Optimization of KVM Network Based on CPU Affinity on Multi-cores

Zhaoliang Guo, Qinfen Hao
School of computer science and engineering, Beihang University, Beijing, 100191, China
E-mail: guozhaoliang612@gmail.com, haoqf@buaa.edu.cn

***Abstract*—In order to improve the network performance of KVM virtual machines, an approach based on CPU affinity is proposed in this paper. By running the processes on the same CPU, the processes can run more efficiently by reducing performance degrading situations such as cache misses. In some performance-critical situations, it makes sense to bind the process to the same CPU. We run the benchmarks NETPERF and PING to validate our method. The results show that with the optimization of CPU affinity, the network latency of KVM decreased by 20% (taskset) and 10%(cgroup), while sustaining the same network bandwidth compared with four CPU cores. The CPU affinity method can effectively improve the network performance of KVM virtual machine by improving the cache performance and reducing the called times of some important models.**

Keywords- virtualization; KVM; network performance; CPU Affinity; cgroup; virtio

I. 引言

KVM(Kernel-based Virtual Machine)是一种基于 Linux 内核的虚拟化技术，能够利用处理器的硬件支持使 Linux 内核成为一个虚拟机监控器(Virtual Machine Monitor)[1]。KVM 具有良好的性能，但是在 I/O 比较密集型的应用情况下，其开销和性能与直接运行在本地机器上相比会有很大的差距[2]。特别是在一些对性能要求较高的应用中，比如实时的网络数据传输和处理中，KVM 还需要进行进一步的改进和优化。

另一方面，随着多核处理器[3]的流行，如何有效地利用多核环境来提高网络性能也存在着一些挑战[4,5,6,7,8,9,10]。在多核环境下，系统可以同时运行多个服务，比如一个核上运行 web 服务器，另一个核上运行数据库服务器。存在的一个问题就是如何调度的问题。如果把所有的服务都调度到一个核上，并不能有效地利用系统中的其他核。如果把不同的服务调度到不同的核上运行，各个核之间就会发生资源争用的情况。因为各个核之间会共享多种资源，比如网卡，以及共享的内存等资源。

本论文重点是在多核环境下，如何优化 KVM 的网络性能。首先对 KVM 的总体架构进行了介绍，然后分析了 KVM 的 I/O 虚拟化方法以及网络通讯机制，接着通过处理器亲和性的优化方法，实现了进程绑定，对 KVM 的网络性能进行了优化，优化结果显示，在小数据包情况下，在不减少吞吐量的情况下，可以将 KVM 的网络延迟性能降低 20%和 10%左右。最后利用 Linux 中的系统级的性能评测工具 OProfile[11]对其原因进行了进一步的分析。

II. 背景

A. *KVM 总体架构*

KVM 隶属于操作系统层面的虚拟化，为 ISA 层虚拟化技术所支持，不需要对客户操作系统进行修改，属于全虚拟化范畴。KVM 在运行时，需要 ISA 层虚拟化的支持，即需要 CPU 支持 Intel VT 技术或者 AMD SVM 技术。

KVM 在创建或运行虚拟机前，会先在内核空间创建 KVM Driver。KVM Driver 将整个 Linux 内核 VMM 化。KVM Driver 负责创建 vCPU，为虚拟机分配内存，I/O 相关的活动交给用户进程 QEMU 实现。在运行过程中，虚拟机对宿主机操作系统表现为一个普通的用户进程。VMM 对虚拟机的操作通过 Linux 内核所提供的各种 API 与系统调用实现，例如调度，异步事件处理等。KVM 设计思想的核心就在于以内核模块方式实现 CPU 与内存的虚拟化，通过用户进程实现 I/O 与设备的虚拟化，并且直接调用内核空间相关功能模块对虚拟机进行操作，使得 KVM 在付出较少代价的情况下，获得了完全的 VMM 功能。

