OpenStackCloudTuningforHighPerformanceComputing

[1]TARINI PRASAD PATNAIK, *Gandhi Institute of Excellent Technocrats, Bhubaneswar, India*
[2]SIBASIS RATH, *Indus College of Engineering, Bhubaneswar, Odisha, India*

***Abstract*—High-Performance computing (HPC) is scarcely attempted in clouds because of slow and inefficient Inter-VM communication on the same server as well as huge latency between remote units. This was changed by introduction of ivshmem, a PCI device–based shared memory between VMs on the same server, but unfortunately, this mechanism became broken with Linux update few years ago. We have restored this shared memory system and made, for the first time, full cloud integration using latest versions of OpenStack, Linux, QEMU, libvirt and MPICH. Also, the analyses of different factors influencing both TCP/IP and ivshmem communication is presented along with tuning techniques that could significantly increase performance. Finally, we have created ivshmem communication channel that can replace standard Neutron TCP/IP network, resulting three to six times performance improvement.**

Keywords-OpenStack;cloudcomputing;highperformancecomputing;cloudtuning

I. INTRODUCTION

Nowadays, Cloud Computing is the dominant general-purposecomputing-paradigm,andOpenStack[1]isthemostpopularopen-sourcecloudoperatingsystemforprivateclouds. Unfortunately, High-Performance-Computing (HPC)in a cloud was not possible in the past, because of the hugeoverhead for inter-VM communication on the same serverand between servers as well, as it was shown for example by[2], [3]. However, with the advent of a remake of ivshmem[4],asharedmemory betweenVMs on thesameserverbecamepossibleagain,thusmakingHPCinprinciplefeasibleforan OpenStackcloud.

In this paper, a set of OpenStack tuning measures arediscussed that augment the possibilities users have for HPCin OpenStack, provided that MPICH [5] is engaged. For thatpurpose, we investigated the case that each MPICH processisallocatedtooneVMandmeasuredtheinter-VMcommunicationonthesameserverbyusingthecallsMPI_PUTfordata-exchangeandMPI_WIN_LOCKfordata-synchronization. Both calls were wrapped by us aroundthe original MPICH calls of the same name in order to beabletoruntheivshmemremakeinOpenStack.

Wewillshowinthefollowingvariousperformancetuning methods for HPC in OpenStack via ivshmem, as wellas for the classical inter-VM communication via TCP/IP thatis based on Neutron‘s [6] Open vSwitch [7] architecture. Byourmeasures,inter-VMbandwidth andlatencycouldbe improved by a factor of up to six, from worst case to bestcase,asourperformancemeasurementshaveshown.

Therestofthepaperisorganizedasfollows: inchapter2,the state of the art in inter-VM communication in OpenStackis given. Chapter 3 presents a tuning for the classical TCP/IPcommunicationthatisbasedonlevel-3cachingandacustomvirtio[8]networkbridge.Chapter4explainstuningmeasures for ivshmem based on proper NUMA allocationand vCPU pinning. In this chapter, also the best TCP/IPmethodiscomparedtothebestivshmemmethoddemonstrating a superior improvement factor for the latter.Thepaperendswithaconclusionandreferencelist.

II. STATEOFTHEARTININTER-VMCOMMUNICATIONINOPENSTACK

As any cloud, OpenStack is a distributed system, even ifthe cloud is physically located in the same rack or in thesame computing center where TCP/IP would not be needed,because L2 switching would be sufficient. By studying [7],wewereabletodrawablockdiagramoftheresultingsoftware overhead (Fig. 1) for the case that two MPICHprocesses are executed by two VMs on the same server.According to Fig. 1, any data frame has to go two timesthrough the following stages: TCP/IP stack, device driver,virtual network interface, qbr Linux Bridge, Open vSwitch(OVS)IntegrationBridge,OVSVLANBridge,physicalEthernet Interface, physical Ethernet Switch. Although OVSis part of OpenStack‘s Neutron network service and certainportconfigurationscouldreducenumberofintermediateinterfaces, it still produces significant overhead. Togetherwith the VM communication overhead, the consequence isanunacceptablelowHPCperformance.

