满足数据高效处理要求,为了保障业务网络性能,需要将数据在不同NF间进行均匀分发以达到高效处理的目的.因此,在NFV系统中部署负载均衡服务至关重要.

ETSI官方发布的标准文稿中定义了4种负载均衡系统模型[5],根据负载均衡服务与网络功能耦合的情况可分为2类:内部负载均衡和外部负载均衡服务系统.前者内嵌于NF产品中,大多由专门厂家定制研发,但部署受限于具体产品架构,缺乏通用性和良好的可移植性;相应地,在外部负载均衡服务系统架构中,负载均衡器可以作为单独的网络功能NF进行部署,由NFV管理系统进行统一业务编排,部署和扩展灵活.因此,本文所提出的负载均衡机制基于外部负载均衡服务系统模型实现.

传统的网络负载均衡系统通常以专用硬件设备的形式部署于网络中,这种方式存在明显的缺陷[6-7]:1)其扩展性受到单点设备容量限制,很难满足用户业务流量快速增长需求.2)不能满足实时业务的高可达性需求.尽管通常采用成对部署的方式来避免单点故障的影响,只能提供简单的1+1冗余备份.3)缺乏业务快速迭代所需要的灵活性和可编程性,硬件系统通常很难升级和修改.4)成本较高,通常只有靠增加新的设备进行部署.相比较而言,面向NFV的负载均衡系统以软件形式部署于商用服务平台之上,因此成本得以大大降低.同时,可以利用横向扩展方式来解决可扩展性问题,此时处理容量可通过灵活增加部署虚拟机的方式来实现,从而提供$N+1$的冗余备份.同时,服务可达性和可靠性也得到增强,系统部署和升级改造也得以简化.

尽管具有上述优点,设计和部署一个面向NFV的负载均衡软件系统仍然具有非常大的挑战:1)由于NFV承载大量电信业务流量和访问请求,相关负载均衡机制必须能够在虚拟化环境中提供高吞吐量、低延迟的处理性能;2)必须保证业务服务链连接的准确一致,即属于同一个业务连接的数据报文能够转发至后序相同的NF实例,并能够应对后端资源池发生动态变化状况和其他故障;3)业务服务链所处的虚拟化网络环境复杂多变,其网络性能受到多种因素影响而波动情况明显.因此,必须根据网络链路状况以及NF工作负荷情况提供实时、准确的负载均衡调度策略.

基于数据平面开发套件(data plane development kit, DPDK)等相关技术[8],本文设计实现面向NFV的4层高性能网络负载均衡机制及系统HVLB(high performance virtual load balance system),能够灵活部署并运行于主流虚拟化平台环境,主要贡献包括4点:

1) 设计实现基于控制与转发处理分离的负载均衡处理架构,实现调度策略制订和数据转发的有效分离.同时,提出基于综合能力反馈的NF选择与调度策略,将网络链路和NF计算资源使用状况作为调度依据,有效减小业务数据分发处理过程中的开销.

2) 针对虚拟化网络环境下数据包传输与处理特点,基于网卡分流功能实现队列处理功能硬件卸载,采用多核多队列并发高效处理数据报文,并保证各处理队列间的任务流量均衡,保障数据包的高效处理.

3) 设计基于虚拟NUMA节点资源分配策略,实现从虚拟机到用户空间处理线程之间的数据零拷贝,并保证各处理队列之间数据访问隔离,有效地减少内存申请和释放及线程同步开销.

4) 基于KVM的虚拟化平台部署并实现HVLB原型系统.在虚拟化环境下,HVLB单队列和多队列处理性能均远远优于主流开源软件系统LVS(Linux virtual server)[9].针对UDP小尺寸数据包,在给定计算资源情况下,HVLB在万兆网卡上达到了线速的处理与转发性能.

1 相关工作

目前,负载均衡技术广泛受到产业界和学术界的关注,现有工作主要分为3类:

1) 传统依靠专有硬件设备实现的负载均衡方案[10-14].如引言中所述,这些方案着眼于利用硬件来提升网络数据处理性能;但是成本较高,且可扩展性、灵活性较差,不适合NFV场景中业务动态部署的需求.

2) 基于软件的网络负载均衡方案.部分方案以通用软件包的形式,按照扁平的结构进行部署,目前已经在Web搜索或其他并行计算系统中得到广泛应用,如LVS,Nginx[15],HAProxy[16]等.这部分方案可在虚拟化环境下进行灵活、快速地部署.但数据传输与处理基于内核,存在网卡中断、数据包拷贝等固有开销,网络性能较差;且需依靠多层次部署才能实现容量扩展,无法满足NFV业务的需求.

3) 随着用户业务需求进一步扩大,出现了第3类层次化负载均衡解决方案,其实质是一个可扩展的负载均衡集群系统.相关机制依托ECMP协议进行业务分流.将数据平面功能划分为若干层次,在路由器、负载均衡器和终端服务器之间建立多层负载处理,具有良好的可扩展性和处理能力.比较典型的有微软公司的Ananta[7],Duet[17],普林斯顿大学的Niagara[18]和谷歌公司的Maglev[6]等.Ananta是一个分布式的软件负载均衡器,它采用ECMP协议进行系统扩展和管理控制,并使用业务流表保持连接的一致性.尽管提供了快速路径转发机制,其单机数据包的处理性能受限于操作系统内核,在虚拟化环境中存在处理瓶颈.Duet和Niagara是基于硬件和软件混合设计的负载均衡器,旨在解决Ananta机制存在的单机转发性能问题,其使用数据中心现有的交换机来构造高性能负载均衡系统,处理与转发需要大量依靠硬件交换机,使得无法在NFV虚拟化环境中进行灵活部署和扩展.Maglev是谷歌公司为其云平台研发的网络负载均衡系统,统一部署并运行于商用物理服务器中,处理容量可以通过添加移除服务器的形式进行调整.在性能优化方面,Maglev采用网卡和进程共享内存资源的方式实现内核跨越,转发性能可以达到线速.同时引入特有的一致性Hash与连接追溯算法,提供均匀、准确的后端服务器选择和调整机制.Maglev系统适用于较大规模云平台且性能优异,但无法直接部署并应用于NFV虚拟化环境中;同时,其采用的服务集群选择策略未考虑链路状况及计算负载变化对其处理能力的影响,因而不能提供最佳的目标NF选择策略.

综上所述,现有研究工作无法直接应用于NFV虚拟化应用场景,且未充分考虑虚拟化环境下的网络链路和业务链负载变化状况对业务处理性能的影响,进而无法为业务系统提供高性能负载均衡服务.

2 概 述

2.1 架构与处理流程

HVLB采用控制与转发分离的架构设计,如图1所示,主要包括控制器(controller, CR)和转发器(forwarder, FR)两部分.其中CR主要完成虚拟服务配置、调度策略制订以及对转发器工作状况、NF计算资源占用以及网络链路状况监测.此外,CR还提供与NFV管理系统的接口,以实施资源动态配置与调整.在实际部署中,CR既可作为NFV管理系统的一部分,也可以单独部署.相应地,FR则专注于数据包的接收、调度策略执行及转发处理.在实际的数据中心环境部署中,可根据虚拟网络配置情况部署多个CR,每个CR负责管控不同的FR,从而提升管理效率,保障性能.

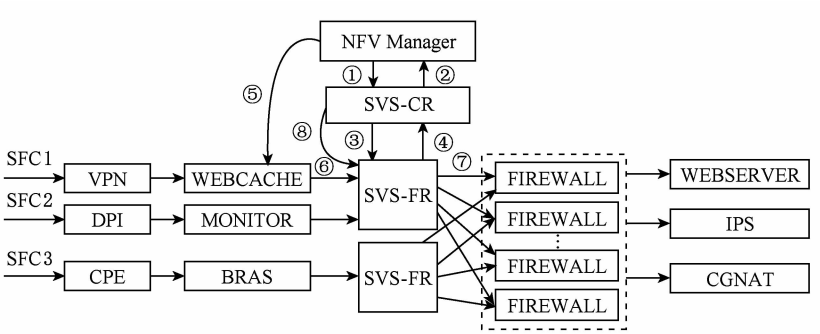


Fig. 1 The architecture and working flow of HVLB

图1 HVLB架构及处理流程

HVLB主要工作流程包括:由NFV管理系统根据业务处理和传输状况向HVLB发起负载均衡请求(步骤①),同时将需要进行负载均衡的NF及业务链的信息发送给CR,其中包含虚拟服务、待服务的前序NF信息等;收到请求后,CR给出确认回复(步骤②),同时挑选工作能力较强的FR进行处理,并向该FR下发虚拟服务配置指令(步骤③);如果未能找到合适的FR则通知NFV管理系统建立新的FR实例. FR接收到CR配置指令后,进行确认回复,并及时更新其虚拟服务列表(virtual service list, VSL)(步骤④). NFV管理系统下发路由更新策略(步骤⑤),引导待服务NF业务流转发至步骤③中选定或新生成的FR(步骤⑥),由该FR完成业务数据转发和响应接收(步骤⑦),并更新其业务连接表(connection table, CFT). CR开始根据备选目标NF计算资源占用及网络链路状况为FR制订业务调度转发策略,并更新下发至FR(步骤⑧). FR根据CR策略将数据包转发至备选NF集群中的多个NF处理,然后根据NFV管理系统路由信息将数据包转发至该业务链中的后续NF.

