doi:10.11959/j.issn.1000-436x.2016161

面向网络性能优化的虚拟计算资源调度机制研究

王煜炜[1,2]，刘敏[1]，房秉毅[3]，秦晨翀[1,2]，闫小龙[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；3. 中国联合网络通信集团有限公司，北京 100032）

摘 要：针对基于 Xen 的 vCPU 调度机制对虚拟机网络性能的影响进行了深入研究和分析。提出一种高效、准确、轻量级的网络排队敏感类型虚拟机（NSVM）识别方法，可根据当前虚拟机 I/O 传输特征将容易受到影响的虚拟机进行准确识别和区分。进而设计一种新型虚拟计算资源调度和分配机制 Diff-Scheduler，将不同类型虚拟机的 vCPU 实施分池隔离调度，同时提高 NSVM 类型虚拟机 vCPU 的调度频率。原型系统实验结果表明，相比 Xen 默认的调度机制，Diff-Scheduler 能够大幅提高虚拟机网络性能，同时保证计算资源分配的公平性。
关键词：云计算；虚拟化；vCPU 调度；网络排队敏感型；虚拟计算资源分配
中图分类号：TP302 **文献标识码**：A

Study on virtual computing resource scheduling for network performance optimization

WANG Yu-wei[1,2], LIU Min[1], FANG Bing-yi[3], QIN Chen-chong[1,2], YAN Xiao-long[1,2]
(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China;
2. University of Chinese Academy of Sciences, Beijing 100049, China; 3. China Unicom, Beijing 100032, China)

Abstract: A deep insight into the relationship between vCPU scheduling and I/O transmit in Xen was provided. Then an effective and lightweight recognition method, through which could identify the so-called NSVM (network queuing sensitive virtual machine) that was more vulnerable to the congestion in I/O transmit was put forward. Furthermore, a novel mechanism for resource assignment and scheduling called Diff-Scheduler was proposed. It could schedule the vCPU of the NSVM more frequently than other VM in different pools independently. Evaluations based on a prototype of Xen platform featured Diff-Scheduler show that the proposed mechanism significantly improves the network performance of VM. Specifically, comparing with the default mechanism of Xen, Diff-Scheduler proposed jointly enhances throughput, latency remarkably and ensures the fairness of resource allocation at the same time.
Key words: cloud computing, virtualization, vCPU scheduling, network queuing sensitive, virtual computing resource assignment

1 引言

当前，云计算作为新兴的计算服务方式得到了产业界、学术界的广泛关注。亚马逊[1]、谷歌[2]、微软[3]以及国内的阿里巴巴[4]、中国联通[5]、中国移动[6]、中国电信[7]等企业和电信运营商均建立了自己的公有云和私有云平台，从而可以为用户提供高效、安全、灵活的计算服务。随着以大数据为代表的分布式计算框架日益普及，大量执行复杂且网络吞吐量密集任务的应用已经移植到云平台。

虚拟化作为云计算的重要核心技术使多个用户可以高效、隔离地使用物理机资源。通常情况下，云计算服务提供商会将资源按照用户需求以虚拟机（VM, virtual machine）的形式租用给用户。不同用户的虚拟机以资源共享的方式占用同一物理服务器的 CPU、内存、外设存储以及网络等资源。云

收稿日期：2016-01-25；**修回日期**：2016-06-23
基金项目：国家自然科学重点基金资助项目（No.61132001, No.61120106008, No.61472402, No.61472404, No.61272474, No.61202410）
Foundation Item: The National Natural Science Foundation of China (No.61132001, No.61120106008, No.61472402, No.61472404, No.61272474, No.61202410)

计算服务商主要依托虚拟化监管软件 Hypervisor 实现对各个虚拟机的资源管理。典型的 Hypervisor 包括 Xen、KVM、Hyper-V、ESX-*i* 等，其中，Xen 作为主流的开源虚拟化平台，已被国内外众多云计算服务提供商广泛采用[8]。

然而，越来越多的研究表明[9~12]，虚拟化技术的引入给虚拟机网络性能带来了较大的负面影响。多个虚拟机共享物理服务器硬件资源，使采用虚拟化技术创建的虚拟机并不能完全提供等同于传统物理机的特性。一方面，对于经常执行大量网络应用密集型通信任务的 VM 而言，执行通信任务需要和执行计算密集型任务的 VM 竞争计算资源；另一方面，虚拟化平台提供的虚拟计算资源（vCPU, virtual CPU）调度机制未能充分考虑 2 种任务之间的竞争给网络性能带来的干扰，进而造成虚拟机网络性能受到显著影响，具体表现在网络吞吐量降低、延迟增大等。当物理主机承载的计算负载较高时，此类影响尤为强烈。

本文针对基于 Xen 的 vCPU 调度机制对虚拟机网络性能的影响进行了深入研究，通过实验验证可知，vCPU 调度机制带来的虚拟机 I/O 共享环阻塞是造成其网络性能下降的重要原因。为了保障网络性能，需要解决以下几个问题：1) 明确 vCPU 调度机制干扰 hypervisor 的 I/O 通信系统的主要方式和影响要素；2) 如何从所有虚拟机中选择出对此类干扰敏感且受影响严重的虚拟机类型，进而进行区分式的调度处理；3) 如何在不违背云计算资源公平共享原则的基础上对计算资源进行重新合理调配，从而有效减少网络性能受到的影响[12]。

克服上述 3 个问题存在较大的技术挑战。vCPU 调度对网络性能干扰的根本原因在于虚拟资源共享服务模式本身。因此，在现有基于 Xen 的虚拟化平台中，设计实现既能保证计算资源分配公平同时又能为执行网络密集型任务的虚拟机提供区分式的调度策略面临很大的挑战。

本文围绕面向网络性能优化的虚拟资源调度机制展开研究，基于上述核心问题及挑战提出了有针对性的解决方案，主要创新贡献如下。

1) 通过实验验证对 vCPU 机制影响虚拟机网络传输过程进行了详细分析，进一步明确 I/O 共享环中的请求项队列长度变化是表征数据传输阻塞程度的重要指标。

2) 基于虚拟机接收和发送数据方向上 I/O 共享环溢出、过载及停滞特征，提出一种高效、轻量级的虚拟机类型识别方法，把虚拟机识别为网络排队敏感型虚拟机（NSVM, network queuing sensitive virtual machine）和非网络排队敏感型虚拟机（NNVM, non-network queuing sensitive virtual machine）2 种类型。前者较后者更容易受到 vCPU 调度机制引起的资源竞争的影响。

3) 提出了一种面向网络性能优化的新型虚拟计算资源分配及调度机制 Diff-Scheduler，该机制优化并改进了 Xen 原有的 vCPU 调度机制，将 NSVM 与 NNVM 类型虚拟机的 vCPU 分池隔离与区分调度。

4) 基于 Xen 4.4 研发了原型系统并进行性能对比实验。实验结果表明，相关机制能够大幅提高虚拟机网络性能，并保证计算资源分配的公平性。

2 相关工作

诸多相关研究[11~18]表明，虚拟机网络性能的下降与所属虚拟化平台提供的 vCPU 调度机制有关。文献[11]针对虚拟化技术对 Amazon EC2 虚拟机网络性能的影响进行了研究，并对延迟、吞吐量及分组丢失率指标进行了测量。文中指出在数据中心网络处于非拥塞的情况下，虚拟化技术的引入会导致网络吞吐量及传输延迟等指标显著下降。但文中并没有给出影响虚拟机网络性能的具体原因。Shea 等研究者在文献[12]中指出，虚拟机同时执行通信和计算任务的“双重身份”是导致虚拟机网络性能下降的根本原因。Xen 中的 vCPU 调度机制影响了虚拟机内部 CPU 对网络相关任务的处理，进而导致了虚拟机网络性能下降。文中通过改变 vCPU 调度周期来缓解网络性能影响，并通过简单的 CPU 隔离机制验证其有效性。Gamage[13]和 Kangarlou[14]针对 TCP 应用的发送过程和接收过程，在特权域 Domain 0 增加代理处理模块，由其代替 VM 对相关 TCP 连接和数据分组传输进行处理，从而保证发送/接收窗口大小，提高 TCP 应用的吞吐量。上述做法在提高 TCP 传输性能的同时给 Domain 0 域带来较大的处理和存储负担，且针对其他类型数据分组，如 UDP 则无法进行加速处理。Xu[15]提出将 VM 分为 LSVM 和 NLSVM 区分对待，并将 LSVM 的调度时间进一步划分成若干小的时间片，从而提高了 LSVM 的调度频率，保障了 LSVM 的网络性能。

但是，文中并未明确如何具体识别 LSVM 和 NLSVM，在 VM 较多的物理 CPU 共享队列，LSVM 仍需要等待较长时间才能赢得 CPU 时间片。Xu[16~18] 提出以用户为中心的 VM 传输延迟优化方案，文中针对 VM 和 VM 之间的传输延迟进行测量和分析，并给出了一种轻量级的测量 VM 受到竞争干扰的方法，用户可以直接对所租用的各个 VM 当前网络性能进行评估，并且按照评估结果将应用任务进行分配。该方法一定程度上避免了虚拟化技术对 VM 网络延迟性能的影响，但以用户为中心的方案并未从根本上解决上述性能下降问题，若不同的用户同时采用上述策略，则会造成 VM 网络性能的迅速下降。