III. TUNINGTHECLASICALTCP/IPINTER-VMCOMMUNICATION

For TCP/IP tuning and the subsequent chapters, we usedOpenStack Juno, Ubuntu 16.04 as guest OS, CentOS 7.1 ashostOS,QEMU2.9.50,libvirt2.0,MPICH3.2andvirtio 1.1.1 as software environment. Additionally, we sent datafrom one MPI process to the other, while varying its sizefrom 2^2 to 2^{20} bytes. The transmission was accomplished byone-sided MPI_PUT via the standard MPICH Nemesis-sockchannel. However, because of the fact that there is normallynosharedmemorybetweendifferentVMs,evenonthesameserver,MPICHemulatesthisfunctionalitybyTCP/IPcommunication, sending data packets back and forth thatcarrysharedvariables.

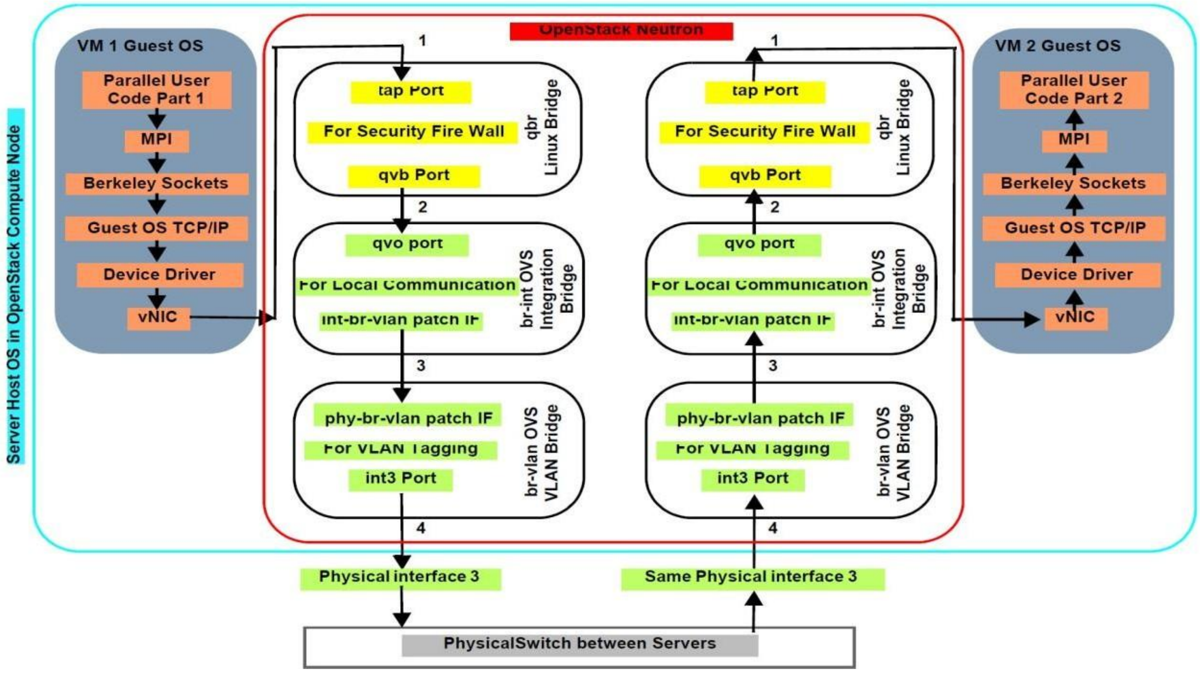

Figure 1.Software overhead in OpenStack for inter-VM communication.1,2,3 =Ethernet Data Frames, 4=VLAN-tagged Ethernet Frames, OVS = OpenvSwitch.

A. Level-3Caching

Now, the influence on OVS of applied level-3 caching isdiscussedincomparisontolevel-2cachingonly.Asreference, we used the elapsed time for transferring data forthe case of no cloud (VMs only) for both, OVS with level-3caching and without. The results are shown in the orange,greyandyellowcurvesofFig.2and Fig.3.

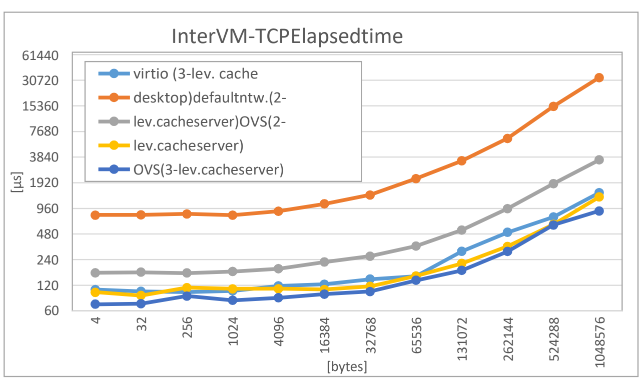

Figure2.ElapsedtimesforTCP/IP-basedinter-VMcommunication.