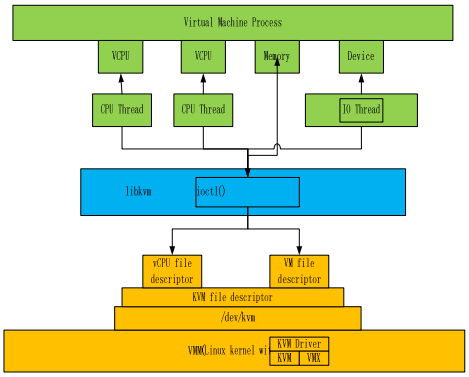

Figure 1 Architecture of KVM

图 1 KVM 架构图

B. *KVM 网络通讯机制*

KVM 是一种基于宿主机的虚拟化方法，VMM 负责拦截虚拟机的 I/O 操作，而运行在用户控件的 QEMU 负

责执行实际的 I/O 操作。KVM 支持两种不同的 I/O 虚拟化方法：设备仿真和半虚拟化。

设备仿真是指通过纯软件的方法来模拟常见的设备。它的主要优点是不需要修改客户设备驱动，缺点是效率低，模拟的难度大，而且在处理 I/O 指令时开销很大，需要经过多次上下文切换，包括虚拟机到 VMM 的切换，以及从内核到用户态的切换。KVM 采用 QEMU 模拟的方式实现了 I/O 设备的全虚拟化技术，但是性能较差，是整个系统的瓶颈所在。而半虚拟化技术是对全虚拟化技术的优化。

KVM 引入了半虚拟化的 I/O 虚拟方法 virtio[12]。Virtio 驱动以前后端的形式出现，前端驱动位于虚拟机里面，后端驱动位于运行在用户空间的 QEMU，前后端之间通过一个环形缓冲区进行通信，通过其环形缓冲区能够将数据直接从宿主机的内核空间拷贝至虚拟机的内核空间，因此可以减少数据拷贝和上下文切换开销，提高了 I/O 处理的性能。

具体的对于网络数据包的处理，以数据包的接受为例，传统的设备仿真方法和半虚拟化技术的区别如图 2 所示：

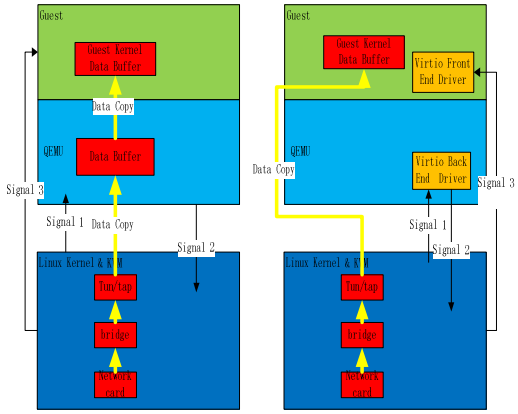

Figure 2 Comparison of network simulation models

图 2 KVM 网络传输路径对比图

在设备仿真模式下，数据包经过主机部分的网卡驱动，网桥，TUN/TAP 三个模块，然后数据包被拷贝至用户空间进程 QEMU 的数据缓冲区，QEMU 进程再将缓冲区中的数据拷贝至客户端的内核空间。而采用半虚拟化的 virtio 时，QEMU 进程能够直接将数据包拷贝至虚拟机的内存空间，从而减少了一次数据拷贝，并且减少了用户模式到内核模式以及客户模式到内核模式的切换次数。因此，virtio 相对于设备仿真来说具有更好的性能

C. KVM 网络 I/O 模拟存在的问题

由前面的分析可以看到，KVM 网络的 I/O 路径主要包括三部分，一部分是宿主机的部分网络协议层，以及联系宿主机和虚拟机的网络驱动层，还有虚拟机内的客户操作系统的网络协议层。虚拟机上的数据包的接受路径比起物理机上的接受路径要复杂的多，对于某些时间敏感的实时应用来说，过长的网络 I/O 路径会造成较大的网络传输时延，从而影响整个系统的性能。我们的主要目的是在不影响网络传输带宽的情况下，尽可能的提高 KVM 虚拟机在小数据包情况下的网络传输时延

