Implications of Process-Migration in Virtualized Environments

Simon Pickartz
Institute for Automation of Complex Power Systems
RWTH Aachen University
Aachen, Germany
spickartz@eonerc.rwth-aachen.de

Jens Breitbart
Department of Informatics
Chair for Computer Architecture
Technical University Munich
Munich, Germany
j.breitbart@tum.de

Stefan Lankes
Institute for Automation of Complex Power Systems
RWTH Aachen University
Aachen, Germany
slankes@eonerc.rwth-aachen.de

ABSTRACT

The exploitation of the whole computing power of current supercomputers is only achieved by few High-Performance Computing (HPC) applications. We do not expect this issue to be automatically solved with future hardware. One technique to achieve excellent system utilization is *co-scheduling*, where at least two applications with divers resource requirements share the resources of one compute node. Co-scheduling enabled by Virtual Machine (VM) migration is able to improve runtime and energy consumption of HPC applications. In this paper we investigate the impact of full-virtualization on the performance of intra-node communication between VMs for various VM counts. Our analysis reveals that compute-bound applications can achieve up to 97 % of the native performance when executed within 16 VMs while communication intensive operations such as collectives suffer from increased latencies by a factor up to 16. The results can be used as decision-making guidelines for the scheduling system to find suitable solutions for the overall system performance.

Keywords

Virtualization, Intra-Host, Co-scheduling, MPI, HPC

1. INTRODUCTION

We notice that common HPC applications are only capable of exploiting a fraction of the compute power offered by today's supercomputers. Although, some highly tuned applications are able to get close to the systems's peak performance, most applications are limited by a single resource, e. g., I/O or memory bandwidth. This characteristic is not expected to change with upcoming hardware, rather will the increasing gap between computing power and I/O performance [8] result in even more applications not being able to utilize all available resources.

One of the main goals of compute centers is the maximazation of the overall system utilization as it allows more scientists to conduct their research. A way to achieve this goal without manually tuning all applications used at a system, is *co-scheduling* [6], i. e., running two or more applications with divers resource demands in parallel on an overlapping set of nodes. Co-scheduling may benefit from adaptations to the schedule during the runtime as jobs come and go, each exhibiting individual resource demands. Such a dynamic schedule can only be realized if it is possible to move already running processes across nodes of the system. In previous studies we have investigated different techniques enabling such migrations and found full-virtualization, e. g., VMs based on KVM, to provide a good trade-off between performance and flexibility in terms of a greater application range [16].

However, identifying the correct size of a VM , i. e., the amount of Virtual CPUs (VCPUs), is non trivial. On the one hand, large VMs limit the flexibility of the scheduler as it can only migrate whole VMs from one node to another. Small VMs on the other hand may result in the execution of multiple VMs with the same application on the same host system, which slows down communication between the processes of the application. This slowdown is due to the fact that processes running natively on the host or within the same VM can communicate via shared memory, whereas inter-VM communication is typically handled via (virtual) network interfaces. In this paper we investigate the impact of full-virtualization on the performance of intra-host communication. Therefore, we compare the performance of different benchmarks and an example HPC application executed natively to their execution in one or more VMs running on the same host system.

Our results show that the influence of intra-host inter-VM communication on the applications' performance is highly dependent on their characteristics. Compute-intensive benchmarks achieve up to 97 % of the native performance when executed within multiple VMs on the same host. However, communication bound applications are slowed down by up to 26 % in our studies. From a microbenchmark analysis we could conclude that especially collective operations would benefit from a locality-aware communication layer. Here, the latency was increased by a factor of up to 16.

This paper is structured as follows: The following three sections give a detailed overview of the hard- and software setup used for our experiments. Section 5 presents the performance analysis results of selected benchmarks and applications. Before concluding the paper (Sect. 7), we discuss related work in Sect. 6.

COSH 2016 Jan 19, 2016, Prague, CZ

DOI: http://dx.doi.org/10.14459/2016md1320971

2. HARDWARE

We ran all tests on a two-socket NUMA node equipped with Intel IvyBridge CPUs (E5-2650 v2) clocked at 2.6 GHz. Each CPU possesses 8 cores resulting in a total of 16 CPU cores in the entire system. One CPU core has support for two hardware thread contexts (HTC, often called Hyper-threading), i. e., a total of 32 HTCs are provided by the whole system. An HTC has its own set of registers but shares the instruction pipeline and both L1 and L2 caches with the second HTC of the same core. The instruction pipeline has dedicated hardware for floating point, integer, and SIMD instructions, which can be co-issued with various constrains. The L3 cache is shared among all CPU cores of a CPU.