上述方案从不同角度提出了改善虚拟机网络性能的方法，但是尚未有方案能够完整解决引言中提到的 3 个主要问题，并提出针对性的高效处理策略。

3　vCPU 调度对 I/O 通信影响分析

本文从分析 vCPU 调度机制对 Xen 中虚拟机 I/O 通信机制的影响入手，基于 I/O 传输特征对 VM 类型进行合理、准确识别，提出一种面向网络性能优化的虚拟计算资源分配及调度机制 Diff-Scheduler，可大幅度提升虚拟机的网络性能。

3.1　基于 Xen 的 vCPU 调度机制

vCPU 是由 Hypervisor 分配给 VM 的虚拟 CPU，是虚拟化系统对其物理 CPU 资源的抽象。系统运行过程中，VM 分得的 vCPU 将在各个物理 CPU 核上形成等待调度的 vCPU 队列。vCPU 调度是指 Hypervisor 挑选 vCPU 并分配其一段时间物理 CPU 使用权限的过程。合理的 vCPU 调度机制是 VM 高效执行任务的关键。目前，Xen 的稳定版本默认采用 Credit Scheduler 调度机制[19]，调度器依据 Weight 值为每个虚拟机分配其 Credit，同时为每个物理 CPU 维护一个 vCPU 队列。当 vCPU 没有被调度时，就处于非活跃状态。直至其下一次被调度时，才能重新处于活跃状态。对于频繁执行网络通信任务的 VM 而言，如果其 vCPU 长时间处于非活跃状态，将会大大影响其网络通信性能，下面将结合 Xen 的 I/O 通信机制进行详细分析。

3.2　基于 I/O 共享环的网络数据传输

在 Xen 中，VM 的网络通信通过位于其内部的前端设备和位于特权域 Domain 0 内部的后端设备交互实现，如图 1 所示，设备驱动采用前后端分离模型，分为前端驱动与后端驱动 2 部分。前端驱动位于 VM 域，后端驱动位于特权域 Domain 0。前后端驱动通过 Xen 提供的共享内存机制进行通信。VM 发送和接收数据过程均对应一个大小固定的共享 I/O 环型存储队列，用于存放数据分组收发过程的控制信息。其内部存放着 2 类数据结构：请求（request）项与响应（response）项。请求项与响应项分别存入共享 I/O 环并形成请求队列与响应队

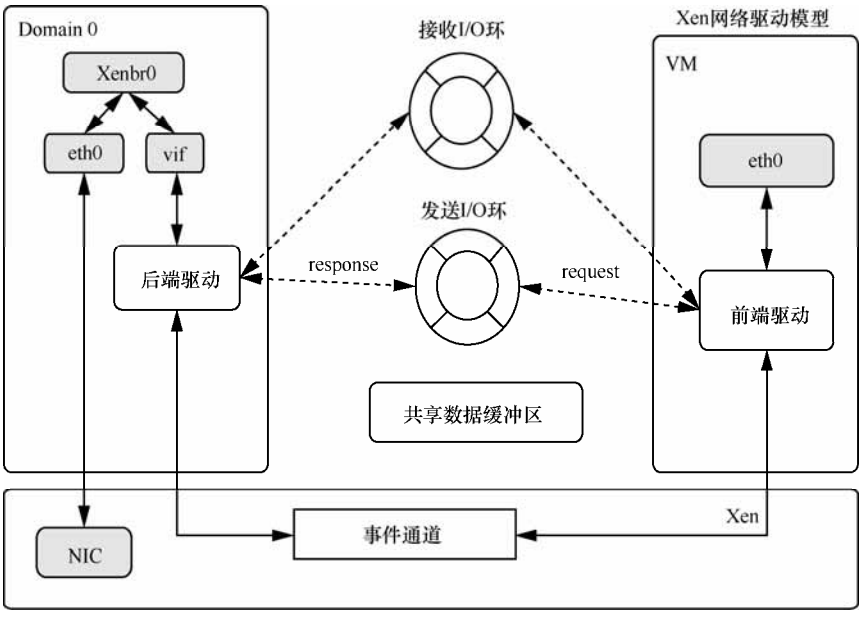


图 1　基于 Xen 的虚拟网络 I/O 通信架构

列，且响应队列队尾总是紧跟于请求队列队首之后。对应发送和接收过程，相关的控制项包括“发送请求”（sending request）、“发送响应”（sending response）、“接收请求”（receiving request）、“接收响应”（receiving response）4 种类型。

数据分组的收发与生产者和消费者之间传递商品的过程类似，以 VM 接收数据分组为例，首先，位于 Domain 0 的后端驱动从接收 I/O 环中取出“接收请求”，并将该数据分组存入共享内存缓冲区域；同时，依据存有该数据分组的共享内存页 ID 及数据偏移量等信息生成“接收响应”，并将其放入接收 I/O 环指定位置；然后，通过事件通道通知前端驱动，前端驱动将“接收响应”从 I/O 环中取出，并将从共享内存缓冲区中接收到的数据分组交给虚拟机操作系统协议栈继续处理。发送过程和上述接收过程类似。

3.3 vCPU 调度对数据传输影响分析

上文中提到，VM 的网络性能受 vCPU 调度机制的影响。本文通过研究发现，共享 I/O 环中请求项队列长度是表征虚拟机当前网络传输拥塞状况的重要因素。在数据传输过程中，共享 I/O 环中请求项队列长度会随 vCPU 队列等待延迟发生相应变化，进而影响虚拟机的网络性能，这是影响虚拟机网络性能的重要原因。下面通过具体实验分别对 vCPU 不需排队以及 vCPU 平均等待队列长度为 1 的 2 种场景下 VM 吞吐量与请求项队列长度变化关系进行验证分析。其中，vCPU 不需排队是指在宿主机计算资源充足，vCPU 无需排队等待可直接获取 CPU 时间片的情况，简称 Queue-0 场景；vCPU 平均队列长度为 1 指的是 vCPU 平均需要等待 1 个调度周期（Xen 默认为 30 ms）才能获取 CPU 时间片的情况，简称 Queue-1 场景。

为获取相关信息，本文对 Xen 虚拟网络设备后端驱动进行修改，在用户空间获取 I/O 共享环中的请求项队列长度值。该值是个不大于 256 的非负数。当共享 I/O 环“溢出”时，测得的请求队列长度已不能准确代表其真实长度，本文将其校准为最大值 256。这里的溢出并不是真正意义上的内存堆栈溢出，而是对应请求项、响应项队列长度加上本次待放入的请求项数量后大于等于最大值 256 的情况。此时表明 I/O 共享环发生了比较严重的拥塞，实际情况并不会出现真正的堆栈溢出，Xen 系统设计了自适应调节机制，采取相关退避或等待策略。

实验环境如下：网络拓扑环境由 2 台服务器及连接它们之间的路由设备组成，如图 2 所示。其中宿主机服务器 S 上部署 Xen 4.4 系统及本文所提的相关 vCPU 优化调度功能模块；另一台服务器 T 作为测试对端，每台服务器均配有 8 核物理 CPU、32 GB 内存和 2 块千兆局域网网卡。Domain 0 特权域和测试用虚拟机内运行的操作系统均为 Ubuntu14.04，内核版本分别为 Linux 3.13.11 和 Linux 3.13.0。

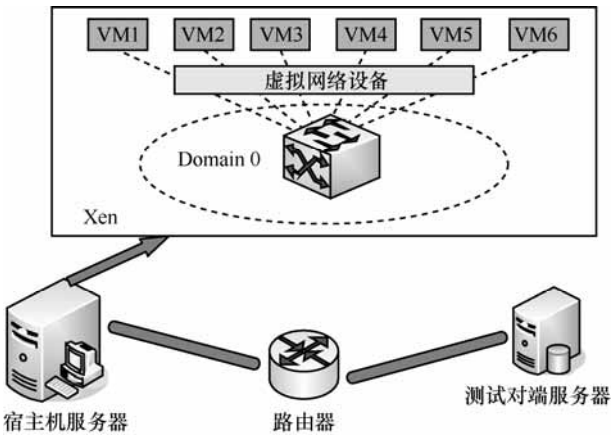


图 2 实验网络拓扑环境

在宿主机服务器 S 上，除 Domain 0 之外共建立 6 个虚拟机，分别命名为 VM1~VM6。虚拟机之间通过虚拟交换设备相连，进而通过千兆高速局域网与测试对端服务器 T 连接。各 VM 内分别安装 Iperf[20]与 Sysbench[21]工具，用来生成 TCP、UDP 等业务流和计算负载。同时可用 Iperf 对吞吐量、分组丢失率和流量进行监测分析。

具体测试参数如下：Domain 0 预分配了 2 个物理 CPU 与 10 GB 内存空间。VM1~VM6 各自被预分配 2 GB 内存；实验中采样周期 T_m 为 10 ms，识别周期 T_s 为 100 ms。

实验中默认将 2 个 CPU 预分配给 Domain 0 域。Queue-0 场景下 VM1~VM6 分别被预分配 1 个 vCPU；Queue-1 场景下，VM1~VM6 分别被预分配 2 个 vCPU，因此，平均等待队列长度为 1。本文在 VM1 上用 Iperf 工具生成 TCP 和 UDP 业务流，同时在 VM1~VM6 上用 Sysbench 工具生成 100%计算负载；然后，通过实验观察发送和接收方向上 I/O 请求项队列长度与 TCP、UDP 业务流吞吐量变化关系