2.2 虚拟服务配置与管理

HVLB旨在为NFV业务提供业务负载分担,尽可能减少网络传输和处理瓶颈. 作为特殊的网络功能中间件,HVLB可为多个NF或多条业务链同时提供服务. 因此,首先需要对其提供的服务进行标识和管理.

虚拟服务(virtual service, vs)是HVLB对外进行服务宣称的虚拟实例,用五元组信息(关键字,协议,地址,端口,真实服务)标识,即 $vs_i=\langle key_i, protocol_i, vip_i, vport_i, rs_i\rangle$. 其中,protocol为传输协议,vip为虚拟服务IP地址,vport为对应端口号,关键字key为此虚拟服务的唯一标识,由协议、地址和端口所对应的值进行Hash计算得到. 真实服务(real service, rs)实质是为该虚拟服务提供处理的目标NF集合,用 $rs_i=\langle key_k, protocol_k, dip_k, dport_k, health_k, reserved_k\rangle$ 标识,其中前4项含义与关键字计算方法和虚拟服务一致,不同之处在于虚拟服务中的IP地址并不需要配置在具体网络接口上,而真实服务中的IP地址和端口信息为各NF所在虚拟机中的真实配置信息. health为每个NF的工作状态,初始化设置为1. 在系统处理过程中,若CR通过NFV管理系统获悉某个NF故障或网络不可达,即将该NF健康状态置为0. 此时暂时将该 rs 从对应 vs 中真实服务列表中剔除. reserved为调度转发策略保留信息字段,主要用来存放对应某 vs 中的各目标NF的转发比例. 当需要进行虚拟服务添加时,首先完成2.1节中的虚拟服务信息配置,由CR统一管理并维护虚拟服务集合 $VS=\{vs_i\}(i=1,2,\cdots,N)$,同时下发至各个FR,由FR维护各自虚拟服务列表. 相关虚拟服务配置结构如图2所示:

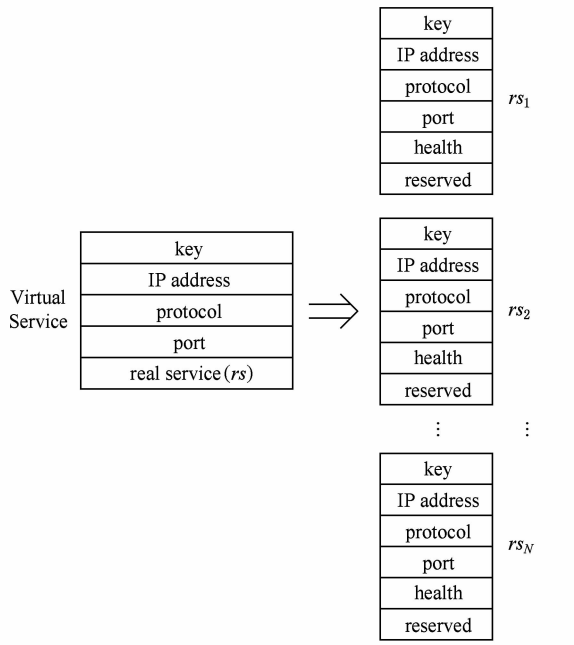


Fig. 2 The configuration of virtual service and management in HVLB

图2 HVLB虚拟服务配置与管理

3 转发器

3.1 概 述

HVLB设计实现了高性能负载均衡转发器数据处理框架,作为独立的应用程序,转发器FR部署并运行于虚拟机的用户空间,实现对Linux操作系统内核协议栈的完全跨越. 整体处理流程如下:转发器基于轮询模式批量接收来自网卡的数据请求和响应,接收到的数据包将会按照其五元组信息被分发至不同的队列进行并行处理. 首先进行业务分类操作,该过程通过查找和匹配虚拟服务关键字进行,只有匹配命中的数据包才会进入后续处理流程. 完成业务分类后,仍然以数据包五元组Hash值为关键字,在业务连接表中进行连接状态查询. 若命中,则证明该业务连接已存在,此时重用表中已选择的NF作为目标NF;若未命中,则证明该业务连接为新连接,需要从控制器CR更新的调度和转发策略中选择目标NF. 由于CR已经为各NF确定了数据

转发比例,FR 只需运行一个简单的 0～1 的随机数计算程序,然后将计算结果和 4.2.3 节中各 NF 转发比例范围进行比对即可完成调度操作.同时,在业务连接转发表中插入新的表项并维护业务连接状态.业务连接表格式如表 1 所示:

Table 1 The Structure of the Connection Table

表 1 业务连接表结构

Serial Number	Structure Information
1	key
2	vip
3	vport
4	status of the timer
5	connection status
6	connection information

HVLB 同时支持 TCP 和 UDP 传输协议,对于无连接的 UDP 协议,FR 对相同五元组的数据分发视同一次"连接",通过维护其超时状态来进行管理,超时时间默认为 300 s,若超时之后仍然没有后续相同五元组的数据包到来,则将相关表项删除.

完成调度策略选择后,转发器对数据包的路由和 MAC 信息进行修改,根据 NF 目的地址查找路由表匹配出下一跳 IP 地址和端口,查找 ARP 表匹配出下一跳 MAC 地址,路由表和 ARP 表内容通过控制器配置虚拟服务时进行统一添加,并在处理过程中进行定期更新维护.最后将数据包地址和信息写入发送队列缓冲区描述符,请求网卡完成数据包转发.完成上述处理的同时,转发器 FR 还需实时对各队列数据处理情况进行监测;另一方面,实时接收并处理来自控制器 CR 的虚拟服务配置和调度策略更新消息.

3.2 高速数据包处理

为了满足 NFV 承载的业务服务质量需求,我们希望 HVLB 的处理和转发速度能够达到线速,因此需要尽可能减少处理与转发过程中的各种开销.分析负载均衡的处理过程,主要的开销包括 I/O 传输和数据处理 2 部分.前者主要指的是操作系统及网络协议栈处理过程中引入的中断、系统调用以及数据包拷贝等开销;后者主要来源于计算资源调度竞争、线程上下文切换、数据访问竞争等[19-20].

HVLB 转发器实质上是一个位于虚拟机用户态的高性能网络处理协议栈.整体架构如图 3 所示,核心操作由处理与转发线程(processing and forwarding thread, PFT)和统计与交互线程(statistics and interaction thread, SIT)两类完成.从网卡接收到的请求和响应被分发至多个队列进行处理,每个队列对应一个 PFT 线程,单独完成负载均衡处理与转发操作;相应地,由 SIT 线程完成虚拟服务配置、各 PFT 对应队列的工作状态信息监测,并负责完成与控制器 CR 的消息交互操作.

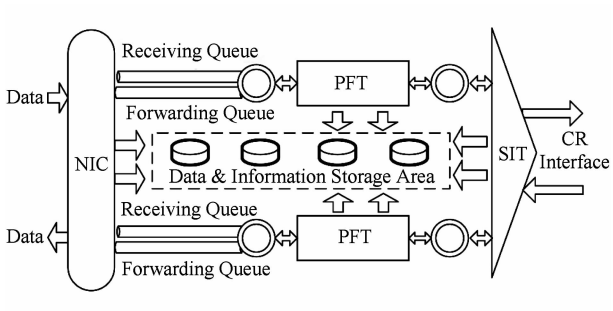


Fig. 3 The processing architecture of the FR

图 3 转发器处理架构

每个 PFT 周期性进行接收请求和响应轮询,轮询操作通过一个高速环形队列 Rece_Ring 进行,该队列存在 2 个指针,分别指向当前环形队列中的未处理数据请求和已处理请求,请求中包含了相关数据包的地址指针信息.PFT 读取对应地址信息,并随即进入后续操作处理.PFT 和 SIT 之间的通信通过 DPDK 提供的无锁环形队列实现,每个 PFT 和 SIT 之间维护一个环形通信结构 Control_Ring,以独立的生产者和消费者方式与其他 PFT 和 SIT 之间实现访问隔离,减少由线程竞争引发的开销.为保障处理性能,转发器采取了一系列高效数据处理策略,下面进行详细介绍.

3.2.1 数据高效直通和零拷贝

数据高效传输是负载均衡相关业务高性能处理的前提,在虚拟化传输环境中,传输开销主要包括中断开销、内核空间和用户空间的上下文切换以及数据拷贝开销等.基于单根 I/O 虚拟化(single root I/O virtualization, SR-IOV)技术[21],HVLB 为转发器所在虚拟机分配了特定的 HVLB-VF(virtual function),在网卡和虚拟化管理单元 Hypervisor 之间建立数据直通通道,实现对 Hypervisor 以及操作系统操作的跨越;另一方面,由于 HVLB 转发器部署于虚拟机用户空间,通过轮询的方式实现数据批量的接收与发送,实现对虚拟机操作系统的跨越.通过上述 2 次跨越操作,转发器在虚拟机用户空间和网卡之间建立了高效直通通道.同时,基于用户空间 I/O 技术,转发器在网卡和接收处理队列之间建立

共享数据包存储区域,从网卡接收的数据包被直接存入共享数据包存储区域,接收队列轮询到数据请求的同时将数据地址指针信息通知 PFT 进行处理,整个过程无需进行数据拷贝操作,有效减少了数据传输开销.