III. 基于处理器亲和性的进程绑定实现

为了提高 KVM 虚拟机在小数据传输情况下的延迟，我们采用了基于处理器亲和性的方法，将 KVM 进程绑定到某个核上运行，从而降低其传输时延。

A. 处理器亲和性原理

在网络协议处理过程中，调度程序有可能会把 KVM 进程迁移至不同的处理器上进行处理，但是这些操作对于系统缓存效率有间接的影响。现在典型的支持 SMP 或者多核的操作系统的调度程序在调度是会尽量的考虑到负载平衡，会把那些跑在高负载处理器上的进程调度到那些负载较轻的处理器上。任何由于调度引起的进程迁移情况的发生，都会使系统付出一定的性能代价。因为被迁移的进程在迁移到新的处理器上时，需要逐渐的把相应的各级缓存数据也迁移过来。会导致在一段时间内程序访问指令和数据的局部性减弱，从而降低了 Cache 的命中率。所以如果能够使单个程序始终在同一个 CPU 上运行，这样的话会使 Cache 得命中率得到提高，从而提高程序的响应时间，提高系统性能。

CPU 亲和性（affinity）[13][14]就是进程要在某个给定的 CPU 上尽量长时间地运行而不被迁移到其他处理器的倾向性。亲和性分为软亲和性和硬亲和性。软亲和性是指调度程序会尽可能的把进程调度到同一个处理器上运行。这意味着进程通常不会在处理器之间频繁迁移，从而减少负载。这仅仅是尽最大努力，当不能将其调度到同一个处理器时，调度程序会将其迁移至其他处理器上。Linux 内核进程调度器具有 CPU 软亲和性（affinity）的特性。硬亲和性是指我们可以显示的指定进程可以运行在哪个处理器上，并且一旦指定之后，进程就只能在指定的处理器上运行。

B. 实现方法

我们主要通过两种方法来实现绑定，也就是硬亲和性的方法。一种方法是用 Linux 工具集里面的 taskset 工具将进程绑定在某个处理器，另一种方法是利用 Linux 中的控制组提供的接口来实现进程绑定。第一种绑定是通过 taskset 方法来实现。Taskset 可以用来设置进程的亲和性，通过系统调用提供的接口，可以修改进程的处理器亲和性掩码，从而可以将进程绑定到指定的处理器上运行。第二种方法是通过控制组的方法。控制组是（cgroup）[15]是 Linux 内核提供的一种基于进程划分集合的机制，能够将系统中的进程集合，以及他们相应的子进程，聚合或划分成层次结构的树状结构，就像 Linux 的文件系统结构一样。相应的树中各个节点对应的是系统中的进程集合。一个 cgroup 组能够将一组进程

与系统中的一个或多个分系统资源通过一定的参数联系起来的，通过参数设置从而能够控制分配给进程组的系统资源。比如处理器资源，内存资源，网络带宽资源等。一个分系统相当于一个模块，它通过控制组（cgroup）提供的进程分组功能，对各组进行一定的操作和设置。典型的分系统就是资源控制系统，可以利用此分系统调度分配给进程组的各种系统资源，或者设置进程组所能消耗的资源的阈值。如下图所示，可以利用控制组系统合理的把系统中的处理器资源分配给各个进程组，通过设置合理参数值，控制组系统会根据相对的参数值的比例来分配处理器资源，从而能够做到细粒度的资源分配。