及对应 I/O 共享环溢出情况。实验结果如图 3~图 8 所示。

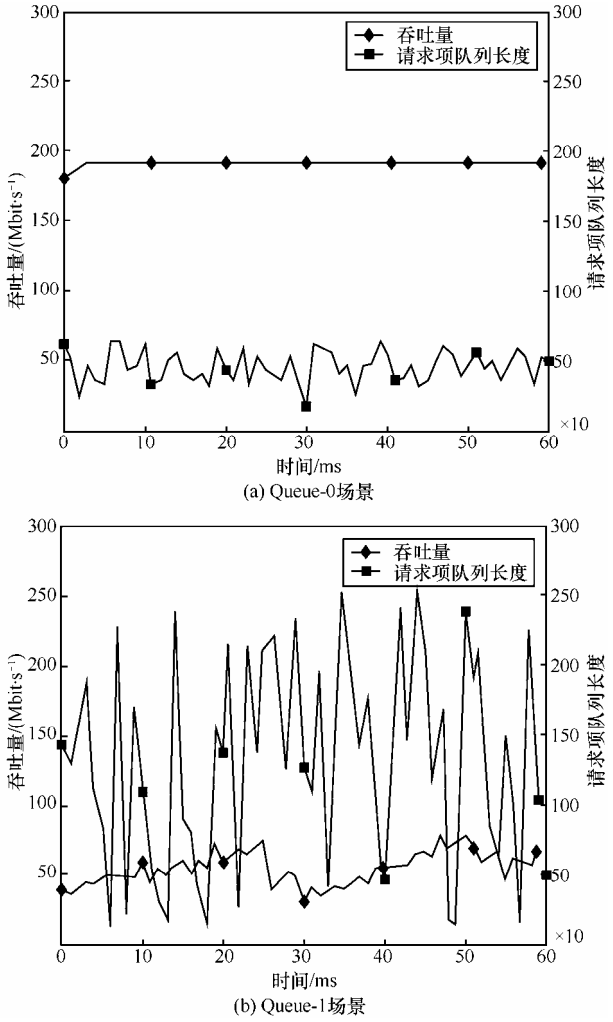


图 3　TCP 吞吐量与请求项队列长度变化关系（接收方向）

数据分组接收方向上，图 3 和图 4 分别对应 Queue-0 与 Queue-1 这 2 种场景下，虚拟机 TCP、UDP 业务流吞吐量与 VM 请求项队列长度的变化关系。从图 3(a)中可以看出，在 Queue-0 场景下，TCP 业务吞吐量稳定在 190 Mbit/s。请求项队列平均长度在 35~55 范围内小幅度平稳变化；相应地，如图 3(b)所示，在 Queue-1 场景下，TCP 业务平均吞吐量降幅达 70.8%，达到 55.5 Mbit/s，且波动变化剧烈。请求队列项平均长度约为 150，其值同样呈现大幅振荡情况。

图 4 分别对应 Queue-0 和 Queue-1 这 2 种场景下，UDP 业务流吞吐量和请求项队列平均长度之间的对应变化关系。2 种场景下，吞吐量的变化趋势与 TCP 类似，对应 Queue-0 场景为 194 Mbit/s，对应 Queue-1 场景为 68 Mbit/s；然而，请求项队列平均长度分别为 48 和 29，呈现出和 TCP 不同的情况，这是因为在 Queue-1 场景下发生了更多的 I/O 共享环拥塞，部分请求项累计在 I/O 环中等待处理。

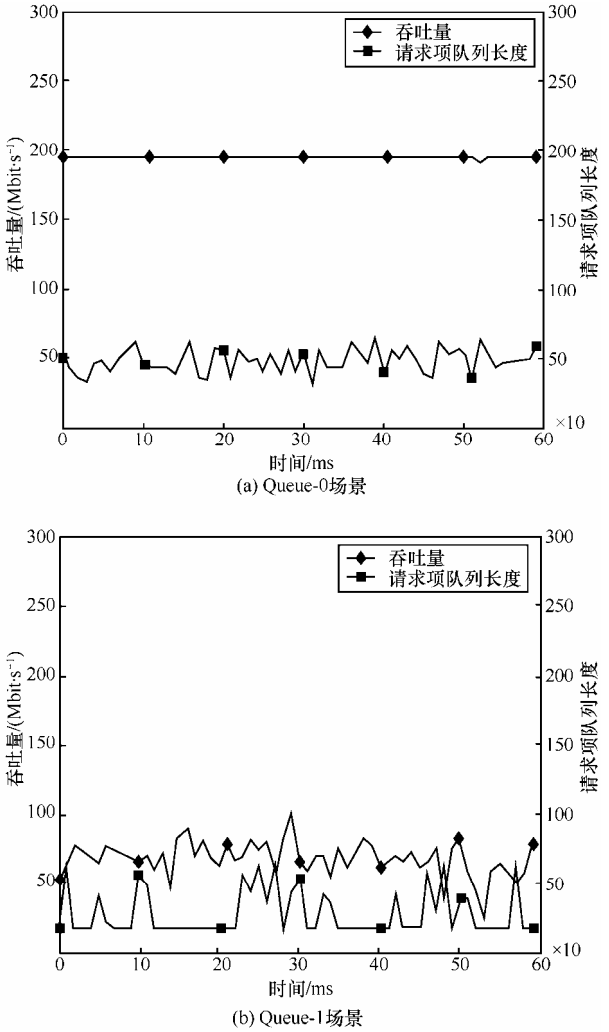


图 4　UDP 吞吐量与请求项队列长度变化关系（接收方向）

图 5 分别对应数据分组接收方向上，TCP 和 UDP 业务对应 Queue-0 和 Queue-1 这 2 种场景下 I/O 环的溢出情况（1 表示溢出，0 表示不溢出），从图 5 可看出，Queue-1 场景下 I/O 环产生了多次溢出情况，且对于 UDP 而言，其产生溢出的频率高于 TCP 业务，几乎接近于 100%。

通过对上述实验结果分析可知，在数据分组接收方向上，VM 请求项不能得到及时处理，因而长期维持在一定数量规模，其值在消耗殆尽与突然大量补充 2 种状态之间频繁交替切换，进而造成了 VM 网络性能的下降。

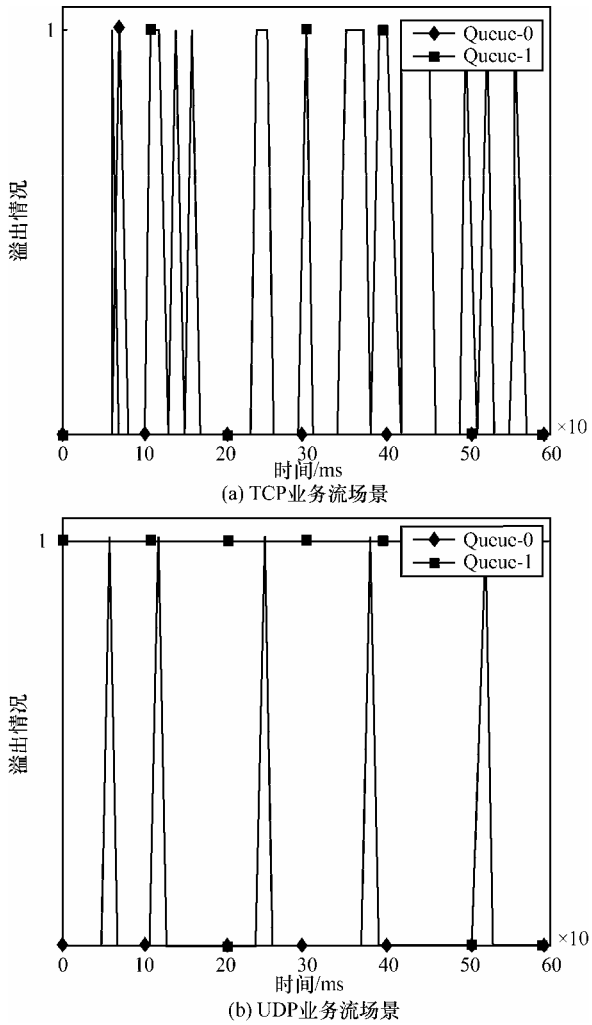


图 5　I/O 共享环溢出情况（接收方向）

数据分组发送方向上，图 6 和图 7 分别对应 Queue-0 与 Queue-1 这 2 种场景下，虚拟机 TCP、UDP 业务流吞吐量与请求项队列长度的关系。从图 6(a)可以看出，在 Queue-0 场景下，TCP 业务流吞吐量稳定在 190 Mbit/s 左右，且对应请求项队列长度平均值稳定在 5 左右，这说明由于该场景下 VM 的计算资源较为充足，发送请求项能得到及时处理；对应 Queue-1 场景，如图 6(b)所示，请求项队列长度间断重复出现长度为 0 的情况，统计不为 0 情况的次数为 20，其不为 0 的有效平均长度为 50.7。同时，TCP 平均吞吐量为 87.6 Mbit/s，降幅达 53.9%。

图 7 分别对应 Queue-0 和 Queue-1 这 2 种场景下，UDP 业务吞吐量和请求项队列平均长度之间对应变化关系。类似于 TCP 情况，从图 7(a)中可看出，在 Queue-0 场景下，UDP 业务流吞吐量稳定在 194 Mbit/s 左右，且对应请求项队列长度平均值同样稳定在 5 左右；相应地，Queue-1 场景的情况如图 7(b)所示，请求项队列长度间断重复出现长度为 0 情况，统计不为 0 情况的次数为 18，其不为 0 的有效平均长度为 53.7。同时，对应 UDP 吞吐量为 77.6 Mbit/s，降幅达 60%。