3.2.2 队列处理功能硬件卸载

3.2 节已述及,PFT 线程要负责处理负载均衡过程中的多项操作,因而会产生较大的处理开销. 如图 4 所示,利用主流智能网卡支持的 RSS[22] (receive side scaling)和 FD[23] (flow director)技术,在不改动网卡硬件的前提下,转发器进一步将硬件队列分配以及部分 Hash 计算操作卸载至网卡进行,从而有效减少处理开销,同时降低计算资源消耗,相关 FD 和 RSS 规则设置代码参见图 5.

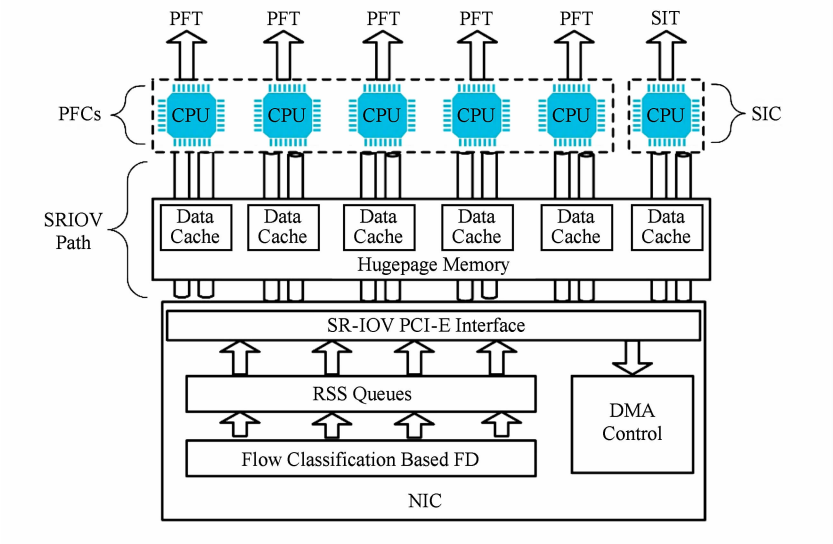


Fig. 4 Hardware function offload of queue processing

图 4 队列处理功能硬件卸载

```
int i,j=0;
rte_eth_conf.rx_mode.mq_mode=ETH_MQ_RX_RSS;
rte_eth_conf.fdir_conf.mode=RTE_FDIR_MODE_PERFECT;
mbuf_pool=rte_pktmbuf_pool_create();
struct rte_eth_fdir_filter arg=
{
  .soft_id=1;
  .input={
    .flow_type=RTE_ETH_FLOW_NONFRAG_IPV4_UDP;
    .FLOW={
      .udp4_flow={
        .dst_port=rte_cpu_to_be_16(6678);}}}
  .action={
    .rx_queue=7;
    .behavior=RTE_ETH_FDIR_ACCEPT;
    .report_status=RTE_ETH_FDIR_REPORT_ID;}}
rte_eth_dev_filter_ctrl(port,RTE_ETH_FILTER_FDIR,
RTE_ETH_FILTER_ADD,&arg);
struct rte_eth_rss_reta_entry64 reta_conf[2];
for (idx=0;idx<2;idx++){
  reta_conf[idx].mask=NULL;
  for (i=0;i<RTE_RETA_GROUP_SIZE;i++,j++){
    if (j==7)
      j=0;
      reta_conf[idx].reta[i]=j;
  }
}
```

Fig. 5 Configuration of RSS and FD for HVLB-VF

图 5 RSS 和 FD 在 HVLB-VF 中的设置

队列处理功能硬件卸载过程主要包含 2 部分操作:

1) 硬件队列分配及 CPU 绑定

系统初始化时给转发器所在虚拟机分配 N 个 vCPU,分别绑定 N 个物理 CPU 核,将其中 $N-1$ 个 vCPU 与相同数量的 PFT 线程绑定,称为处理与转发核(process and forward core, PFC);将剩余 1 个 vCPU 与统计与交互线程 SIT 绑定,称为统计与交互核(statistics and interaction core, SIC). 同时,计算每个数据包的五元组 Hash 值,根据接收与处理核的数量,依托网卡建立相同数目的硬件接收与转发队列.

在数据接收方向上,基于控制信令接收端口号,通过 FD 规则匹配将所有接收到的来自控制器的控制消息数据报文发送至 SIT 线程上进行处理,从而将控制报文和数据报文区分. 进一步基于 RSS 将余下相同五元组数据通过接收队列均匀且独立分发至多个 PFT 处理,由于各个 PFT 预先绑定了独立的 CPU 资源,因而可以避免由于 CPU 上下文切换产生的开销. 完成相关处理后,依靠 3.2.1 节中直通通道从发送队列中进行转发. 相应的 vCPU 和物理 CPU 数量可根据业务流量进行预设和调整.

2) 基于元数据的数据信息预处理

由 3.1 节分析可知,为了完成业务连接查询匹配,需要计算每个数据包的五元组 Hash 值.随着数据传输速率增加,上述计算开销也会随之增大,进而会给处理性能带来影响.基于 DPDK 提供的 RSS 功能接口,我们直接将操作 1 中用于硬件队列分配所计算的五元组 Hash 值存入相关数据包的元数据信息中,业务连接状态查询时直接通过 DPDK 提供的 *pktmbuf.hash.rss* 接口访问,有效地避免重复 Hash 计算操作带来的处理开销.

3.2.3 虚拟 NUMA 节点资源分配优化策略

目前主流多处理器系统大多基于非统一内存架构(non uniform memory access architecture, NUMA)设计,不同的 NUMA 节点的 CPU 插槽(socket)中对应不同的物理 CPU 核和本地内存资源.在 HVLB 的实际部署过程中要求为虚拟机分配多个 vCPU,每一个 vCPU 绑定一个物理 CPU 核.若当前虚拟机所需 CPU 数量 C_{vm} 大于 Socket 含有的 CPU 核总数 C_{socket} 时,则需要从不同的 Socket 中分配 CPU 资源.此时,存储访问开销将取决于处理器和不同存储区域之间的相对位置.原则上应该尽量避免不同的 Sockets 中 CPU 核交叉访问各自的本地内存资源,因为这会导致大量的上下文切换及竞争开销,使得数据包处理性能大幅下降[24-25].

为了解决上述问题,FR 初始化时即进行基于 NUMA 节点的资源分配策略优化操作,如图 6 所示.首先根据业务处理需求对转发器所需的 C_{vm} 的数量上限进行预估,然后利用 KVM 的 libvirt 接口从已有 NUMA 节点的 Socket 中划分出相同数量的 CPU 核.同时以大页(hugepage)形式为上述 CPU 分配本地内存,形成为转发器服务的物理 NUMA 资源池,其中包含若干独立的 NUMA 节点资源.完成上述操作后,便可以在虚拟机侧感知底层基于 NUMA 的资源分配情况.进而基于 DPDK 功能接口将转发器所在虚拟机 vCPU 及虚拟内存和底层物理 NUMA 资源池中的 CPU 和本地内存进行绑定映射,从而形成和物理 NUMA 资源池相对应的虚拟 NUMA 资源池,包含相同数量的虚拟 NUMA 节点.

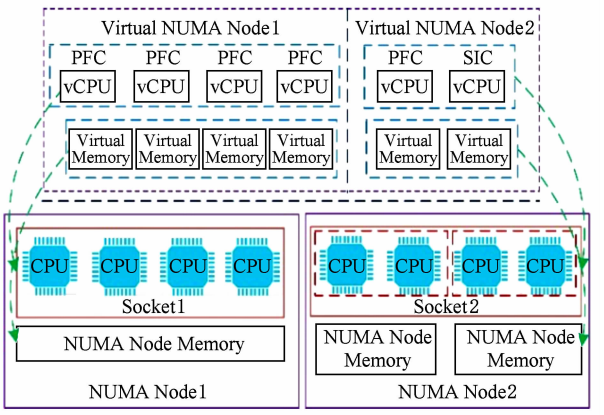


Fig. 6 The resource assignment and management of the virtual NUMA

图 6 虚拟 NUMA 的资源分配与管理

当需要为转发器分配计算资源时,首先根据实际 C_{vm} 值确定所需的 vCPU(也即物理 CPU 核)数量,从虚拟 NUMA 资源池中选择若干 vCPU 和各 PFT 进行绑定,并进一步为每个 PFT 均匀分配独立的数据缓存区,用于单独存储业务队列数据及业务连接转发表等信息.在处理过程中数据包始终存储在预设内存缓冲区内;相应地,转发器也为 SIT 设置了独立缓存区,用于存储控制器之间的交互信息.应用上述资源分配和管理机制的好处在于:1)可以直接通过数据包地址指针高效访问对应数据包缓冲区,对数据包进行修改的过程无需任何拷贝操作,有效提升了数据包处理效率;2)确保不同线程所属的处理数据相互隔离,避免了线程访问共享数据的竞争开销.