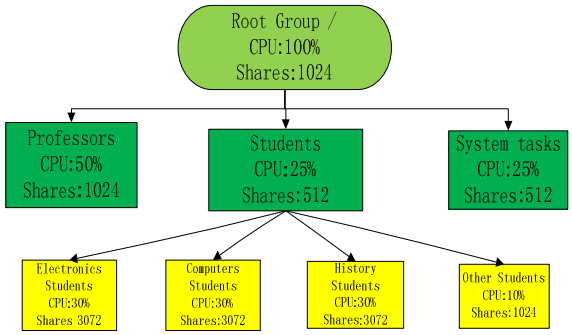

Figure 3 Cgroup Models

图 3 控制组示意图

当然分系统也可以是其他任何可以基于进程组进行管理和操作的子系统。目前 Linux 内核中控制组（cgroup）子系统支持的子资源管理系统主要有用于统计处理器利用率的 CPUACCT 子系统，以及用于绑定处理器和内存的 CPUSETS 子系统，以及用于启停控制组中各个进程的 FREEZER 子系统等。CPUSETS 能够将计算机中的处理器资源和内存资源分配给指定的进程集合。CPUSETS 使当前进程只能在预先分配好的处理器和内存集合中运行，能够限制这些进程消耗的处理器和内存资源。CPUSETS 通过系统调用来设置进程的处理器亲和性掩码，使进程只能运行在这些设置了掩码的处理器上。通过 CPUSETS 子系统可以实现进程绑定，将 KVM 进程绑定至某个核上运行。

IV. 实验以及数据分析

本节主要针对 KVM 在 virtio 模式下的网络性能进行测试。首先测试优化前的网络性能，然后测试优化后的网络性能。每项测试都通过 NETPERF[16]和 ping 程序来测量网络带宽和时延。在每次测试的时候都启动 OProfile 来测量在网络处理过程中各个模块对应的开销，然后通过对这些数据的对比，来分析绑定优化能使延迟减少的原因，从而为进一步的对 KVM 的网络性能优化指出可以参考的意见。

A. 实验软硬件配置信息

本实验使用的测试平台由两台物理服务器组成，两台物理服务器通过千兆的交换机连接。服务器中的处理器是 Intel®Xeon®5160 型号的处理器，是双核双处理器系统。其结构图如下：

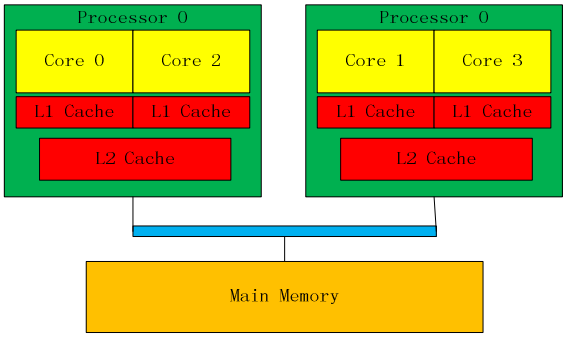

Figure 4 CPU Architecture

图 4 处理器结构图

硬件参数信息如下表：

Table 1 SYSTEM INFORMATION

表 1 系统软硬件信息

System Under Test(SUT)	
处理器型号	Intel®Xeon®5160 3.0GHZ 双核
缓存	4MB 二级缓存
前端总线频率	1333MHZ
内存	DDR2 667MHZ 4G
网卡	Intel 80003ES2 千兆网卡
宿主机内核版本	Linux-2.6.32
虚拟机内核版本	Linux-2.6.28
QEMU 版本	Qemu-kvm-0.14.0