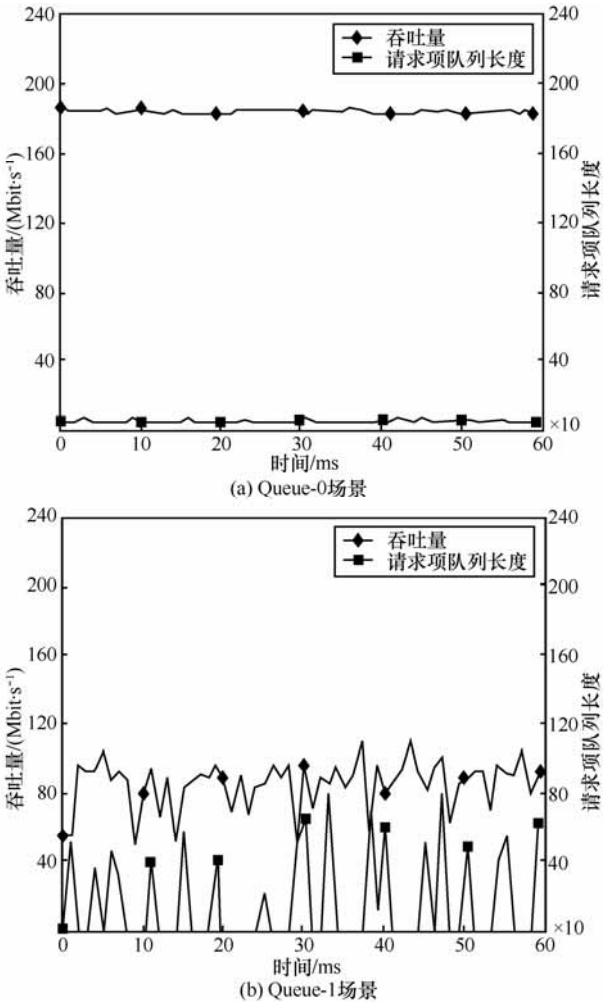


图 6　TCP 吞吐量与请求项队列长度变化关系（发送方向）

图 8 分别对应发送方向上，TCP 和 UDP 业务在 Queue-0 和 Queue-1 这 2 种场景下 I/O 环溢出情况（1 表示溢出，0 表示不溢出），从图 8 可以看出，在发送方向上，由于 VM 受到 vCPU 排队影响无法及时处理发送数据分组的请求，因此，基本上不会造成 I/O 共享环的溢出情况。

通过对上述实验结果分析可知，在数据分组发送方向上，VM 前端驱动发送数据分组的任务受到 vCPU 排队影响而发生间断性阻塞，造成无法及时处理相关发送数据分组的请求，因而请求队列项出现了间断性重复为 0 的情况，因此，造成了网络性能的下降。

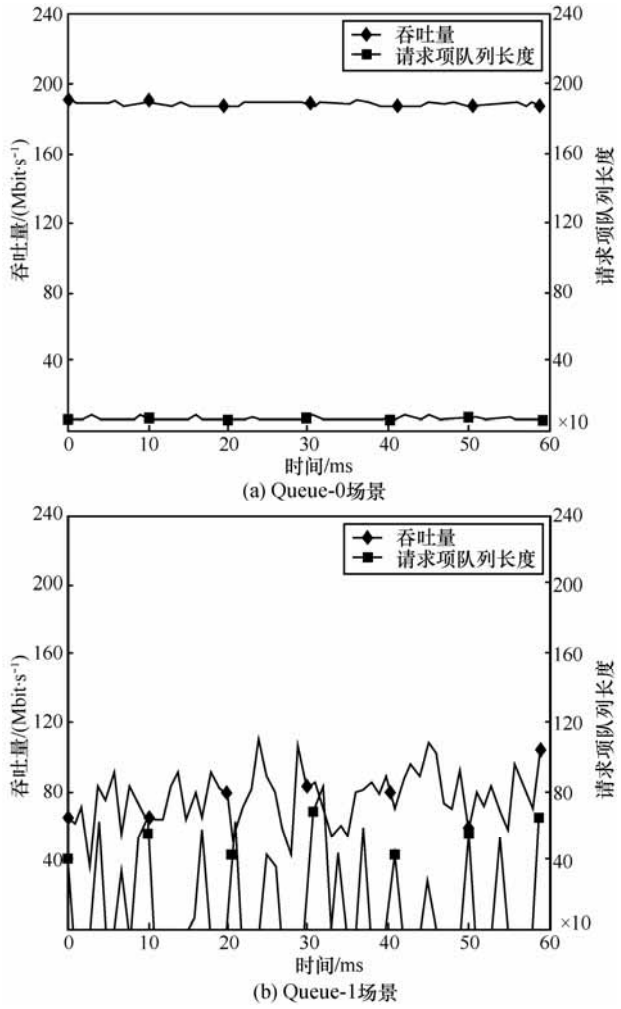


图 7　UDP 吞吐量与请求项队列长度变化关系（发送方向）

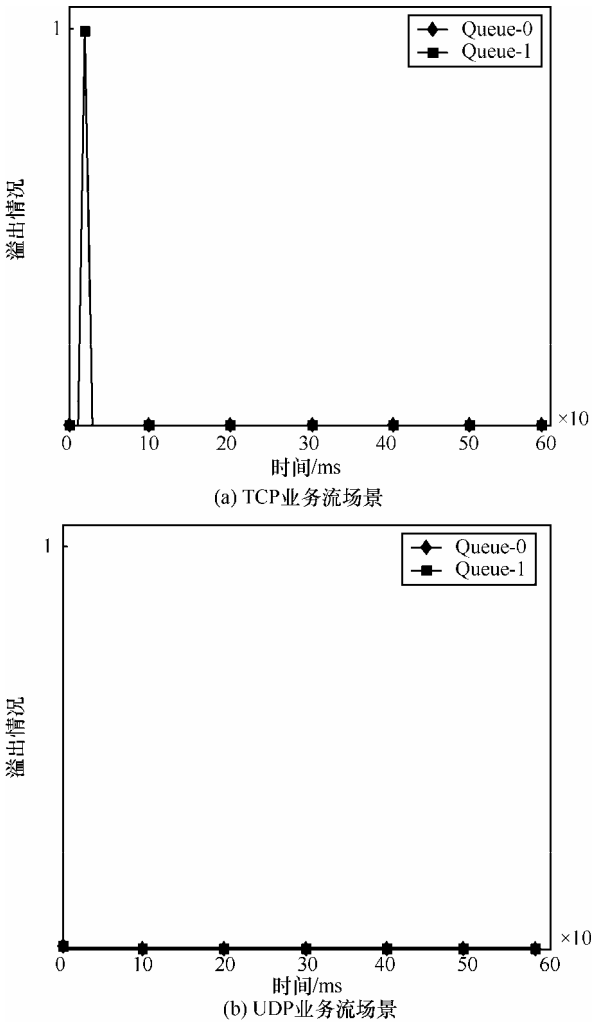


图 8　I/O 共享环溢出情况（发送方向）

综合上述接收和发送方向的分析可知，Xen 默认的 vCPU 调度机制产生的排队等待导致了 vCPU 交替处于活跃与非活跃状态，从而对虚拟网络设备前后端驱动之间的数据交互过程产生干扰。该干扰使虚拟机无法及时从 Domain 0 接收数据和发送数据。通过多次实验可以定量分析出网络性能受限和 VM 请求项队列长度变化之间的关系，进而提出 VM 类型识别方法。

4　基于 I/O 传输特征的 VM 类型识别

4.1　VM 类型及通信状态定义

根据虚拟机的 I/O 传输特征，本文将 VM 分为 2 种类型：NSVM 和 NNVM。其中，NSVM 指的是执行网络通信任务密集且网络性能容易受到 vCPU 调度排队影响的虚拟机，此类虚拟机大多承载了较密集的通信任务；而 NNVM 主要承载的是计算密集型任务，且受到 vCPU 排队调度的影响较轻。为了更好地识别虚拟机类型，定义如下通信状态。

1) 溢出状态（overflow state）：VM 当前 I/O 环请求项、响应项队列长度加上本次待放入的请求项数量后大于等于最大值 $QL_{\max}$，且频率非常频繁，$QL_{\max}$ 值为 256。

2) 过载状态（overload state）：VM 当前 I/O 环未处于溢出状态，但是经常大于一定阈值 QL_{th}，且频率非常频繁。其中，QL_{th} 为 I/O 请求项队列长度过载阈值，本文默认为 150，可根据实际情况调整。

3) 正常状态（normal state）：VM 的 I/O 请求项队列长度介于 0 和 QL_{th} 之间，通信未受到明显影响。

4) 停滞状态（stagnate state）：VM 的 I/O 请求项队列长度在测量周期内交替频繁重复出现零和非零值情况，多数对应 VM 发送数据的场景。

4.2　VM 类型识别

VM 类型识别通过监测 VM 的 I/O 请求队列项长度变化和统计分析 VM 的通信状态来完成，假设当前物理宿主机 S 中存在 N 个 VM，用 $V_i(i=1,2,\cdots,N)$