3.2.4 基于队列负载状况感知的 QoS 保障

转发器实现数据的多队列并行高效处理,依托 RSS 等技术使得到达所有队列的业务流分布均匀.然而,由于每条业务流的数据发送速率不同,造成各队列间的数据处理负载有所差异.随着业务量不断增大,可能造成某个队列处理首先过载,后续发送至该队列中的数据将会由于处理不及时而发生丢包.针对上述情况,转发器进一步采取了基于队列负载状况感知的 QoS 保障策略 RSS-A,表 2 为主要变量及符号释义对照表.

Table 2 Key Notations in the FR

表 2 转发器器处理过程主要符号释义

Symbol	Description
q_i	the i-th receiving and processing queue
f_i^j	the j-th flow in q_i
s_i^j	data transfer rate of the j-th flow in q_i
S_i	total transfer rate of all the flow in q_i
nr_i	unprocessed requests in the j-th flow of q_i
nl_i	processed requests in the j-th flow of q_i
ol_i^j	workload degree of q_i
f_{ov}^i	overload frequency of q_i
L_{no}	unoverloaded queue list
L_o	overloaded queue list

SIT 在每个 PFT 缓冲处理队列上都安置了监测探针,用来监测各队列上业务负载实时处理情况.假设转发器中共有 N 个接收与处理队列,记作 q_i $(i=1,2,\cdots,N)$,f_i^j 表示队列 q_i 中第 j 条业务流 $(j=1,2,\cdots,M)$,对应数据传输速率为 s_i^j(单位为 Mpps),该队列所有业务传输速率 S_i 为

$$S_i = \sum_{j=1}^{M} s_i^j. \tag{1}$$

统计周期为 T_{mon},取样次数为 N_{sam},缓冲队列长度为 Q,当前第 i 个队列的第 j 次取样结果中未处理请求数量为 nr_i,已处理请求数量为 nl_i,用 ol_i^j 表示该队列负载程度,则有

$$ol_i^j=\begin{cases}1, & nr_i^j+nl_i^j\geqslant Q\times\varepsilon;\\ 0, & nr_i^j+nl_i^j<Q\times\varepsilon.\end{cases} \tag{2}$$

若 nr_i 和 nl_i 之和大于等于给定阈值,则此时该队列处理负载过重,记 $ol_i^j=1$,否则 $ol_i^j=0$;ε 为负载判决因子.在此基础上,进一步计算监测周期内队列过载频率 f_{ov}^i 如下:

$$f_{\text{ov}}^i = \frac{\sum_{i=1}^{N_s} ol_i^j}{N_{\text{sam}}}. \tag{3}$$

SIT 将每个队列的 f_{ov}^i 值进行排序,并依次与预设门限值 θ_{h} 进行比较,当某个队列的 f_{ov}^i 值超过门限值 θ_{h} 时,则认定该队列处于处理过载状态.将所有过载队列的 f_{ov}^i 值从大到小排序,形成过载队列列表 L_{o};同时将所有未过载队列的 f_{ov}^i 按照从小到大排序,形成未过载队列列表 L_{no}.随即对各过载队列进行如下调整:从 L_{o} 中的数据传输速率值 s_i^j 从小到大进行排序.然后,从中依次选择若干业务流迁移至 L_{no} 中.迁移过程尽量保障 L_{o} 中排序在前的队列中的业务流优先迁移至 L_{no} 中对应排序位置的队列中,从而形成处理能力的互补.具体迁移操作通过将相关业务流数据包的五元组信息下发至 FD 规则中来实现,从而改变原有数据的接收与处理队列.令过载队列 q_i 上待迁移业务流的数量为 m,待迁移业务流传输速率总量为 S_m,目标队列 q_k 数据传输速率总量为 S_k,则迁移前后应满足如下条件:

$$0<f_{\text{ov}}^i<\theta_{\text{h}} \text{ 且 } S_m+S_k\leqslant S_i. \tag{4}$$

为了防止迁移过程对目标队列的过载影响,采取逐条逐项迁移操作的方式,并实时比较迁移前后目标队列过载频率值的变化情况,确保目标队列不出现过载情况.此外,由于目标队列上 PFT 的业务连接表中没有迁移业务流的相关信息.因此,完成迁移操作的同时需要将业务表中的连接状态信息更新至目标队列上 PFT 的业务连接表中,上述策略可有效避免队列过载,保障转发器的处理性能,相关策略如算法 1 所示.

算法 1. 基于负载感知的队列任务调整策略.

输入:N_{sam},θ_{h},N,ol_i^j.

① for $i=1\rightarrow N$
② for $j=1\rightarrow N_{\text{sam}}$ do
③ $sum=sum+ol_i^j$;
④ end for
⑤ $f_{\text{ov}}^i=sum/N_{\text{sam}}$;
⑥ if ($f_{\text{ov}}^i\geqslant\theta_{\text{h}}$)
⑦ $Insert\_queue(f_{\text{ov}}^i)\rightarrow L_{\text{o}}$;
⑧ else
⑨ $Insert\_queue(f_{\text{ov}}^i)\rightarrow L_{\text{no}}$;
⑩ $flow\_sort(L_{\text{o}},L_{\text{no}})$;
⑪ end if
⑫ end for
⑬ $flow\_migration(L_{\text{o}},L_{\text{no}})$;
⑭ $info\_extract(L_{\text{o}})\rightarrow migrlist$;
⑮ $FD\_config(migrlist)$.

4 控制器

控制器是 HVLB 系统的管理核心,负责虚拟服务配置、NF 选择与调度策略制订等.此外,控制器还负责各类信息监测与分析.

4.1 信息监测与统计分析

1) 转发器工作状况监测.HVLB 系统需要并发处理大量业务数据,在实际处理过程中,CR 需要实时掌握各 FR 的工作状况,包括各处理队列上的负载情况等.CR 通过与 FR 的 SIT 之间的消息接口进行信息交互并对相关信息进行分析,进而按照负载状况及处理情况对各 FR 进行排序.当 NFV 管理系统发起负载均衡服务请求时,CR 将参考上述列表内容,从中选择理处理能力相对较强的 FR 进行服务.当某个 FR 处于严重过载或者网络不可达时,CR 会及时将流量转移至其他 FR 或通知 NFV 管理系统启动新的 FR 实例.限于篇幅,相关内容本文不作详述.

2) 网络链路状况监测.业务链中的各 NF 通过虚拟化网络建立数据传输连接,虚拟化环境的传输特性决定了相关链路状况变化的不确定性,通过与 NFV 管理系统的接口,CR 对各 FR 至不同 NF 间的网络链路状况进行监测,以制订准确的调度策略.

3) NF 计算资源使用状况监测.目标 NF 可能为多条业务链所共用,因此其计算资源的使用情况也在

不断变化中,而计算资源使用状况对 NF 网络性能有着重要影响,需要实时监测 NF 计算资源状况.

4) 业务连接信息维护. CR 统一管理和维护所有 FR 数据处理信息.并将同一虚拟服务的业务连接表进行聚合,然后重新分发至各 FR 上.因此每个 FR 可以实时维护其他 FR 上同一虚拟服务的业务连接信息.当 FR 发生故障时,可以将业务流实时调整至其他 FR 进行处理,保障业务数据的传输.

4.2 基于综合能力反馈的 NF 选择与调度

完成虚拟服务配置后,CR 为 FR 上每一个虚拟服务制订调度和转发策略,进而将业务数据分发至不同的目标 NF 处理.在 NFV 典型应用场景中,NF 选择首先要保障不会引入新的性能处理瓶颈,主要考虑 2 方面的因素:1)虚拟化传输过程中引入的固有开销以及数据中心网络中的流量干扰,使得各 NF 与转发器之间的网络链路状况变化明显,直接影响数据的转发性能[19-20];2)各 NF 可能同时被不同业务链复用而导致计算资源的竞争,进而影响其网络性能[26-28].基于上述分析,控制器采用基于综合能力反馈的 NF 选择与调度策略,包括网络传输能力和计算能力 2 部分.表 3 为相关变量及符号释义对照表.