Thetwoorangecurvesdepicttheelapsedtimeandbandwidth for inter-VM data exchange without cloud. In thiscase, each data packet must go to the first IP router of theserver where it is reflected back thus forming a loop. As onecan see, if OpenStack is engaged with OVS (grey curves),performance is much better than without cloud. The reasonfor that is that each packet must only travel to the first switchoftheserver,asshowninFig.1,andnottothefirstIProuter.E itherattheswitchorattherouter,packetsarereflected back.Engaginglevel-3cachingfurtherimprovesperformance (yellow curves). Similarly, replacing the serverbyadesktopPC(lightbluecurves)showedalsogoodperformance, but disadvantage was limited scalability andcloud incompatibility. Finally, the only measure that madesignificant difference was replacement of Neutron‘s OVS byvirtio-net.

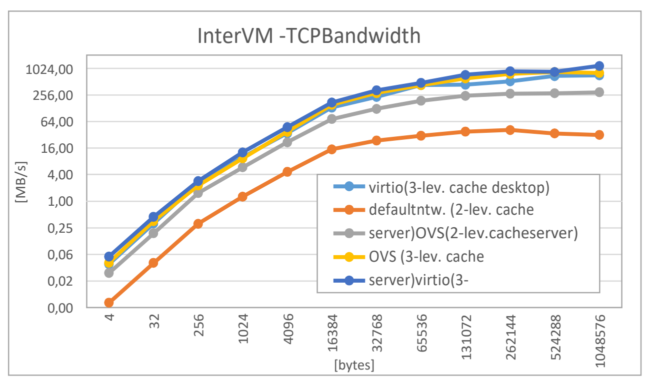

Figure3.BandwidthforTCP/IP-basedinter-VMcommunication.

B. Virtio-Net

The dark blue curves in Fig. 2 and Fig. 3 show the effectofvirtioinsteadofOVS.Thisreplacementispossiblebecaus etheKVMhypervisorofOpenStackhastwointerfaces: the first is used by the various QEMUs [9] itcooperates with. The second API is virtio that provides for“paravirtualization”.ThisisamoreefficientIOvirtualization method than the so-called full software emulation made byOpenStackbymeansofKVM/QEMU.BesidesbeinganAPI, virtio is also a library of paravirtualized device drivers forthe guest OS. Virtio-net is the paravirtualized device-driverfor a virtual Ethernet card (vNIC). Internet-based inter-VMcommunication can profit from virtio-net if a Berkeley sendcall results in calling virtio-net. This driver is aware that it isexecuted in a VM, and it is therefore actively cooperatingwith its QEMU. With virtio-net, KVM does not need tointercept guest-OS device-driver accesses to emulated PCdevices,becausetheyarenotperformed.Instead,theparameters for virtio calls are directly forward to QEMU.Technically, the Berkeley send call is not mapped onto avNIC in guest OS, but via QEMU onto a virtio-net sendqueue(“virtqueue“)inhostOS.Virtqueuesaremuchsimpler and thus faster than vNICs, because they are only buffers.The rest in virtio-net happensas described in section A.Please note also that the paravirtualized guest-OS device-driver is called front-end driver, while the modified host-OSEthernet-driver is termed back-end driver. The original host-OS back-end driver cannot be used in virtio-net because itsinputisaLinuxdata-structurecalledsk_buf,whilethefront-enddriveroutputsso-calledvirtqueueentries.Both,thefront-end and the modified back-end driver, are contained inthe virtio library. The resulting block diagram is shown inFig. 4, where only the first half of the communication isdisplayedbecauseitissymmetric.

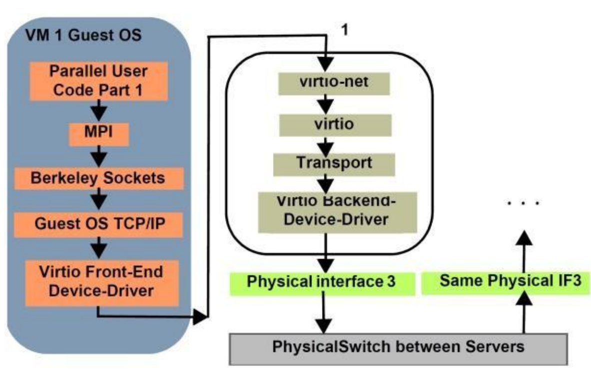

Figure4.Virtio-netinter-VMcommunicationarchitecture.