表示。其中，NSVM 类型的 VM 个数记作 N_{ns}；NNVM 类型的 VM 个数记作 N_{nn}。M 表示 S 上物理 CPU 核总数（不包括预分配给 Domain 0 的 CPU 数量）。初始状态下，每个 V_i 预分配的 vCPU 数量为 u_i，且随机、平均地在不同物理 CPU 上排队等待调度。根据 Xen 默认的计算资源分配原则，VM 占用 CPU 资源的分配比例权重 W_i 计算如下

$$W_i = \frac{u_i}{\sum_{i=1}^{N} u_i} \tag{1}$$

令 C_i 表示每个 VM 初始化分得物理 CPU 数量，即每个 VM 理论上可平均占用的 CPU 数量，则有

$$M = \sum_{i=1}^{N} C_i \tag{2}$$

$$C_i = M \frac{W_i}{\sum_{i=1}^{N} W_i} \tag{3}$$

令采样周期为 T_{m}，识别周期为 T_{s}，则在一个识别周期内采样次数 N_{s} 计算如下

$$N_{\mathrm{s}} = \frac{T_{\mathrm{s}}}{T_{\mathrm{m}}} \tag{4}$$

4.2.1 数据分组接收方向识别

在数据分组接收方向，Domain 0 接收从网卡发送的数据并通过 I/O 机制转发给 VM，若 VM 由于 vCPU 调度排队而导致不能及时接收数据分组，会使其持续处于溢出和过载状态。本文定义 rq_i^j 为第 j 次采样时 VM 请求项队列长度，rr_i^j 为响应项队列长度，rqn_i^j 为待放入请求项个数；ov_i^j 和 om_i^j 分别表征采样周期内 VM 是否处于溢出和过载状态，则有

$$ov_i^j = \begin{cases} 1, & rq_i^j + rr_i^j + rqn_i^j \geqslant QL_{\max} \\ 0, & rq_i^j + rr_i^j + rqn_i^j < QL_{\max} \end{cases} \tag{5}$$

$$om_i^j = \begin{cases} 1, & QL_{\mathrm{th}} \leqslant rq_i^j \leqslant QL_{\max} \text{且} ov_i^j = 0 \\ 0, & 0 \leqslant rq_i^j < QL_{\mathrm{th}} \end{cases} \tag{6}$$

在一个识别周期内对应 I/O 环溢出频率 f_{ro}^i 和 I/O 环请求项队列过载频率 f_{rq}^i 分别计算为

$$f_{\mathrm{ro}}^i = \frac{\sum_{j=1}^{N_{\mathrm{s}}} ov_i^j}{N_{\mathrm{s}}} \tag{7}$$

$$f_{\mathrm{rq}}^i = \frac{\sum_{j=1}^{N_{\mathrm{s}}} om_i^j}{N_{\mathrm{s}}} \tag{8}$$

在识别周期内，若 I/O 环请求项队列溢出频率 f_{ro}^i 大于给定阈值 δ_{th} 或过载频率 f_{rq}^i 大于判别阈值 θ_{th} 时，该 VM 为 NSVM 型，反之为 NNVM 型。

4.2.2 数据分组发送方向识别

在数据分组发送方向，VM 通过 Domain 0 发送数据分组，若 VM 由于 vCPU 调度排队而导致不能及时发送数据，会使 VM 频繁处于通信停滞状态。定义 sq_i^j 为发送方向上第 j 次采样时请求项队列长度，oe_i^j 表征采样周期内请求队列项长度是否为 0，ne 为识别周期内队列长度不为零的次数，当监测到队列长度不为零时，ne 自动累加。q_{se}^i 为 I/O 请求项队列平均长度，f_{se}^i 为发送方向 I/O 请求停滞频率，即

$$q_{\mathrm{se}}^i = \frac{\sum_{j=1}^{N_{\mathrm{s}}} sq_i^j}{ne} \tag{9}$$

$$f_{\mathrm{se}}^i = \frac{N_{\mathrm{s}} - ne}{N_{\mathrm{s}}} \tag{10}$$

在识别周期内，若请求项队列平均长度 q_{se}^i 大于给定阈值 QL_{eth} 且停滞频率 f_{se}^i 大于判别阈值 γ_{th} 时，该 VM 为 NSVM 型，反之为 NNVM 型。

4.2.3 接收和发送方向联合识别

由于 VM 在执行通信过程中数据分组的收发是同时进行的，因此，只要接收和发送的某一个方向判定为 NSVM 型，即可以判定该 VM 为 NSVM 型，反之为 NNVM 型。完成识别后，2 种类型的 VM 分别记入集合 V_{ns} 和 V_{nn}，整体识别流程如图 9 所示。

5 Diff-Scheduler 机制

基于 VM 类型识别结果，本节提出一种面向网络性能优化的新型虚拟计算资源分配及调度机制 Diff-Scheduler，该机制基于 2 种类型 VM 的资源需求，将物理计算资源进行重新分配，进一步提高 NSVM 的默认调度频率，最终提升其网络性能，同时保证了计算资源分配的公平性。

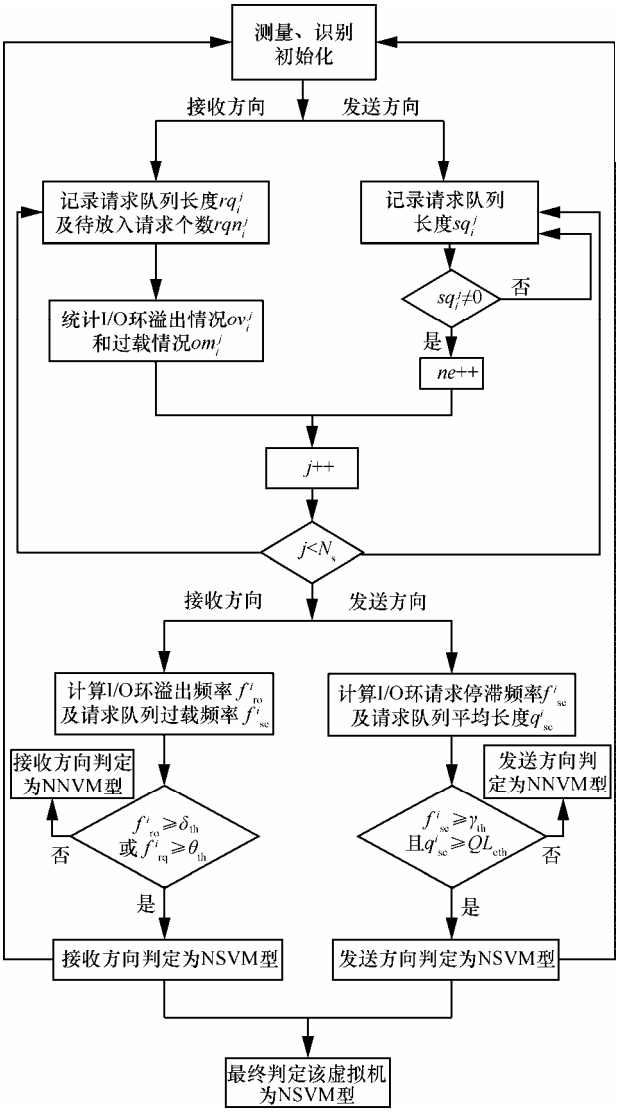


图 9　VM 类型联合识别流程

5.1　VM 计算资源使用情况分析

利用 Xen 提供的接口实时测量各个 VM 实际所占用的 CPU 负载，通常以 VM 所占单个物理 CPU 的倍数来表示，记作 L_i，i 为 VM 编号（i=1,…,N）。每个 VM 分别比较 L_i 和 C_i 的大小关系，判断当前计算资源是否满足其需求。当 $L_i \leqslant C_i$ 时，说明该 VM 没有消耗完分配给它的 CPU 计算资源，处于“计算资源非饥饿”状态；反之，若 $L_i > C_i$ 时，则说明当前时刻该 VM 处于“计算资源饥饿”状态。

经过上述分析，可将对应 NSVM 类型的 VM 分为 2 种子类型，即 NSVM 中的“计算资源饥饿”子型 NSVM_h 和“计算资源非饥饿”子型 NSVM_nh，其个数分别为 N_{nsh} 和 N_{nsnh}；相应地，NNVM 类型的 VM 也可进一步划分为 2 种子类型，即 NNVM 中的“计算资源饥饿”子型 NNVM_h 和 NNVM_nh“计算资源非饥饿”子型 NNVM_nh，其个数分别为 N_{nnh} 和 N_{nnnh}，显然有

$$\begin{cases} N_{\mathrm{ns}} = N_{\mathrm{nsh}} + N_{\mathrm{nsnh}} \\ N_{\mathrm{nn}} = N_{\mathrm{nnh}} + N_{\mathrm{nnnh}} \\ N = N_{\mathrm{ns}} + N_{\mathrm{nn}} \end{cases} \tag{11}$$