Our platform supports Intel's virtualization technologies VT-x and VT-d [17, 2]. The former extends the ring concept of the x86-architecture by the so-called *non-root* execution mode. This allows for guest Operating Systems (OSs) to run at Ring 0—the most privileged level which is typically used for OS kernels—but with certain restrictions. As OSs are usually written with the assumption of having full control over the hardware, their execution at a different privilege level might result in unexpected or faulty behavior. The non-root execution mode gives a guest OS the impression of owning the hardware while still allowing the host kernel to intercept operations that should not be permitted to the guests. This hardware assisted virtualization provides computing power within VMs close to that of native execution.

However, in contrast to the processor and memory, the virtualization of I/O devices is still a challenge. Each interaction of a guest system with its devices requires I/O operations. For common hardware such as standard Gigabit Ethernet Network Interface Cards (NICs) it is possible to emulate these devices in software. However, this approach fails for high-performance networks, e. g., InfiniBand (IB), only reaching about 50 % of the native performance [13]. To overcome these performance penalties, Intel introduced the VT-d extensions providing guests direct access to the real hardware. It avoids expensive guest-to-host transitions every time the guest accesses its devices by granting direct access to the respective control registers. However, this technique by itself does not enable virtualized HPC clusters as one device can only be passed to one guest at a time. Hence, each VM communicating over IB would require an individual Host Channel Adapter (HCA).

This problem was identified by the Peripheral Component Interconnect Special Interest Group (PCI-SIG) proposing Single Root I/O Virtualization (SR-IOV) as an extension to the PCIe standard [10]. SR-IOV allows for hardware supported I/O device multiplexing by introducing two PCIe function types: Physical Functions (PFs) and Virtual Functions (VFs). The latter are a pared-down version of the PF providing all PCIe capabilities necessary for data movement. The compute node used for our evaluation is equipped with a two-port Mellanox ConnectX-3 IB adapter with support for SR-IOV. It allows for the creation of up to 16 VFs, i. e., the adapter can be attached to 16 VMs at a time.

3. KERNEL-BASED VIRTUAL MACHINE

We used Kernel-based Virtual Machine (KVM) as virtualization solution to perform our evaluation [15]. This hypervisor implements full-virtualization for the x86 architecture based on Intel's VT-x extension described in the previous section or AMD's virtualization extension (AMD-V). A VM is an ordinary processes from the hypervisor's point of view and can be treated like any other process running on the system. Similar to real hardware, it can be equipped with multiple VCPUs which are mapped onto threads of the KVM process representing the VM.

In contrast to other hypervisors, e. g., Xen [4], KVM only implements the necessary components for the virtualization of the CPU and the main memory. Other parts of the computer system have to be emulated in software. Therefore, KVM is usually deployed in conjunction with the user-space emulator QEMU [5].

The virtualization of I/O devices is facilitated by means of the Intel VT-d extensions and SR-IOV. The former allow for the pass-through of PCIe devices to VMs both prior the boot time and during the runtime of the VM (*hot-plugging*). This is an important feature for VM migration in HPC environments, as VMs cannot be migrated with an attached pass-through device, but pass-through devices must be unplugged from a VM prior migration. With SR-IOV it is possible to attach the same physical PCIe devices to multiple VMs at a time. Support for both SR-IOV and hot-plugging allows for near native IB performance, but also enables the scheduler to migrate VMs for the optimization of the overall system utilization.

Modern HPC systems are typically built by using NUMA systems. Software running on top should be NUMA-aware for a good exploitation of such architectures, i. e., threads and processes should be scheduled such that remote memory access is avoided as far as possible. KVM brings NUMA-awareness in terms of NUMA topologies that can be defined for the guests running on top. Therefore, it is possible to comprise one or more VCPUs in so-called *cells* which are recognized as NUMA domain by the guest system. Furthermore, memory policies can be imposed to these cells. Thereby restrictions to memory placement can be defined enabling the creation of a complete reflection of the host's topology to that of the guest.

4. TEST APPLICATIONS

This section briefly introduces the benchmarks and test application that have been used for the evaluation of intra-host inter-VM communication. They have been built with the Intel compiler and were executed by using Intel MPI.