Table 3 Key Notations in the CR

表 3 控制器处理过程主要符号释义

Symbol	Description
NF_c	alternative set for objective NFs
nf_i	the i-th NF in NF_c
P_i	path set between nf_i and FR
p_i^j	the j-th path between nf_i and FR
b_i^j	available bandwidth of p_i^j
d_i^j	average latency of p_i^j
t_i^j	transmitting ability metric for nf_i
P_c	path set after filtrating between nf_i and FR
T_c	set of the transmitting ability for nf_i
Ω_c	set of the computing ability for nf_i
$\tilde{\omega}_i$	average CPU utility of nf_i
N_i^c	number of the CPU cores for nf_i
ω_i^j	average CPU utility in the j-th core of the nf_i
ϕ_i	deviation ratio of the element in T_c compared with the maximum
η_i	deviation ratio of the element in Ω_c compared with the maximum
a_i	normalization metric of the comprehensive ability for nf_i
λ_i	forwarding proportion for nf_i
tr_i	range of the forwarding proportion for nf_i

4.2.1 网络传输能力计算与筛选

假定某一虚拟服务对应的待选择的目标 NF 集合为 $NF_o=\{nf_1,nf_2,\cdots,nf_i,nf_{i+1},\cdots,nf_N\}$,FR 和 NF_o 集合之间传输路径集合为 $P=\{P_1,P_2,\cdots,P_i,P_{i+1},\cdots,P_N\}$.其中每个 nf_i 和 FR 之间存在 M_i 条传输路径,记作 $P_i=\{p_i^1,p_i^2,\cdots,p_i^j,p_i^{j+1},\cdots,p_i^{M_i}\}$,对其中每条路径 p_i^j 引入平均可用带宽 b_i^j 和延迟 d_i^j 两个评估属性,即 $p_i^j\leftarrow\langle b_i^j,d_i^j\rangle$.为了更好地对网络传输能力进行度量,首先针对每个 nf_i 的传输路径能力进行初评:需同时满足大于链路最低瓶颈带宽 b_{low} 和小于最低传输瓶颈延迟 d_{high} 两个条件,否则将其传输能力值置 0,进一步定义 nf_i 网络传输能力 t_i^j 如下:

$$t_i^j=\begin{cases}b_i^j/(d_i^j)^{\lambda}, & b_i^j\geqslant b_{\text{low}}\text{且 } d_i^j\leqslant d_{\text{high}},\\ 0, & b_i^j<b_{\text{low}}\text{或 } d_i^j>d_{\text{high}},\end{cases}\tag{5}$$

其中,λ 为带宽影响因子,默认 $\lambda=1.25$.然后,在 P_i 中选择 t_i^j 值最大的路径作为备选路径 p_i^c,经过初次筛选,每个 nf_i 都对应一条网络传输能力最佳的路径.同时,将网络传输能力为 0 的 k 条路径从备选集合 p_i^c 中剔除,筛选后路径集合为 $P_c=\{P_1^c,P_2^c,\cdots,P_i^c,P_{i+1}^c,\cdots,P_{N-k}^c\}$,$k$ 为综合能力值为 0 的路径数量,对应网络传输能力集合为 $T_c=\{t_1^c,t_2^c,\cdots,t_i^c,t_{i+1}^c,\cdots,t_{N-k}^c\}$.经过网络能力计算和筛选后,目标 NF 备选集合更新为 $NF_c=\{nf_1,nf_2,\cdots,nf_i,nf_{i+1},\cdots,nf_{N-k}\}(i=1,2,\cdots,N-k)$.

4.2.2 NF 计算处理能力计算与筛选

在网络传输能力计算的基础上,本节将对各 nf_i 的计算能力进行评估,引入 CPU 平均占用率 $\tilde{\omega}_i$ 作为标准.假设当前 nf_i 分配的 CPU 用 c_i^j 来表示,数量为 N_i^c,每个 c_i^j 的 CPU 资源占用率为 ω_i^j,则有

$$\tilde{\omega}_i=\sum_{j=1}^{N_i^c}\omega_i^j\Big/N_i^c,\quad i=1,2,\cdots,N-k,\tag{6}$$

采取和网络传输能力筛选类似的方法,备选 nf_i 的 $\tilde{\omega}_i$ 必须满足小于阈值 ω_{high} 的条件,否则将其计算能力值置 0,并从目标集合中剔除.筛选后对应 NF 计算能力集合为 $\Omega_c=\{\tilde{\omega}_1,\tilde{\omega}_2,\cdots,\tilde{\omega}_i,\tilde{\omega}_{i+1},\cdots,\tilde{\omega}_{N-k-r}\}$,目标 NF 集合为 $NF_c=\{nf_1,nf_2,\cdots,nf_i,nf_{i+1},\cdots,nf_{N-k-r}\}$,其中 r 为计算能力值为 0 的备选 NF 数量.

4.2.3 调度及分发策略制订

完成网络传输和计算能力的计算和筛选后,本节将计算 nf_i 的综合能力,并进一步完成调度与分发策略的制订.在前面处理结果的基础上,进一步将

Ω_c 和 T_c 中筛选后的 $N-k-r$ 个 nf_i 的处理结果进行归一化处理:对于网络传输能力而言,首先求出其中网络传输能力最大值,即 $t_{\max}\leftarrow\max\{T_c\}$,并分别计算其中每个元素相对 $t_{\max}$ 的偏移率 ϕ_i,该值越小,表明其网络传输能力越强;相应地,求出计算能力最大值 $\tilde{\omega}_{\max}$. 然后对 Ω_c 元素进行类似的归一化处理,分别计算其相对最大值的偏移率 η_i,其值越小,表明其计算能力越强,ϕ_i 和 η_i 计算方法如下:

$$\begin{aligned}\phi_i&=\frac{t_{\max}-t_i}{t_{\max}},\\ \eta_i&=\frac{\tilde{\omega}_{\max}-\tilde{\omega}_i}{\tilde{\omega}_{\max}},\end{aligned}\tag{7}$$

对计算出的偏移率 ϕ_i 和 η_i 进行加权处理,即:

$$\begin{aligned}a_i&=\gamma\times\phi_i+(1-\gamma)\times\eta_i,\\ i&=1,2,\cdots,N-k-r,\end{aligned}\tag{8}$$

其中,γ 为常数,默认值 $\gamma=0.5$. 按照不同 nf_i 对应 a_i 值的大小,进一步分别为各 nf_i 计算业务转发比例 λ_i,用以指导数据转发,即有

$$\lambda_i=\Big(\sum_{i=1}^{N-k-r}a_i\Big/N-k-r\Big)\times100\%,\tag{9}$$

完成上述过程后,控制器将该虚拟服务对应的各 nf_i 业务转发比例值进行汇总,进一步计算 nf_i 在选择比例范围 tr_i 如下:随即更新各转发器的虚拟服务配置信息中的 reserved 字段数值. FR 在进行调度处理时,只需按照随机数计算结果进行匹配即可(参考 2.2 节).

$$tr_i=\begin{cases}(0,\lambda_i], & i=1,\\ \Big(\sum_{j=1}^{i-1}\lambda_j,\sum_{j=1}^{i}\lambda_j\Big], & 2\leqslant i\leqslant N-k-r,\end{cases}\tag{10}$$

上述调度策略参见算法 2.

算法 2. 基于综合能力的目标 NF 选择策略.

输入:T_{mon},NF_{o},p_i^j,ω_i^j,γ,d_{high},b_{low},ω_{high}.

① $P_c=\varnothing$,$A=\varnothing$,$k=0$,$\eta_i=0$,$num=N$;

② for $i=1\rightarrow N$

③ for $j=1\rightarrow M_i$ do

④ $t_i^j=transmit\_capacity(p_i^j)$;

⑤ end for

⑥ end for

⑦ $transmit\_filter\{t_i^j,d_{\text{high}},b_{\text{low}}\}\rightarrow P_c$;

⑧ for $i=1\rightarrow N-k$

⑨ $\tilde{\omega}_i=compute\_capacity(\omega_i^j)$;

⑩ end for

⑪ $compute\_filter\{\tilde{\omega}_i,\omega_{\text{high}}\}\rightarrow\Omega_c$;

⑫ $t_{\max}\leftarrow\max\{T_c\}$;

⑬ $\phi_i=(t_{\max}-t_i)\div t_{\max}$;

⑭ $\tilde{\omega}_{\max}\leftarrow\max\{\Omega_c\}$;

⑮ $\eta_i=(\tilde{\omega}_{\max}-\tilde{\omega}_i)\div\tilde{\omega}_{\max}$;

⑯ $a_i=\gamma\times t_i+(1-\gamma)\times\eta_i$;

⑰ $\lambda_i=schedule\_data(a_i,N-k-r)$.

5 部署与性能分析

HVLB 核心作用在于为 NFV 提供高效的负载均衡数据分发,其处理性能可能受到多种因素的影响,包括处理队列及分配的 CPU 数量、NUMA 节点资源分配情况以及处理的数据包类型等. 本节对 HVLB 的处理性能进行全面评估. 为了便于比较,我们搭建了原型系统实验平台,包括 8 台物理服务器 $S_1\sim S_6$,C_1 和 C_2,在 S_1 上建立 4 个虚拟机 V_1,V_2,V_3,V_4,其中 V_1 运行 HVLB 转发器,V_2 运行性能对比系统,V_3 运行 HVLB 控制器,V_4 作为控制器备份. 在 $S_2\sim S_6$ 上建立多个虚拟机,用来模拟备选 NF 集群. 在 C_1 和 C_2 上部署和安装测试工具 PktGen-DPDK[29] 和 Thrulay[30]. 实验过程中由 C_1 和 C_2 单独或同时发送请求至 HVLB 转发器. 令转发器和 NF 处于同一虚拟子网,并设置 FR 为 NF 网关,实验配置参数如表 4 所示:

Table 4 Configuration of the Experiment Platform

表 4 实验平台配置

Device	Configuration
Physical Server	Ubuntu 14.04, KVM, 4 Intel Xeon E5-2620; 24Cores CPU; 64 G RAM, Intel x710 10 Gb NIC
FR(Virtual Machine)	Fedora 20, 8Cores CPU, 16 GB RAM; 2 Virtual NICs
CR(Virtual Machine)	Fedora 20, 4Cores CPU, 6 GB RAM; 1 Virtual NICs

5.1 HVLB 单机处理性能

首先来看转发器的单机处理性能,我们在 V2 上部署业界广泛采用的 LVS[9] 进行性能对比. LVS 系统可以灵活地部署于虚拟化环境中,但其数据转发处理依赖 Linux 内核进行. 为公平起见,我们同时为 V1 和 V2 分别配置基于 SR-IOV 的虚拟功能 VF1 和 VF2,保证实验过程中 2 个系统的数据收发均能跨越 Hypervisor 而直接到达各自虚拟网卡,并同时保证 CPU 和内存资源处于相同 NUMA 节点. 进而对比 HVLB 和 LVS 处理不同类型、不同大小的数据包时的网络性能.