The first disadvantage of virtio is that TCP/IP is stillengaged. The second is that the virtqueue entries have to behandled by two QEMUs each, one for the source, the otherfor the target VM. Finally, for every data frame sent by thereal Ethernet card, each QEMU has to make a system call toKVM,whichisatime-consumingprocedure,becauseitrequires a full process context switch, with all MMU pagetableentriesreloaded.Becauseofthat,westrivedforabetterinter-VMcommunicationmethod.

IV. IVSHMEM

Ivshmem is a virtual PCI device in a guest OS which isemulatedby KVM/QEMU. Itestablishes aLinux/POSIXshared memory (SHM) between the VM and its host OS.Ivshmemthusenableszero-copyVM-to-Hostcommunication and vice versa, which is very efficient withrespect to bandwidth and latency, because no internal databufferexists.Ivshmemcanalsobeusedforinter-VM communication with the host OS SHM as intermediate step.Ivshmem is implemented by mapping its virtual PCI devicememory to the host OS SHM. This is possible, because thememory is emulated by QEMU as a data structure inside ofitself, and because multiple QEMUs can communicate witheachotherinhostOS.

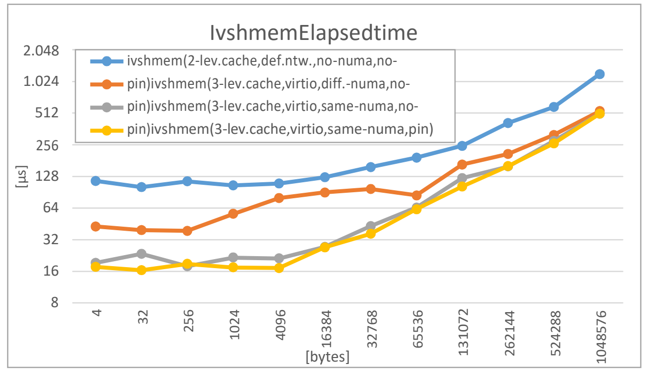

Figure5.Elapsedtimesforivshmem-basedinter-VMcommunication

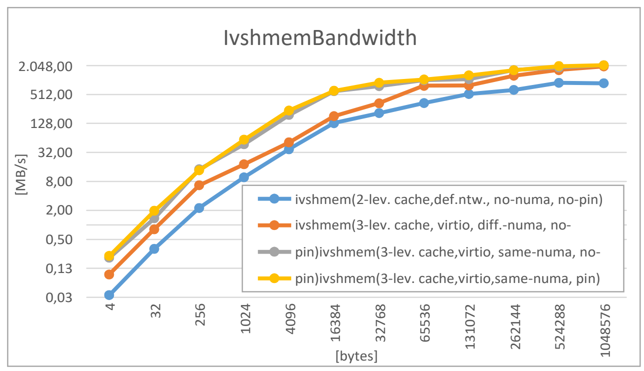

Figure6.Bandwidthforivshmem-basedinter-VMcommunication.

Ivshmem was supported for a few years by libvirt andQEMUasavirtualPCIdevicethatallowedforsharedmemory (SHM) between guests and host. A synchronizationmechanismcalledShared-Memory-Server(SMS)wascreated,whichbecamepartofQEMUasitsivshmem-server.ThisservercouldsendinterruptstoVMslocatedonthesamecompute node, thus allowing unblocking of an application ina target VM, which was waiting for data from an applicationin a source VM. Mutual blocking and unblocking was usedfor SHM access synchronization without consuming CPUcycles(nospinlock),whichisindispensableforHPC.However, this valuable feature could only be implemented ifbothapplicationswereimplementedastheuserpartsofLinux user-IO device-drivers (uio). Additionally, it was thetask of the user to write, by means of code examples of both,the kernel and user part of the uio driver and to integrate hisapplication into this driver. This was a not applicable forHPC with MPI, because MPI processes are typically notdevice drivers. The reason for ivshmem being limited to uiodevice drivers was that the one and only possible spinlock-free synchronization between VMs in Linux is possible bymeansofablockingreadintheuserpartofuio.Assoonas an interrupt arrives at a VM, this user part is called as aninterrupt service routine and the blocking read at the targetVMisunblocked.Unfortunately,withnewerLinuxversions,unblockingdidnotworkanymore,withtheconsequencethativshmemwasnearlyuselessbecauseofmissingSHM-accesssynchronization. Our contribution was to patch codes forboth, the kernel and the user part of uio device drivers forivshmemandtofindproperversionandconfigurationmatches for Linux, QEMU and libvirt so that everythingworksagain.WehavealsocreatedanewSHMcommunicationandsynchronizationchannelinsideofMPICH. In particular, we achieved ivshmem to run for thefirst time in a cloud with OpenStack, resulting in a three tosix times performance improvement compared to TCP/IP(Fig.7andFig.8).