4.1 Microbenchmarks

For the analysis of key figures assessing the communication performance of intra-host MPI communication among multiple VMs, we used both a self-written benchmark and a selection of the low-level benchmarks from the Intel MPI Benchmarks (IMB) [1]. The self-written benchmark[1] determines point-to-point latency and bandwidth by exchanging messages between two processes in a *PingPong* pattern [1]. From the IMB we use a set of benchmarks for the evaluation of the performance of MPI collective operations. We select the following five collective operations:

barrier as it is an indispensable collective operation for the synchronization of a set of MPI processes.

[1] https://github.com/RWTH-OS/mpi-benchmarks

broadcast is a common parallel building block, e. g., used at the beginning of solver phases, when a dataset must be send from the master process to all processes of a job.

allreduce is an operation combining partial results spread among different MPI processes into a final result, e. g., it is used at the end of a solver phase.

allgather only collects the partial results without combining them.

alltoall is an extension to allgather allowing for the distribution of distinct data to the receiving processes and typically involves the most communication.

The exact implementation these benchmarks within Intel MPI is unknown, however these collectives are typically implemented using a communication tree for a reduction of the required messages. The mapping of the tree onto MPI processes and there nodes or VMs (in our case) can influence the performance of the collective operation.

4.2 NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPBs) [3] is a suite of different computing kernels that are commonly used by large-scale fluid dynamics applications. It offers varying problem classes suiting different cluster sizes. For our tests we chose Class C, depicting a reasonable size for a small test system like the one used for our work.

Originally, the suite contained eight different benchmarks comprising five computing kernels and three so-called pseudo applications. From the kernels we chose FT and CG. The first computes a discrete 3D Fourier Transformation. This is a communication intensive kernel exhibiting an all-to-all communication pattern. CG computes the approximation to the smallest eigenvalue of a large sparse matrix by using a conjugate gradient method. This benchmark is characterized by irregular memory accesses and communication. Furthermore, we evaluated the three pseudo applications BT, LU, and SP which are basically solvers for equation systems.

As our work considers applications that are partly migrated among nodes in a cluster as a result of co-scheduling, we used the MPI implementation of the presented kernels. However, the three pseudo application exist as *multi-zone* version [18], as well. These solve the equation systems on loosely coupled discretization meshes and are intended for the evaluation of hybrid parallelization approaches. The OpenMP+MPI implementation solves the individual zones in parallel in accordance with the shared-memory paradigm while the exchange of boundary values between these zones is performed by means of message-passing. Therefore, they are ideal application benchmarks for our test scenario as many HPC applications exploit HPC systems by applying different parallelization approaches at the same time.

4.3 MPIBlast

We use a slightly modified version of MPIBlast 1.6.0[2] as a real world application for our analysis. Using MPI-only, it is a parallel version of the original BLAST (Basic Local Alignment Search Tool) algorithm from computational biology for the heuristical comparison of local similarities between genome or protein sequences from different organisms.

[2]http://mpiblast.org/

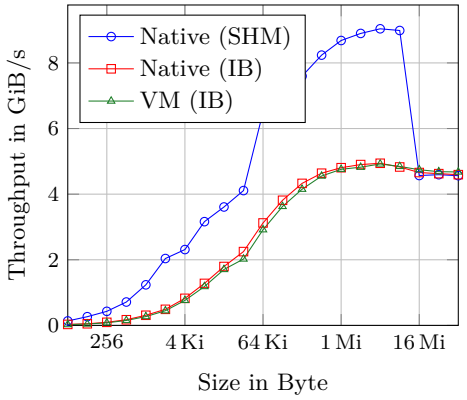


Figure 1: Throughput Overhead

Due to its embarrassingly parallel nature using a nested master-slave structure, MPIBlast allows for perfect scaling across tens of thousands of compute cores [7]. The MPI master processes hand out new chunks of workload to their slave processes whenever previous work gets finished. This way, automatic load balancing is applied. MPIBlast uses a two-level master-slave approach with one so-called super-master responsible for the whole application and possibly multiple masters distributing work packages to slaves. As a result, MPIBlast must be always run with at least 3 processes of which one is the super-master, one is the master, and one being a slave. We used only one master for all our benchmarks and communication mostly only happens between the master and the slave processes. The data structures used in the different steps of the BLAST search typically fit into L1 cache, resulting in a low number of cache misses. The search mostly consists of a series of indirections resolved from L1 cache hits, allowing for a good overlapping of different searches on the 2 HTCs of one core. Our modified version of MPIBlast is available at GitHub[3]. In contrast to the original MPIBlast 1.6.0 we removed all `sleep()` functions calls that were supposed to prevent busy waiting. On our test-system, this resulted in underutilization of the CPU. Removing sleeps increased performance by about a factor of 2. Furthermore, our release of MPIBlast updated the Makefiles for the Intel Compiler to utilize inter-procedural optimization which also resulted in a notable increase in performance.