5.1.1 单处理队列性能

首先来看单处理队列的情况,此时 HVLB 转发器工作在最低配置情况:一个 PFT 通过虚拟机 vCPU 绑定 1 个物理 CPU 核,SIT 绑定 1 个 CPU 核,实验中为 LVS 系统配置相同数量的 CPU 核. 用 PktGen 均匀发送 2 种类型的 UDP 数据包并逐渐增大到线速,大小从 64～1 518 B 变化. 第 1 种为 Constant 5-tuple 类型,简称 UDP-C,此时发包工具均匀产生 1 条业务流,所有的数据包具有相同的五元组;第 2 种为 Independent 5-tuple 类型,简称 UDP-I,发送数据包时不断改变源 IP 地址和发送端口,因此各数据包对应不同的五元组值.

吞吐量性能实验结果如图 7(a)所示,横轴为数据包尺寸大小,左边纵轴为吞吐量(单位 Gbps),右边纵轴为平均处理延迟. 对于 UDP-C 类型数据包,HVLB 的吞吐量性能远超 LVS,以 64 B 小包为例,HVLB 的最大吞吐量约为 1.7 Gbps,约为 LVS 的 3.8 倍,其单核处理 1 024 B 数据包吞吐量性能已达到网卡饱和处理值 10 Gbps. 相应地,LVS 处理各个尺寸大小的数据包均无法达到或者接近 10 Gbps. 对于 UDP-I 类型数据包,HVLB 的处理 64 B 数据包吞吐量约为 1.48 Gbps,为 LVS 的 5.9 倍,但是略低于处理 UDP-C 类型数据的性能,这是因为 UDP-I 类型每个数据包的五元组信息各不相同,每个数据包处理都要进行业务连接表查询以及新表项建立操作,引入额外的处理开销. 对于 UDP-C 类型数据而言,业务连接表中已存储相关信息,只需进行查询操作.

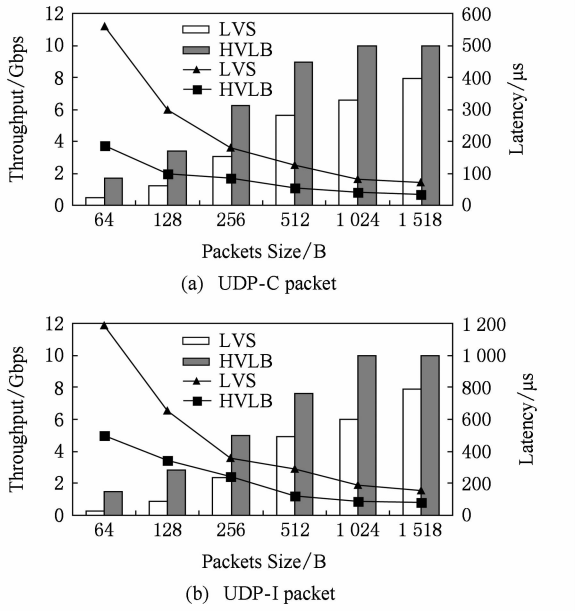


Fig. 7 The network performance under single queue

图 7 单处理队列网络性能

平均延迟性能实验结果如图 7(b)所示,对于 UDP-C 和 UDP-I 类型数据包,在单核大尺寸数据包(512～1 518 B)场景下,HVLB 和 LVS 的延迟性能差距并不明显,随着数据包大小的减少,二者的差别越来越明显,处理 64 B 小包时,对于 UDP-C 类型数据,LVS 平均处理延迟约为 HVLB 的 3 倍;对于 UDP-I 类型的数据包,LVS 平均处理延迟约为 HVLB 的 2.4 倍. 从实验结果分析可知,LVS 基于 Linux 内核处理数据,引入大量中断、拷贝开销,最终影响网络性能.

5.1.2 多处理队列性能

分析多处理队列下系统处理性能,实验中为 HVLB 和 LVS 系统配置相同的计算资源和处理队列,同时发送 UDP-C 和 UDP-I 两种类型,大小为 64 B 的数据包,并保证为每个接收与处理队列发送相同数量的业务流. 结果如图 8(a)和图 8(b)所示,横轴为处理与转发核数量,左边纵轴为数据转发吞吐量(单位为 Mpps),右边纵轴为平均处理延迟. 对于 UDP-C 类型的数据包而言,队列和相应计算资源的增加增强了 LVS 系统的处理能力,在 6 和 7 个处理核时达到最大吞吐量性能约为 3.4 Mpps,其队列平均处理延迟约为 574.5 μs;相应地,HVLB 系统在 6 个处理核(处理线程)时已达到线速 14.88 Mpps;比 LVS 提升近 4.4 倍,队列平均处理延迟为 181.7 μs;对于 UDP-I 类型数据包而言,LVS 和 HVLB 处理性能均低于前者. HVLB 需要 7 个 CPU 才达到线速,平均延迟为 386 μs;相比之下,LVS 在 7 个处理核

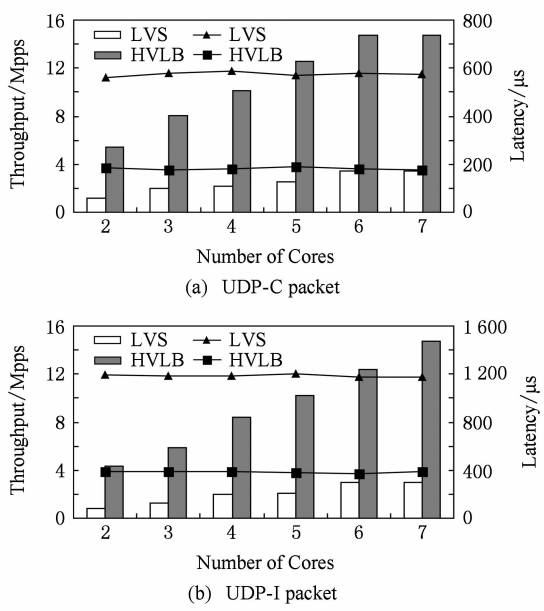


Fig. 8 The network performance under multi-queues

图 8 多处理队列网络性能

情况下达到的最大吞吐量性能仅为 3 Mpps,平均延迟为 1 187.5 μs,说明在大量业务数据并发的场景下,LVS 的处理瓶颈仍然来源于内核数据收发过程中的固有开销.

5.2 NUMA 分配策略对处理性能影响

在前面的实验中,我们对转发器的单机处理性能进行了分析,本节将进一步采用 3.2.3 节中的虚拟 NUMA 节点资源分配策略前后对系统性能的影响.实验中令 2～7 个队列上的 PFT 绑定不同 NUMA 节点资源.为简单起见,当队列数量为偶数时,平均分配 PFT 至 2 个不同的 NUMA 节点的计算资源;当队列数量为奇数时,在平均分配基础上随机分配剩余 PFT 给某个 NUMA 节点,测试内容和 5.1.2 节相同.

实验结果如图 9 和图 10 所示.在图 9 的吞吐量性能对比中,对于 UDP-C 类型的数据包而言,对于没有经过优化的跨 NUMA 节点(crossing NUMA nodes, CNN)情况下,吞吐量性能出现明显下降,在 6 个处理队列条件下,未经过策略优化和优化后(crossing NUMA nodes optimization, CNNO)情况下分别为 12.95 Mpps 和 14.47 Mpps,提升约 11.7%,后者仍然略低于 5.1.2 节中单 NUMA 节点(single NUMA node, SNN)情况下的处理性能;对于 UDP-I 类型数据包而言,未经过策略优化和优化后分别为 10.55 Mpps 和 12.05 Mpps,提升约 14.2%,同样略低于单 NUMA 节点下的处理性能.