B. 实验数据

测试时一台服务器作为客户端用于产生网络流量，另一台安装有虚拟机的服务器作为接收端，接受来自客户端的数据包。

实验一：测试 KVM 在设备仿真模式下，没有绑定和绑定时的带宽。通过 NETPERF 测带宽。实验结果如图所示：

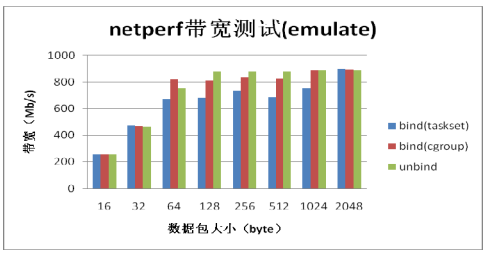

Figure 5 Bandwidth test with emulate model

图 5 模拟模式下带宽测试数据

实验二：测试 KVM 在半虚拟化 virtio 模式下，没有绑定和绑定时的带宽。通过 NETPERF 测带宽。实验结果如图所示：

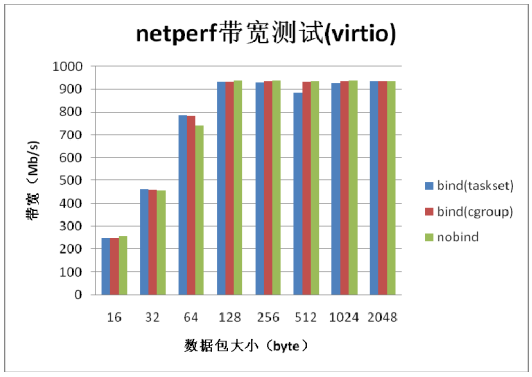

Figure 6 Bandwidth test with virtio model

图 6 半虚拟模式下带宽测试数据

实验三：测试 KVM 在半虚拟化 virtio 模式下，没有绑定和绑定时的延迟。通过 ping 程序来测试时延。实验结果如图所示：

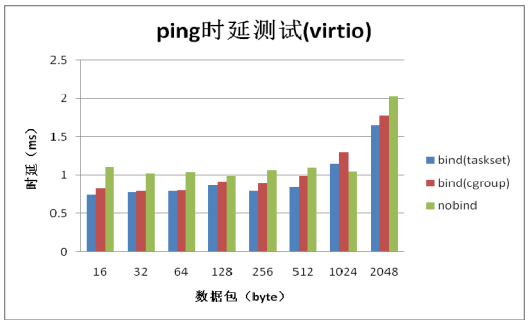

Figure 7 Latency test with virtio model

图 7 半虚拟化模式下时延测试数据

实验四：测试在 virtio 模式下，没有绑定情况下，在时延测试过程中 KVM 网络 IO 的主要开销。先启动性能测试工具 OProfile,然后再用 ping 程序测试时延。实验结果如图所示：

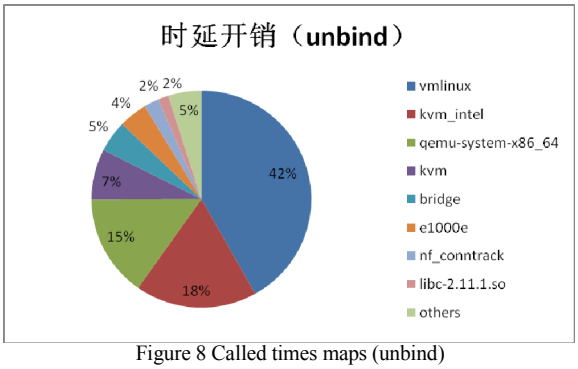

Figure 8 Called times maps (unbind)

图 8 模块调用次数图（不绑定）

实验五：测试在 virtio 模式下，在绑定情况下，在时延测试过程中 KVM 网络 IO 的主要开销。先启动性能测试工具 OProfile,然后再用 ping 程序测试时延。实验结果如图所示：

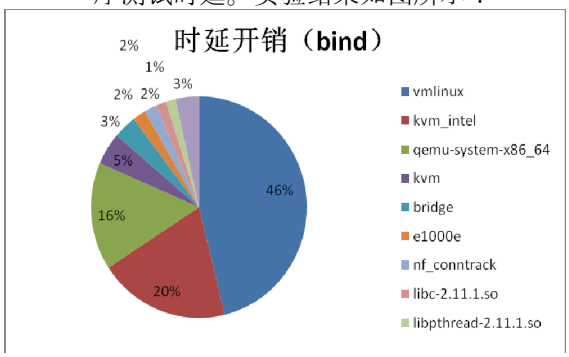

Figure 9 Called times maps (bind)