5.2　CPU 资源计算与重新分配

本节分别计算“计算资源非饥饿”型与“计算资源饥饿”2 种类型 VM 所需的 CPU 资源数量。首先，利用第 5.1 节中的统计分析结果来计算“计算资源非饥饿”状态 VM 所占的 CPU 数量 C_{nh} 以及 NSVM_nh 子型和 NNVM_nh 子型分别所占的 CPU 数量 C_{nsnh} 和 C_{nnnh}

$$C_{\mathrm{nsnh}} = \sum_{k=1}^{N_{\mathrm{nsnh}}} L_k \tag{12}$$

$$C_{\mathrm{nnnh}} = \sum_{r=1}^{N_{\mathrm{nnnh}}} L_r \tag{13}$$

$$C_{\mathrm{nh}} = C_{\mathrm{nsnh}} + C_{\mathrm{nnnh}} \tag{14}$$

其中，L_k 为每个 NSVM_nh 型 VM 对应占用的实际物理 CPU 数量，L_r 为每个 NNVM_nh 型 VM 对应占用的实际物理 CPU 数量。则余下物理 CPU 数量即为“计算资源饥饿”型 VM 应分得的 CPU 数量 C_h

$$C_h = M - C_{\mathrm{nh}} \tag{15}$$

按照 VM 初始化分得的 Weight 值 W_h^j 计算各个“计算资源饥饿”型 VM 应分得的物理 CPU 值 C_h^j，即

$$C_h^j = C_h \frac{w_h^j}{\sum_{j=1}^{N_h} w_h^j} \tag{16}$$

其中，N_h=N_{nsh}+N_{nnh} 为当前统计周期内所有“计算资源饥饿”型 VM 数量。最后计算出 NSVM_h 和 NSVM_nh 型 VM 应分得的物理 CPU 数量 N_{nsh} 和 N_{nnh}

$$C_{\mathrm{nsh}} = \sum_{p=1}^{N_{\mathrm{nsh}}} C_h^p \tag{17}$$

$$C_{\mathrm{nnh}} = \sum_{q=1}^{N_{\mathrm{nnh}}} C_h^q \tag{18}$$

其中，C_h^p 和 C_h^q 分别为 C_h^j 中 NSVM_h 和 NSVM_nh 型 VM 所应分配的物理 CPU 数量。

5.3 计算资源分池优化调度

利用 Xen 提供的工具接口，将物理宿主机服务器上的所有 CPU 资源划分为 2 个分池，分别对应 NSVM 资源分池和 NNVM 资源分池。初始状态下，NSVM 资源分池中的物理 CPU 数置为零，全部 CPU 资源默认在 NNVM 资源池中。

利用第 5.2 中的计算结果，向 NSVM 资源分池中分配 C_{ns} 个物理 CPU，并将 NSVM 对应的 vCPU 分配到该资源分池进行调度，其中，$C_{\mathrm{ns}}=\lvert C_{\mathrm{nsh}}+C_{\mathrm{nsnh}}\rvert$ 为所有 NSVM 需要的 CPU 值，并向下取整。同时，向 NNVM 分池中分配 C_{nn} 个物理 CPU，将 NNVM 对应的 vCPU 分配到该池进行调度，其中 $C_{\mathrm{nn}}=\lvert C_{\mathrm{nnh}}+C_{\mathrm{nnnh}}\rvert$ 为所有 NNVM 需要的 CPU 值，并向下取整。计算资源分池调度示例如图 10 所示。

NSVM 资源分池和 NNVM 资源分池中的调度算法仍然采用 Xen 默认的 Credit Scheduler，但 2 个分池的 CPU 时间片调度周期（slice time）需要进行重新设定。其中，NSVM 资源分池调整为 5 ms（本文默认调整值），NNVM 资源分池仍然保持默认值 30 ms。实际情况下，NSVM 资源分池的 CPU 时间片调度周期可以根据用户应用的密集程度动态调整。

需要补充说明的是，由于实际情况下计算出来的 C_{ns} 和 C_{nn} 很可能不是整数，其小数部分之和约为 1。为了有效利用资源，本文采用随机概率的方式进行余下 1 个 CPU 的分配。

1) 假设 C_{ns} 取整之前的小数部分为 δ（保留一位小数），C_{nn} 取整之前的小数部分为 η，分别计算此时属于 C_{ns} 和 C_{nn} 的余数分配百分比 λ_{ns} 和 λ_{nn}，即

$$\lambda_{\mathrm{ns}}=\frac{10\delta}{10\delta+10\eta}\times 100\% \tag{19}$$

$$\lambda_{\mathrm{nn}}=\frac{10\eta}{10\delta+10\eta}\times 100\% \tag{20}$$

2) 设定一个随机数程序，其中，随机数数目为 $10\delta+10\eta$，每次进行一次随机数的生成，以概率 λ_{ns} 将余下的 1 个物理 CPU 分配给 NSVM 资源分池，以概率 λ_{nn} 将余下的 1 个物理 CPU 分配给 NNVM 资源分池。

6 原型系统设计

Xen 的 I/O 通信系统由位于 Domain 0 域的后端驱动、位于 VM 侧的前端驱动以及共享内存机制共同协作完成。为了更好地实现 VM 类型识别和区分式的分池调度策略，本文在原有 Xen 平台的基础上扩展实现了部分功能模块，设计实现了基于 I/O 传输特征感知的虚拟计算资源分配与优化调度原型系统，架构如图 11 所示。为了不影响原有功能的运行，系统设计遵循如下原则：1) 除信息监测模块以外，所有模块均部署和工作于 Domain 0 特权域，各个虚拟机系统不需要做任何改变，增加了系统的可移植性和部署的灵活性；2) 各个子模块功能均为轻量级，信息统计分析及策略生成操作由 Python 语言编写，运行成本开销较小，不会给 Domain 0 域增加额外大的负担。

6.1 I/O 队列信息监测模块

该模块工作于 Xen 内核空间中，通过模块插入和退出的方式可进行灵活部署，可以周期性地实时监测后端驱动的 I/O 传输队列信息和虚拟机

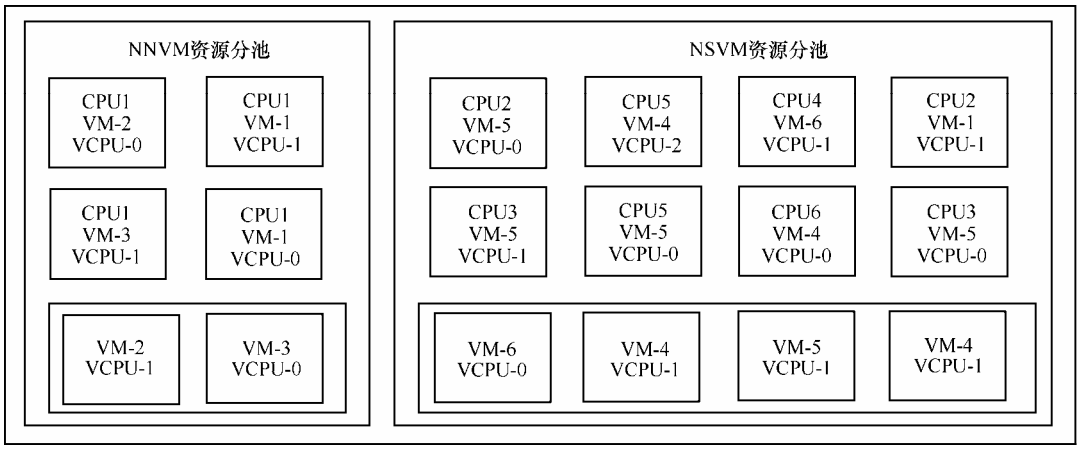


图 10 计算资源分池调度示例

当前 CPU 负载情况，包括请求项队列长度、响应项队列长度及实际占用物理 CPU 数量等。信息统计默认周期为 10 毫秒/次。获取信息有 2 个途径：一是通过 I/O 传输共享数据内存缓存区提供的接口，获取 Xen 的前后端分离驱动的 netback 和 netfront 之间共享环形缓冲区存放的信息；二是通过 Xen 中 Libxenstat 库提供的接口来获取每个虚拟机当前的 CPU 使用信息，随后将信息传递给信息统计分析模块。

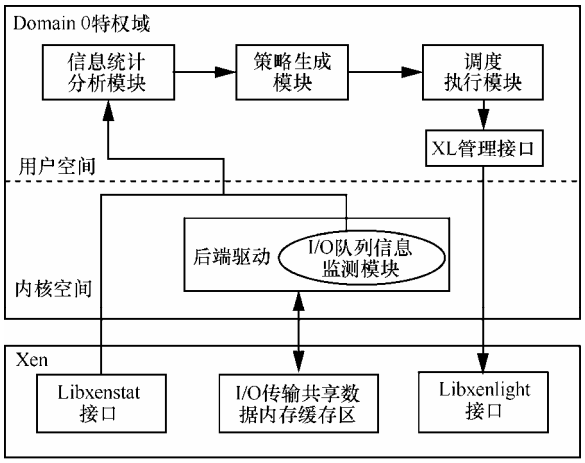


图 11　原型系统架构设计

6.2　信息统计分析模块

接收来自 I/O 队列信息监测模块中的信息，分别提取每个虚拟机对应的发送、接收 I/O 共享环中的各类队列长度等参数信息，并做线性平滑处理，将处理结果发送至策略生成模块。

6.3　策略生成模块

根据信息统计分析模块传递的处理结果，进行虚拟机类型识别操作。本文所提的相关 VM 类型识别和计算资源分配核心策略部署于该模块中。主要过程是将当前虚拟机分为 NSVM 型和 NNVM 型，同时分别计算 2 种类型的 VM 应分配的 CPU 数量，将上述内容汇总为 CPU 分池调度策略，并将策略信息传递给调度执行模块。