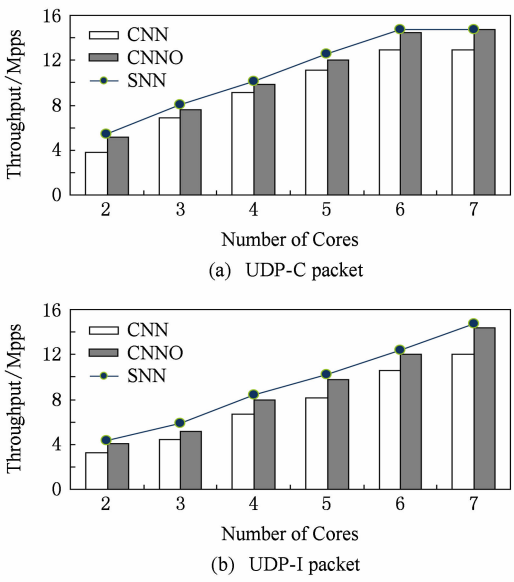


Fig. 9 The network throughput performance under different NUMA resource allocation strategies

图 9 不同 NUMA 资源分配策略下吞吐量对比

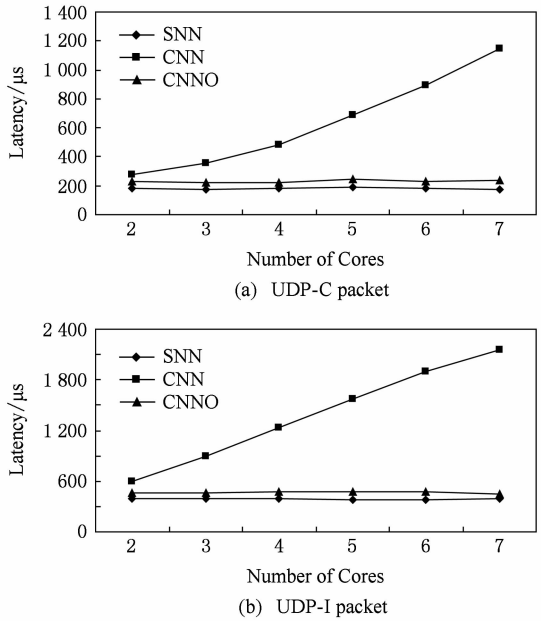


Fig. 10 The latency performance under different NUMA resource allocation strategies

图 10 不同 NUMA 资源分配策略下延迟性能对比

从图 10 中可以看出,相比吞吐量性能而言,跨 NUMA 节点分配(CNN)对延迟处理性能影响更为明显,并且随着处理队列(处理与转发核)的数量的增加而增大,在 7 个处理队列时 2 种类型数据包的平均处理延迟分别达到了 1 150 μs 和 2 149 μs.经过 NUMA 策略优化(CNNO)后,对应 2 种数据包类型的平均处理延迟分别为 239 μs 和 442 μs,略高于单 NUMA 节点情况(SNN)下的 177 μs 和 391 μs.由上述结果可知,采用虚拟 NUMA 节点资源分配策略后,系统吞吐量和延迟性能均得到提升,但相对单 NUMA 节点情况仍有下降,原因是 HVLB 采用一个 SIT,PFT 和 SIT 间存在跨 NUMA 节点访问开销.

5.3 基于负载状况感知的队列 QoS 性能

在 5.1～5.2 节实验中发送的测试数据在各队列分布均匀,如果某些业务数据发送速率较高会造成部分队列处理过载而引发性能下降.本文提出 RSS-A 策略来实现队列间的处理均衡,本节将通过实验验证应用该策略前后对系统性能的影响.我们使用 C_1 和 C_2 同时发送 UDP-C 类型 64 B 数据包至转发器.转发器配置为 6 个处理队列(绑定 6 个处理与转发核).通过改变源端口,在 C_1 上构造并为每个队列生成 10 条业务流,并为每个队列的业务流设置不同的增长速率,分别为 0.02 Mpps, 0.03 Mpps, 0.04 Mpps, 0.05 Mpps, 0.06 Mpps 和 0.1 Mpps,

显然,队列 6 负载最重;同时,由 C2 生成并均匀发送每个队列 10 条业务流,增速为 0.01 Mpps,令 C_1 和 C_2 逐渐增大发送速率.过载判决阈值因子 $\varepsilon=0.95$,策略处理周期为 1 s,测试周期为 5 min,取样次数为 20,$\theta_h=0.5$.

图 11(a)为各队列达到稳态时吞吐量性能情况,可以看出,在业务传输速率分布不均匀情况下(RSS without uniformity, RSS-WOU),队列 6 很快处于过载状况,吞吐量性能受到较大影响,仅为 10.89 Mpps;相应地,应用 RSS-A 策略情况后(RSS-A without uniformity, RSS-A)达到 13.83 Mpps,且各队列处理情况均匀,与图 8 中分布均匀情况下(RSS with uniformity, RSS-U)的结果相比仍相差 6.5%左右.其原因在于业务迁移过程中业务连接表更新等操作造成一定开销.图 11(b)为各队列吞吐量实时变化情况,可看出由于出现过载情况,队列 6 接连出现 2 次业务迁移操作,使得队列 1 和队列 2 出现速率瞬时增加情况,最终达到处理饱和状态.

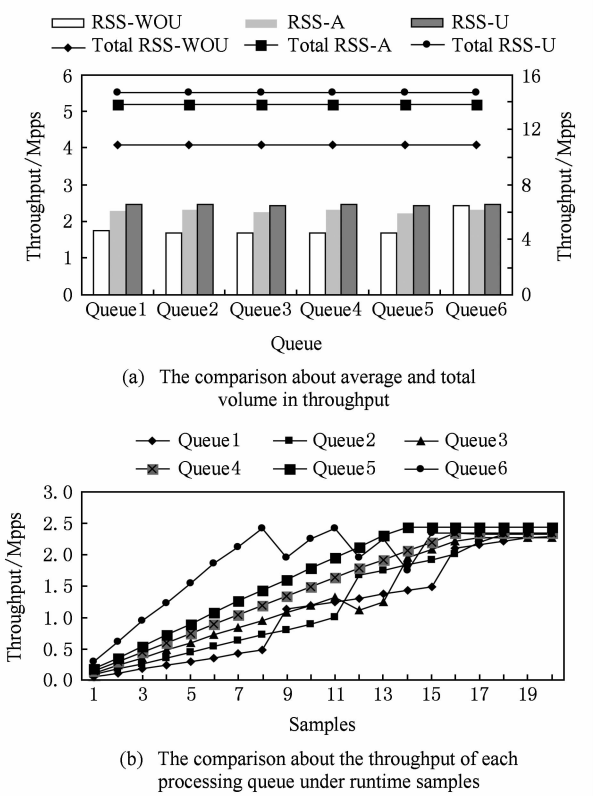


Fig. 11 The network throughput performance with or without the HVLB RSS-A mechanism

图 11 应用 HVLB RSS-A 策略前后吞吐量性能对比

5.4 NF 选择及调度策略性能分析

本节将对调度策略对网络性能的影响进行评估,实验中选择轮询策略(RR)、Maglev-Hash 策略[6]来和 HVLB 策略进行对比,比较吞吐量及平均往返时延(RTT)变化情况.实验采用 Thrulay[30]工具产生 60 条 TCP 业务流,经过 HVLB 处理后分发至 $S_2\sim S_6$ 上的 10 个 NF,转发器和 NFs 间存在 5 条链路,链路固有 RTT 为 2.2 ms,每条链路最大带宽为 600 Mbps,转发器配置 6 个处理队列,参考 5.1.2 节的结果,转发器处理上述业务数据不会产生性能瓶颈,引入处理延迟约为 0.2 ms.实验在 2 种情况下进行:1)稳态情况,HVLB 和 NF 之间链路无其他流量干扰,同时 NFs 上不运行其他程序;2)动态变化情况,用限流工具对 S_2,S_4 和 HVLB 间链路增加 5 ms 延迟,在 S_3,S_5 中 NF 所属虚拟机上通过 Sysbench[31]工具增加负载,使 CPU 占用率维持在 75%~80%.

实验结果如图 12 和 13 所示,图 12(a)和 12(b)中,横轴为时间,纵轴为测试业务流的总吞吐量;图 13(a)和 13(b)中,横轴为 RTT 值,纵轴为测试业务流 RTT 均值概率分布.可以看出,在稳态情况下,3 种调度策略下总吞吐量和 RTT 均值变化平稳,总量维持在 2 820 Mbps 左右,RTT 均值维持在 2.5 ms 左右;动态变化情况下,RR 策略对应总吞吐

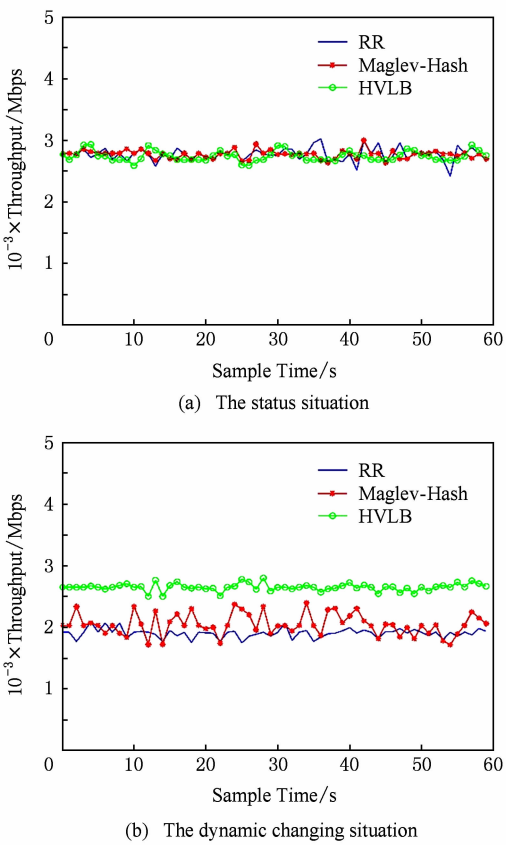


Fig. 12 Effect on network throughput under different scheduling strategies

图 12 不同调度策略下吞吐量性能对比

量值为 1 901.9 Mbps,且抖动剧烈;平均 RTT 延迟为 14.9 ms;Maglev-Hash 策略对应总吞吐量值为 2 044.5 Mbps,抖动较之 RR 策略更为剧烈,RTT 延迟约为 11.9 ms. 分析其原因在于 RR 策略固定地选择链路状况不佳或计算资源竞争严重的 NF,其网络性能受到影响较大;Maglev-Hash 采取均匀选择目标 NF 方式,性能受到影响较小. 采用 HVLB 策略后吞吐量为 2 648.7 Mbps,相比 RR 和 Maglev-Hash 策略分别提升 39.3%和 29.6%,且波动平缓;RTT 均值为 6.8 ms,比另 2 种策略分别降低 54.4%和 42.9%. 上述结果表明,HVLB 的调度策略综合考虑 NF 传输和计算能力,在保障网络性能的前提下制订准确的 NF 选择策略和分发比例.