图 9 模块调用次数图（绑定）

C. *实验数据分析*

从实验一的结果可以看出，在设备模拟情况下，绑定模式并不能提高系统的带宽，相反的由于在绑定情况下一个核的处理能力并不能处理所有的数据包，导致带宽有所下降。对比实验二可以看出，在设备模拟情况下，相对于 virtio 模式，需要消耗更多的处理器资源，此时处理器的处理能力成为影响性能的主要瓶颈。所有亲和性带来的性能优势都被处理器的高负载所掩盖。所以在设备模拟情况下，受处理器处理能力的影响，绑定优化不仅不能提高带宽，反而使带宽有所下降。

从实验二的结果可以看出，在半虚拟化模式下，没有绑定和绑定模式下的带宽基本持平。和实验一的结果对比，可以看出，半虚拟化模式通过减少数据的拷贝次数，对网络 I/O 路径进行优化后，所消耗的处理器资源有所减少，所以在绑定的情况下，一个核的处理能力已经能够满足网络数据的处理，结果是两种模式下的带宽基本持平。而且在某些情况下，例如在数据包为 64 比特的情况下，绑定时的带宽高出 6%左右。从这个实验可以看出在半虚拟化模式下，一个核已经能够满足网络处理的能力，并不会对网络带宽有所影响。

从实验三的结果可以看出，在半虚拟化模式下，通过将 KVM 进程绑定至一个核上，可以减少网络延迟。通过 taskset 绑定平均的网络延迟可以减少 20%左右，最大时可以使网络延迟减少 32%。而通过控制组绑定的话，平均的网络延迟可以减少 10%左右，最大时可以使网络时延减少 25%。只有在数据包为 1024 比特时，时延并没有减少，反而增加了。

通过这三个实验，我们可以看出，半虚拟化模式通过对网络路径的优化，相对于设备模拟的方式，能够减少处理器资源的消耗，而且能够提高系统的带宽。而在半虚拟化模式下，通过不同的绑的方法，能够在保持带宽基本持平的情况下，能够使网络延迟分别减少 20%(taskset)和 10%（cgroup）左右。

为了找出在半虚拟化模式下，通过绑定的方法为何能够减少延迟，我们通过实验四和实验五的数据对比来进行分析。通过实验四和实验五的结果，我们可以看出，绑定可以减少以下几个模块的调用次数：kvm, bridge ,e1000e。为了使数据更具有直观性，通过下图的数据对比：

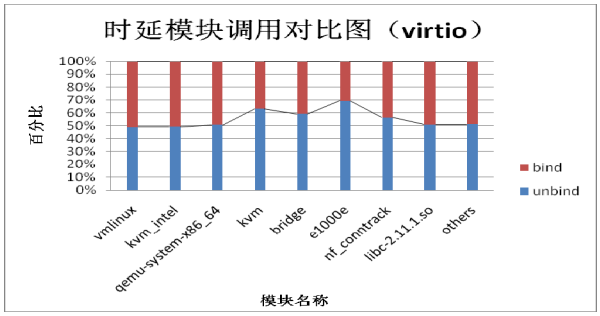

Figure 10 Comparison between bind and unbind

图 10 模块调用次数对比图

从这两个对比图中，我们可以看到，在半虚拟化时，绑定模式和没有绑定的模式相比，kvm，bridge，e1000e,nf_conntrack 四个模块的平均调用次数明显减少，而其他几个模块调用次数基本持平。通过图示，可以看出，e1000e 模块调用次数能够减少 55%左右，而 kvm 模块调用次数能够减少 40%左右，bridge 和 nf_conntrack 模块调用次数能够减少 30%和 20%左右。而这几个模块 e1000e,bridge,kvm 分别涉及到数据包的读取操作，包括把数据包从网卡空间拷贝至内核空间，然后通过网桥模块拷贝至 virtio 驱动的共享队列中，然后再拷贝至虚拟机内应用程序空间。我们可以看到，通过绑定的方法，可以减少 KVM 进程的迁移次数，从而在进行数据包读取操作时，可以尽量的提高处理器的缓存命中率，就是说，网络 I/O 路径都是在同一个处理器上处理和执行，因此所有的代码和数据存取操作都在同一个处理器上，可以达到很高的缓存局部性，从而提高系统的性能。