TheisolatedivshmemperformanceresultsaredepictedinFig. 5 and Fig. 6. The blue curves show that implementingdefaultoremulatedTCP/IPnetwork,asrequirementforinitial MPI synchronization, has still very poor performance,but far better than non-SHM solution, shown in Fig. 2 andFig.3. Performanceimprovesfurtherif thesametuningmeasuresare engaged as before, i.e. level-3 caching andvirtio instead of OVS. The effect is shown in the orangecurves.AdditionalgainsarepossiblebyproperNUMAallocationandCPUpinning.

A. PropperNUMAAlocation,no-CPUPinning

In the gray curves of Fig. 5 and Fig. 6, it becomes visiblethat manually allocating the communicating VMs into thesame non-uniform memory-access (NUMA) region yields infurther significant improvements. In such a uniform memoryaccess region, all VMs have the same access latency to thephysicalshared memory in hostOS. On the otherhand,allocating the MPI VMs to different NUMA regions leads tofrequent cache misses, and data access times are not equalanymoreaswell.

B. vCPUPinning

If cache misses occur together with a rescheduling of thevCPU, that executes a VM, from one physical processingunit (i.e. core) to another, the result is a non-monotonouslyincreasingperformanceforincreasedmessagesize.

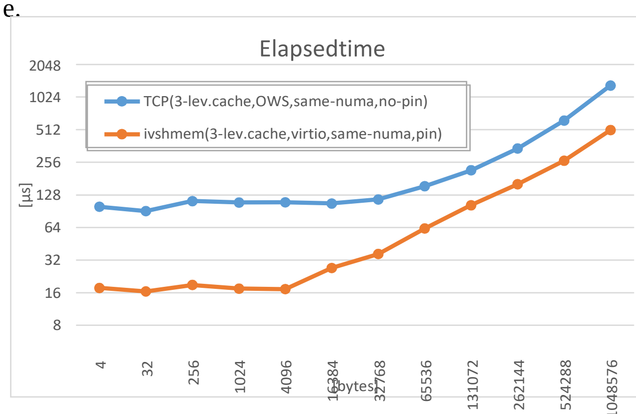

Figure 7.Elapsed times for TCP and ivshmem-based inter-VMcommunication

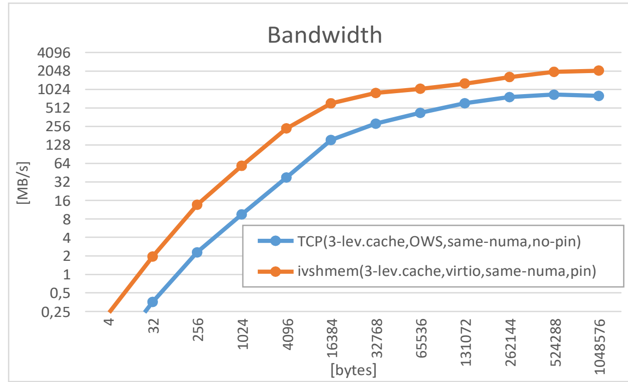

Figure 8.Bandwidth for TCP and ivshmem-based inter-VMcommunication.

This is noticeable in the gray curve of Fig. 5 for sizesfrom 4B to 1KB. Therefore, each VCPU from a VM wasmanually bound by us to a specific physical processing unitwhichisknownas CPUpinning.

The utmost performance boost is achievable if ivshmemis used together with level-3 caching, virtio, proper NUMAallocation and CPU pinning. This is demonstrated by theyellow curves in Fig. 5 and Fig. 6. In this case, bandwidthreaches2GB/sforamessagesizeof1MB,whichistwiceasmuchasforthebestTCP/IPcase.Forsmallermessagesizes,thedifferenceisevenbigger.TheresultswereachievedwithoutusingSR-IOV[10]ashardwareacceleratorforcommunication.