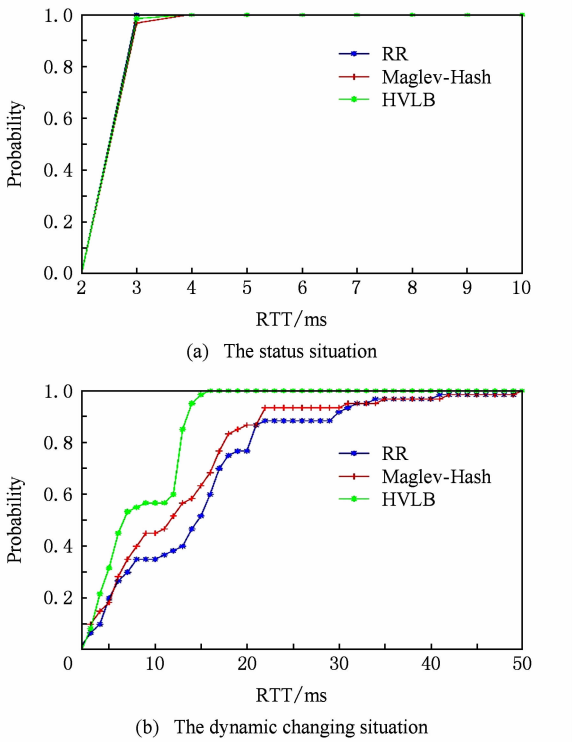


Fig. 13 CDF for latency under different scheduling strategies

图 13 不同调度策略下延迟性能对比

6 结 论

NFV 为电信运营商提供了低成本、灵活高效的业务实施方式. 作为其重要功能组件,负载均衡系统可实现准确的业务分发处理,从而提供有力的服务质量保障. 本文设计实现了面向 NFV 的高性能负载均衡机制及系统 HVLB,实现调度策略制订和数据转发的有效分离,在转发端基于用户空间实现多核多队列高效数据处理架构,保证各处理队列之间的数据访问隔离和任务处理均衡;在控制端将网络链路和计算相结合的综合能力作为目标 NF 选择和调度策略制订依据,在业务准确分发的基础上保障了网络性能. 下一步研究工作将着眼于系统的容量扩展管理、容错处理方面的改进,并进一步在大规模数据中心环境下进行部署与试商用.

参 考 文 献

[1] ETSI-GS-NFV-002. 2014. Network functions virtualization (NFV): Architectural framework [OL]. [2017-10-28]. http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf

[2] ETSI-GS-NFV-003. 2014. Network functions virtualisation: Terminology for main concepts in NFV [OL]. [2017-07-12]. https://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_NFV003v010201p.pdf

[3] Quinn P, Nadeau T. Service function chaining problem statement [OL]. Internet-Draft: IETF Secretariat, 2014 [2017-08-20]. https://www.ietf.org/archive/id/draft-ietf-sfc-problem-statement-10.txt

[4] Mijumbi R, Serrat J, Gorricho J, et al. Network function virtualization: State-of-the-art and research challenges [J]. IEEE Communications Surveys and Tutorials, 2016, 18(1): 236-262

[5] ETSI-GS-NFV-SWA-001. 2014. Network functions virtualisation (NFV): Virtual network functions architecture [OL]. [2017-01-08]. https://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_nfv-swa001v010101p.pdf

[6] Eisenbud D E, Yi C, Contavalli C, et al. Maglev: A fast and reliable software network load balancer [C] //Proc of ACM NSDI. New York: ACM, 2016: 523-535

[7] Patel P, Bansal D, Yuan L, et al. Ananta: Cloud scale load balancing [C] //Proc of ACM SIGCOMM 2013. New York: ACM, 2013: 207-218

[8] Intel. DPDK: Intel data plane development kit [OL]. [2016-12-27]. http://www.dpdk.org

[9] LVS. Linux Virtual Server [OL]. [2016-12-27]. http://www.linuxvirtualserver.org

[10] A10 Network. AX Series [OL]. [2017-02-11]. http://www.a10networks.com

[11] Array Networks. Array networks [OL]. [2017-02-12]. http://www.arraynetworks.com

[12] F5. BIG-IP [OL]. [2017-02-17]. http://www.f5.com

[13] Barracuda. Load balancer application delivery controller [OL]. [2017-02-23]. http://www.barracuda.com

[14] Load balancer. org Virtual Applicance [OL]. [2017-02-23]. http://www.loadbalancer.org

[15] Haproxy. Haproxy Load Balancer [OL]. [2017-02-27]. http://www.haproxy.org

[16] Nginx. Nginx [OL]. [2017-04-25]. http://www.nginx.org

[17] Gandhi R, Liu Hongqiang, Charlie H, et al. Duet: Cloud scale load balancing with hardware and software [C] //Proc of ACM SIGCOMM 2014. New York: ACM, 2014: 27-38

[18] Kang N, Ghobadi M, Reumann J, et al. Efficient traffic splitting on commodity switches [C] //Proc of ACM CoNEXT. New York: ACM, 2015: 58-70

[19] Xu Fei, Liu Fangming, Jin Hai, et al. Prolog to managing performance overhead of virtual machines in cloud computing: A survey state of the art, and future directions [J]. Proceedings of the IEEE, 2014, 102(1): 11-31

[20] Li J, Sharma N K, Ports D R, et al. Tales of the tail: Hardware, OS, and application-level sources of tail latency [C] //Proc of ACM SoCC 2014. New York: ACM, 2014: 1-14

[21] Dong Yaozu, Yang Xiaowei, Li Xiaoyong, et al. High performance network virtualization with SR-IOV [C] //Proc of the 16th Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2010: 1471-1480

[22] Makineni S, Iyer R, Sarangam P, et al. Receive side coalescing for accelerating TCP/IP processing [C] //Proc of the 13th Int Conf on High Performance Computing. Berlin: Springer, 2006: 289-300

[23] Intel. Ethernet Flow Director [OL]. [2017-02-17]. https://www.intel.com/content/www/us/en/ethernet-products/ethernet-flow-director-video.html

[24] Han Sangjin, Jang Keon, Park KyoungSoo, et al. Packetshader: A GPU-accelerated software router [C] //Proc of the ACM SIGCOMM Conf. New York: ACM, 2010: 195-206

[25] Hwang J, Ramakrishnan K K, Wood T, et al. NetVM: High performance and flexible networking using virtualization on commodity platforms [C] //Proc of Networked Systems Design and Implementation. New York: ACM, 2014: 445-458

[26] Wang G, Ng T. The impact of virtualization on network performanceof Amazon ec2 data center [C] //Proc of the 29th IEEE Int Conf on Computer Communications. Piscataway, NJ: IEEE, 2010: 1-9

[27] Shea R, Wang Feng, Wang Haiyang, et al. A deep investigation into network performance in virtual machine based cloud environments [C] //Proc of the 33rd IEEE Int Conf on Computer Communications. Piscataway, NJ: IEEE, 2014: 1285-1293

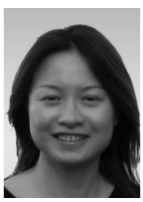
[28] Xu Yunjing, Musgrave Z, Noble B D, et al. Bobtail: Avoiding long tails in the cloud [C] //Proc of Networked Systems Design and Implementation. New York: ACM, 2013: 329-341

[29] GitHub. Traffic generator powered by DPDK [OL]. [2017-09-20]. https://github.com/Pktgen/Pktgen-DPDK

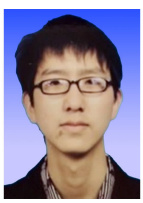
[30] Sourceforge. Thrulay-ng [OL]. [2017-10-12]. http://thrulay-ng.sourceforge.net/

[31] Sourceforge. Sysbench [OL]. [2017-10-22]. http://sysbench.sourceforge.net/

Wang Yuwei, born in 1980. PhD candidate. Senior engineer in the Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include future network, virtualization technology and cloud computing.



Liu Min, born in 1976. Professor in the Institute of Computing Technology, Chinese Academy of Sciences. Her main research interests include mobile management, network measurement and mobile computing.



Ma Cheng, born in 1989. Master candidate in the Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include cloud computing, next generation network and mobile computing.

Li Pengfei, born in 1992. Master candidate in the Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include cloud computing, next generation network and mobile computing.