6.4　调度执行模块

基于 Xen 提供的 XL 管理接口，周期性执行 CPU 分池比例的调整和调度频率调整。

7　实验与结果分析

为了验证本文所提面向网络性能优化的虚拟计算资源分配及调度机制的准确性和有效性，本文与 Xen 默认的 Credit Scheduler 调度机制进行了性能对比实验，比较 VM 网络业务吞吐量、延迟变化情况，此外还包括应用 Diff-Scheduler 机制对 CPU 资源分配公平性影响的验证。

7.1　实验环境

实验环境采用与 3.3 节相同的拓扑与配置，所不同的是本文利用 Linux 限流工具添加 30ms 延迟以及 160 Mbit/s 的最大网络吞吐量限制，用以模拟真实的小规模数据中心网络环境。

虚拟机类型识别参数设置如下：接收方向上，I/O 环请求队列溢出频率阈值 δ_{th} 设定为 0.2，I/O 环请求队列项长度过载阈值为 150，过载频率判别阈值 θ_{th} 为 0.6；发送方向上，I/O 环请求队列长度阈值 QL_{eth} 为 45，停滞频率判别阈值 γ_{th} 为 0.5。

7.2　结果分析

为了更好地验证本文机制的有效性，实验在下列 2 种场景下对 TCP、UDP 吞吐量、平均延迟以及 CPU 资源公平性等性能指标进行比较。

1) vCPU 平均队列长度为 1 的场景，即 Queue-1 场景（参考第 3.3 节），此时为每个 VM 预分配 2 个 vCPU。

2) vCPU 平均队列长度为 2 的场景，简称 Queue-2 场景，此时为每个 VM 预分配 3 个 vCPU。

7.2.1　TCP 吞吐量

如图 12 所示，在 Queue-1 场景下，使用 Xen 默认的 vCPU 调度机制 Credit Scheduler 时，TCP 业务流平均吞吐量仅为 54 Mbit/s。当采用 Diff-Scheduler 机制后，TCP 业务流平均吞吐量达到了 96 Mbit/s，提升约 77.8%；在 Queue-2 场景下，采用 Credit Scheduler 机制，TCP 吞吐量持续降低，仅为 37 Mbit/s，因为 vCPU 排队的时延增加，更多的虚拟机参与 CPU 资源竞争，使性能受到的影响加大。采用 Diff-Scheduler 后，吞吐量达到 74 Mbit/s，性能提升 100%。

7.2.2　UDP 吞吐量

如图 13 所示，在 Queue-1 场景下，分别采用 Credit Scheduler 和 Diff-Scheduler 后，UDP 平均吞吐量分别为 66 Mbit/s 和 118 Mbit/s，提升将近 78.8%；相应地，在 Queue-2 场景下，UDP 吞吐量分别为 45 Mbit/s 和 95 Mbit/s，提升 111.1%。相比 TCP，Diff-Scheduler 对 UDP 业务网络性能提升更明显。

7.2.3　平均延迟

本文利用 ICMP RTT 来测量待测 VM 和测试对端 T 之间的平均延迟。利用 PING 工具产生 ICMP

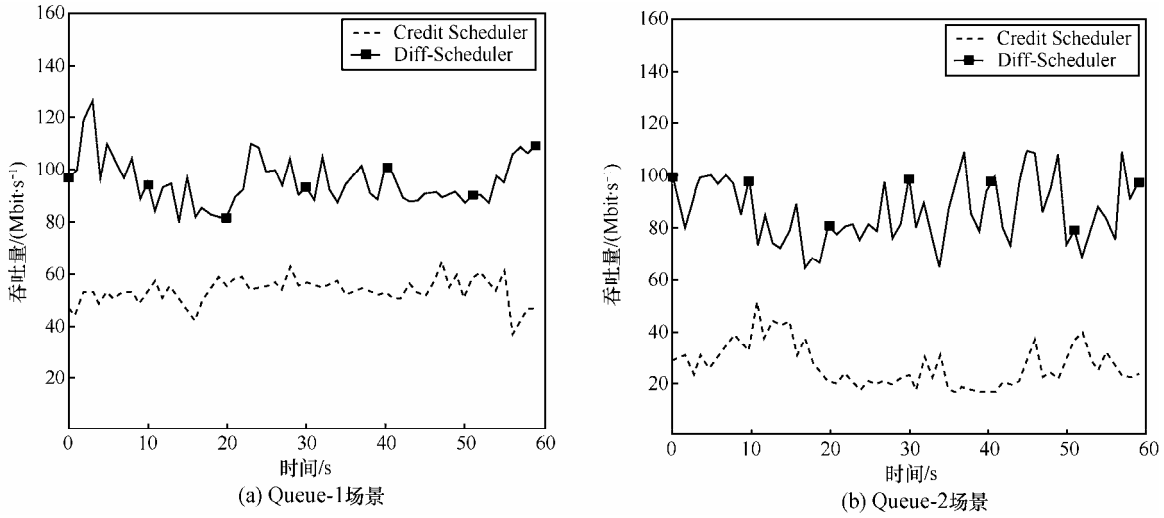


图 12　TCP 吞吐量变化情况对比

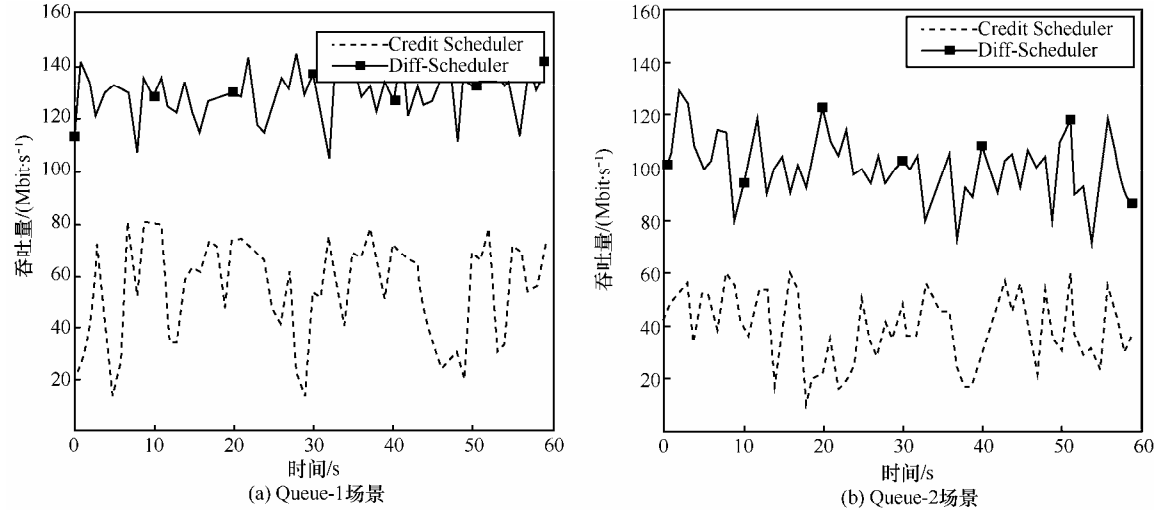


图 13　UDP 吞吐量变化情况对比

请求及响应，发送速率为 10 次/秒，载荷为 100 byte。图 14 分别对应 Queue-1 和 Queue-2 这 2 种场景下的平均延迟概率累计分布函数图（CDF 图）。在 Queue-1 场景下，相比 Credit Scheduler，本文所提 Diff-Scheduler 机制将平均延迟从 71 ms 减少至 32 ms，下降近 55%；在 Queue-2 场景下，平均延迟由 96 ms 减少至 37 ms，下降近 61%；上述结果表明，应用本文所提机制能够有效、持续地降低网络平均延迟。

7.2.4　计算资源分配公平性

通过原型系统实验已验证 Diff-Scheduler 机制能够有效提升虚拟机的网络性能，本节验证该机制计算资源分配的公平性。众所周知，Credit Scheduler 机制进行计算资源分配具有良好的公平性。因此，本文以 Credit Scheduler 机制作为基准标准，利用 2 种机制下，NSVM 和 NNVM 这 2 种类型虚拟机所分得的 CPU 数量比值来评价公平性。

令 C_{NS} 和 C_{NN} 作为 Credit Scheduler 机制下，NSVM 和 NNVM 这 2 种类型虚拟机所分得的 CPU 数量；C_{ns} 和 C_{nn} 作为 Diff-Scheduler 机制下，NSVM 和 NNVM 这 2 种类型虚拟机所分得的 CPU 数量，FR_{ns} 和 FR_{nn} 分别为对应 2 种类型虚拟机的资源公平性比率

$$FR_{\mathrm{ns}}=\frac{C_{\mathrm{ns}}}{C_{\mathrm{NS}}}$$
$$FR_{\mathrm{nn}}=\frac{C_{\mathrm{nn}}}{C_{\mathrm{NN}}} \tag{21}$$