V. 结论标题

虚拟化技术在数据中心以及云计算等领域正发挥着越来越重要的作用，但是一方面虚拟化技术在 I/O 虚拟化方面的性能还不能令人满意，还需做进一步的优化。另一方面，随着现在多核技术的普及和应用，如何设计出一个比较好的调度器，既能充分发挥多核的威力，同时又能合理的利用各个核之间的共享资源，减少资源竞争（比如处理器的各级缓存）引起的开销和性能下降，也是一个难题。本文首先对 KVM 的总体架构以及其 I/O 虚拟化方法进行了分析，然后在多核环境中，通过实现进程绑定，对 KVM 网络性能进行了优化，并通过分析工具找出了其性能提高的原因。通过实验，可以看出在 virtio 模式下，通过绑定可以在不减少网络带宽的前提下，将网络延迟降低 10%和 20%左右。

致谢

感谢在北航体系结构研究所的老师和同学们在论文写作过程中所提供的帮助，特别是刘家军同学对自己论文所提供的建议和帮助。

REFERENCES

[1] Kivity, Y.Kamay, D.Laor,and U.Lublin, “kvm : the Linux virtual machine monitor,” Proceedings of the Linux,2007.

[2] BinBin Zhang,Xiaolin Wang, “Evaluating and Optimizing I/O Virtualization in Kernel-based Virtual Machine(KVM)”,NPC 2010,LNCS 6289,pp.220-231

[3] P.Gepner, M.F.Kowalik, “Multi-core Processors: New Way to Achieve High System Performance,” Proceeding of the International Symposium on Parallel Computing in Electrical Engineering, 2006, pp.9-13.

[4] Sergey Blagodurov, Sergey Zhuravlev,Alexandra Fedorova,”Contention-Aware Scheduling on Multicore Systems”,ACM Transaction on Computer Systems,2010

[5] Sergey Zhuravlev,Sergey Blagodurov, Alexandra Fedorova,”Addressing shared resource contention in multicore processors via scheduling”,ASPLOS,2010

[6] S.P. Bhattacharya, V.Apte, “ A measurement study of the Linux TCP/IP Stack Performance and Scalability on SMP on SMP systems,” Communication System Software and Middleware, 2006, pp.1-10.

[7] A.Foong, J.Fung, D.Newell, “An in-depth analysis of the impact of processor affinity on network performance,” 12th IEEE International Conference on Networks, 2004,vol.1,pp.244-250.

[8] A.Foong, J.Fung, D.Newell, “Architectural Characterization of Processor Affinity in Network Processing,” ISPASS 2005, pp.207-218.K. Elissa, “Title of paper if known,” unpublished.

[9] A.Foong, J.Fung, D.Newell, “Improved Linux* SMP Scaling:User-directed Processor Affinity,” Intel Software Network Articles,2011.

[10] M.Faulkner, A.Brampton, S.Pink, “Evaluating the Performance of network Protocol Processing on Multi-core Systems,” AINA 2009, Bradford.

[11] http://www.oprofile.sourceforge.net/news

[12] R.Russell, “virtio: towards a de-facto standard for virtual I/O devices,” SIGOPS Operatomg System Rev, vol.42, 2008, pp.95-103.

[13] Robert Love, “CPU Affinity”, Linux Journal, 2003.

[14] E.Dow, “Take charge of processor affinity,” IBM developer works, 2005

[15] Linux Kernel Document http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[16] http://www.netperf.org/netperf/NetperfPage.html