Finally,directcomparisonofstandardTCP/IPcloud(OVS)performancewithourivshmemintegrationisdepictedinFig.7and Fig.8,forElapsedtimeandBandwidth respectively. We have deliberately used same-NUMA region for measurements, since NUMA allocation isdonerandomlybyOpenStackschedulerandwitheachinstance creation could engage different memory region. Forthesmallersizemessages,whensynchronizationisdominatingcommunication,performancedifferenceisfactorofsixinfavorofivshmem. Withfurtherincreaseofmessageblock size, data flow becomes main contributor to overallcommunication time and the difference drops to factor ofthree, for the biggest messages. This is remarkable result,consideringthatNUMAtuningwasnotapplied,duetorandomnatureofOpenStackscheduler.

V. CONCLUSION

High-Performance-Computinginacloudwasnotpossible in the past, because of the huge overhead generatedduringinter-VMcommunication onthesameserverandconsequently between remote units. This was changed withthe advent of a remake of ivshmem, which is a physicalshared memory between VMs on the same host server. Asone of the main results, we made it possible for the first timeto use the ivshmem remake in an OpenStack-based privatecloud. Additionally, we introduced several tuning methods,suchasproperNUMAallocationandCPUpinningandtherefore, adapted ivshmem to perform even better. Also, wegaveeffectivemeasuresforTCP/IP-basedemulationof physicalsharedmemory,whicharelevel-3cachingandvirtio instead of Neutron's Open vSwitch. All methods wereevaluated and compared with each other by measurements,showingthatthebestivshmemscenarioisatleastthreetimesasfastasthebestTCP/IPcase.Allmeasurementswere madewith our wrapper versions of MPICH's MPI_PUT for data-exchangeandMPI_WIN_LOCKforsharedmemorysynchronization.

References

[1] Opensourcesoftwareforcreatingprivateandpublicclouds,https://www.openstack.org/

[2] R. Ledyayev, H. Richter, High Performance Computing in a CloudUsingOpenStack,TheFifthInternationalConferenceonCloudComputing, GRIDs, and Virtualization, CLOUD COMPUTING 2014,http://www.iaria.org/conferences2014/CLOUDCOMPUTING14.html,Venice, Italy,May25-29,2014.

[3] H.Richter,AbouttheSuitability ofCloudsinHigh-PerformanceComputing,Proc.SixthInternationalConferenceonComputerScienceandInformationTechnology(CCSIT2016),JournalComputer Science and Information Technology (CC&IT), Volume 6,Number1,January2016,pp.23-33,VolumeEditors:JanZizka,Dhinaharan Nagamalai, ISBN:978-1-921987-45-8,DOI:10.5121/csit.2016.60103, http://airccj.org/CSCP/vol6/csit64803.pdf,AIRCCPublishingCooperation,Zurich, Switzerland, January02-03,2016.

[4] P. Ivanovic, H. Richter, Performance Analysis of ivshmem for High-Performance Computing in Virtual Machines, Proc. 2nd InternationalConference on Virtualization Application and Technology (ICVAT2017), Shenzhen,China, Nov.17-19,2017.

[5] A. Amer, P. Balaji, W. Bland, W. Gropp, R. Latham, H. Lu, MPICHUser'sGuide,Version3.2,MathematicsandComputerScienceDivision-ArgonneNationalLaboratory, Nov.11,2015

[6] Introduction to OpenStack Networking (neutron),https://docs.openstack.org/liberty/networking-guide/intro-os-networking.html

[7] Open vSwitch in OpenStack,https://docs.openstack.org/liberty/networking-guide/scenario-classic-ovs.html

[8] Virtual I/O Device (VIRTIO) Version 1.0. Committee SpecificationDraft01/PublicReviewDraft01,http://docs.oasis-open.org/virtio/virtio/v1.0/csprd01/virtio-v1.0-csprd01.pdf,Dec.03,2013.

[9] S.Weil,QEMUversion2.10.93UserDocumentation,https://qemu.weilnetz.de/doc/qemu-doc.html

[10] P.Kutch,B.Johnson,SR-IOVforNVFSolutions-PracticalConsiderationsandThoughts,rev.1.0,NetworkingDivision,https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/sr-iov-nfv-tech-brief.pdf,Feb. 23,2017