式(21)中的比率值越接近 1，表明 Diff-Scheduler

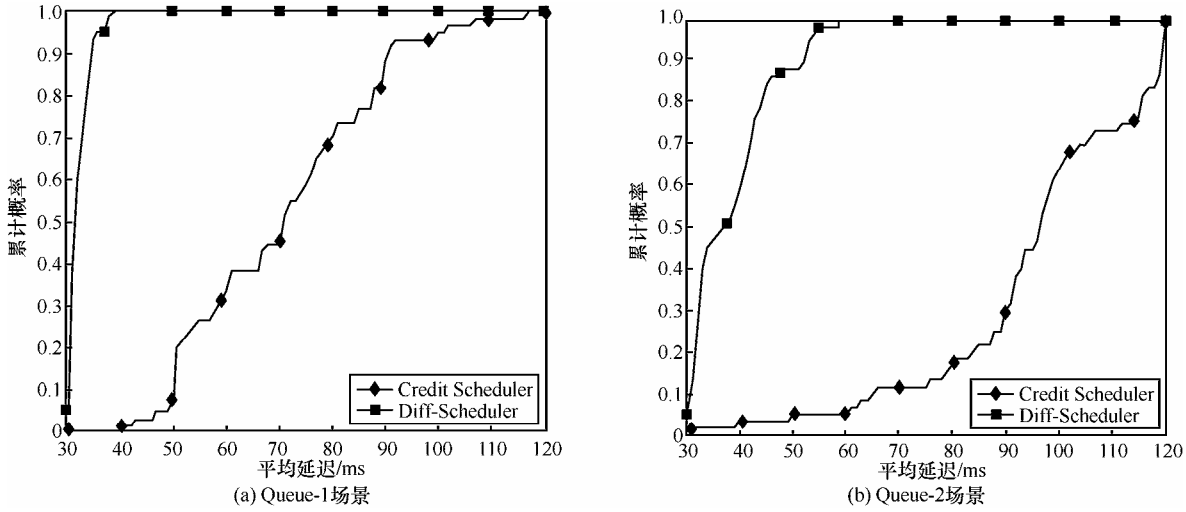


图 14　网络平均延迟变化情况对比

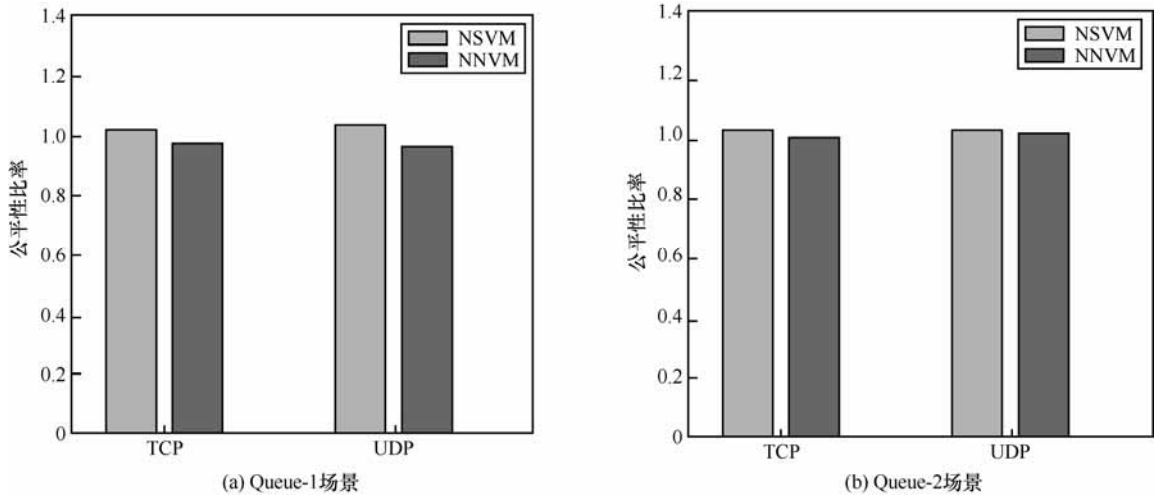


图 15　计算资源分配公平性验证结果

具有更准确的资源分配公平性。实验结果如图 15 所示，在 Queue-1 场景下，对应 TCP 和 UDP 业务，NSVM 类型的计算资源公平性比率分别为 1.023 和 1.034；相应地，在 Queue-2 场景下，对应 TCP 和 UDP 业务流，NSVM 类型的计算资源公平性比率分别为 1.015 和 1.008。对 NNVM 类型而言，其计算资源公平性比率在 2 种情况下同样非常接近于 1。说明 Diff-Scheduler 机制充分保证了 CPU 资源分配的公平性。此外，相比 NSVM 类型，NNVM 类型的公平性比率值略低于 1，这是因为计算资源重新分配与分池调度时，分配剩余 CPU 时产生了细微的随机误差，也是本文所提机制在提升网络性能与资源分配之间的细微权衡。在实际系统运行中，这种细微权衡将不会对计算和网络通信任务带来影响。

8　结束语

云计算为用户提供了一种基于互联网的新型计算模式。作为搭建云计算平台的基础，虚拟化技术通过将计算机的各种实体资源抽象以便实现资源共享。然而，上述计算资源共享的方式也给虚拟机网络性能带来了较大的负面影响。本文以主流开源虚拟化技术 Xen 为研究对象，针对其默认的 vCPU 调度策略造成虚拟机网络性能下降的问题进行深入研究。提出了一种新型的面向网络性能优化的计算资源分配及调度机制。通过实验表明，该机制可有效提高虚拟机网络的吞吐量，显著降低延迟，同时保证计算资源分配的公平性。

未来的研究工作将基于大规模数据中心内部资源池环境进行。包括：1) 数据中心内部多服务器

之间的 VM 资源调度，可根据当前 VM 的 CPU 负载竞争情况与通信情况将计算资源进行重新调配，将通信流量集中的 NSVM 型虚拟机迁移至低负载宿主机服务器；2) 基于应用类型的优化调度方式，数据中心内部有大量不同类型的任务，下一步工作可根据不同类型任务的 QoS 要求进行有针对性的资源调度和隔离，为用户提供高效的区分式服务。

参考文献：

[1] Amazon elastic compute cloud[EB/OL]. http://aws.amazon.com/ec2/.

[2] Google cloud platform[EB/OL]. http://cloud.google.com.

[3] Microsoft server and cloud platform[EB/OL]. http://www.microsoft.com/servercloud/.

[4] Aliyun.com[EB/OL]. http://appcloud.aliyun.com.

[5] Wo cloud[EB/OL]. http://www.wocloud.cn.

[6] Ecloud[EB/OL]. http://ecloud.10086.cn/.

[7] E Cloud[EB/OL]. http://www.ctyun.cn/.

[8] The XEN project[EB/OL]. http://www.xenproject.org/.

[9] OSTERMANN S, IOSUP A, YIGITBASI N, et al. A performance analysis of EC2 cloud computing services for scientific computing[C]//International Conference on Cloud Computing. c2010:115-131.

[10] WHITEAKER J, SCHNEIDER F, TEIXEIRA R. Explaining packet delays under virtualization[J].ACM Sigcomm Computer Communication Review, 2011,41(1):38-44.

[11] WANG G, NG T. The impact of virtualization on network performance of amazon EC2 data center[C]//The 29th IEEE International Conference on Computer Communications. c2010:1-9.

[12] SHEA R, WANG F, WANG H, et al. A deep investigation into network performance in virtual machine based cloud environments[C]// The 33th IEEE International Conference on Computer Communications. c2014: 1285-1293.

[13] GAMAGE S, KANGARLOU A, KOMPELLA R R, et al. Opportunistic flooding to improve TCP transmit performance in virtualized clouds[C]//The 2nd ACM Symposium on Cloud Computing. c2011: 24.

[14] KANGARLOU A, GAMAGE S, KOMPELLA R R, et al. vSnoop: improving TCP throughput in virtualized environments via acknowledgement offload[C]//The 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. c2010: 1-11.

[15] XU C, GAMAGE S, RAO P N, et al. vSlicer: latency-aware virtual machine scheduling via differentiated-frequency CPU slicing[C]// The 21st International Symposium on High-Performance Parallel and Distributed Computing. c2012: 3-14.

[16] XU Y, MUSGRAVE Z, NOBLE B, et al. Bobtail: avoiding long tails in the cloud[C]//The USENIX Conference on Network Systems Design and Implementation. c2013: 329-341.

[17] XU Y. Characterizing and mitigating virtual machine interference in public clouds[D]. Michigan: University of Michigan, 2014.

[18] XU Y, BAILEY M, NOBLE B, et al. Small is better: avoiding latency traps in virtualized data centers[C]//The 4th Annual Symposium on Cloud Computing. c2013.

[19] Credit2 scheduler[EB/OL]. http://wiki.xen.org/wiki/Credit2_Scheduler_Development#Status.

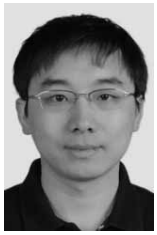
[20] Iperf[EB/OL]. http://iperf.sourceforge.net/.

[21] Sysbench[EB/OL]. http://sysbench.sourceforge.net/.

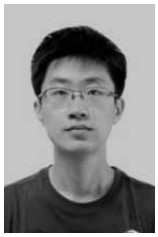
作者简介：

王煜炜（1980-），男，河北唐山人，中国科学院博士生、助理研究员，主要研究方向为未来网络、虚拟化技术和云计算。

刘敏（1976-），女，河南郑州人，博士，中国科学院研究员、博士生导师，主要研究方向为移动管理、网络测量和移动计算。



房秉毅（1980-），男，山东泰安人，博士，中国联合网络通信集团有限公司高级工程师，主要研究方向为下一代网络、移动核心网和云计算。



秦晨翀（1991-），男，山西临汾人，中国科学院硕士生，主要研究方向为虚拟化技术、下一代网络、移动计算。

闫小龙（1990-），男，安徽阜阳人，中国科学院硕士生，主要研究方向为虚拟化技术、下一代网络、移动计算。