5. EVALUATION

We measured the results of our self-written benchmark for the following three scenarios to understand the performance penalties when running MPI processes within multiple VMs:

native (SHM) shared memory communication on the host

native (IB) communication using IB on the same host

VM (IB) communication between processes residing in different VMs using IB with SR-IOV

For native (SHM) the latency between two MPI processes running on the same CPU socket is 0.27 µs. This value drastically increases to 1.37 µs for native (IB). However, for VM (IB) the overhead of the virtualization layer and SR-IOV

[3]https://github.com/jbreitbart/mpifast

Table 1: MPI Barrier (Latency in µs)

Native	1 VM	2 VMs	4 VMs	8 VMs	16 VMs
2.05	2.10	8.07	19.44	9.10	13.58

is hardly notable. The additional latency for VM (IB) of 0.02 µs is rather small. These results are similar when measuring the throughput between two MPI processes (cf. Fig. 1). The maximum bandwidth of VM (IB) is only at around 45 % of that in native execution.

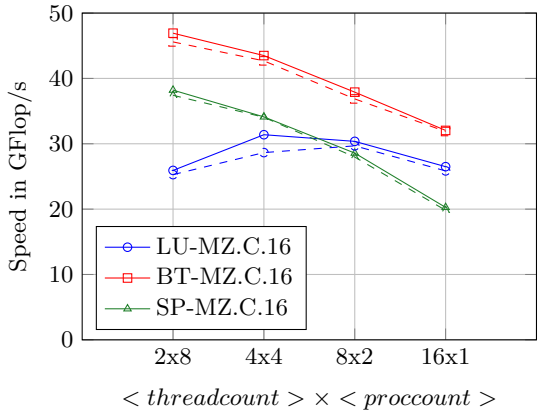


Figure 2: NPB-MZ Native (solid) vs. VM (dashed)

5.1 Collectives

To assess the impact of multiple VMs on communication intensive applications, we investigated a set of collective operations, namely: barrier, allgather, alltoall, allreduce, and broadcast (cf. Tab. 1 and Fig. 4). All benchmarks were started with 16 MPI processes each pinned to an individual core. The results present a scalability study over 2, 4, 8, and 16 VMs, i. e., 8, 4, 2, and 1 processes per VM respectively. Each VM has been equipped with one of the VFs that are available on our test system. As a result, inter-VM communication is using IB as transport, whereas ranks running on the same VM communicate via shared-memory segments. In all cases a compact pinning of ranks to VMs and cores has been performed, e. g., in the scenario with 2 VMs Ranks 0 to 7 have been started on one VM while the remaining ranks resided in the second.

Using more than one VM clearly increases the latency of the barrier by a factor of around 4. This is due to IB communication that takes place between some processes. However, adding more VMs has a moderate influence on the latency with additional 5.51 µs for 16 VMs running on the same host, i. e., instead of having a mixture of shared-memory (intra-VM) and IB (inter-VM) communication, all processes synchronize over IB. The peak of 19.44 µs in the case of 4 VMs might arise from a suboptimal distribution of ranks to VMs. The latency in this scenario can be improved to 10.68 µs by using a scatter pinning which. This results in a different communication scheme as the communication tree is most probably distributed differently across the VMs. However, we have to investigate that point in more detail to get a clearer picture.

The results of the other collective operations (cf. Fig. 4) reveal that their execution within more than one VM often results in significantly increased latencies. For example the broadcast operation is throttled by a factor of 1.5 when the 16 processes are distribted across two VMs compared to native (SHM) for small messages. This factor increases to 6.3 for the 16-VM case. However, for the 2- and 4-VM case the latencies converge at least for large messages of 4 MiB. The *Alltoall* operation within 16 VMs is throttled by a factor of 16 for small messages. However, again for larger messages this discrepancy decreases to a factor of around 4.5.

From the presented microbenchmark analysis it can be concluded that the execution of a communication bound MPI job on multiple VMs running on the same host can impose important performance degradation. This is mostly true for applications exchanging small-size messages and should be considered when taking any scheduling decisions.

5.2 Applications

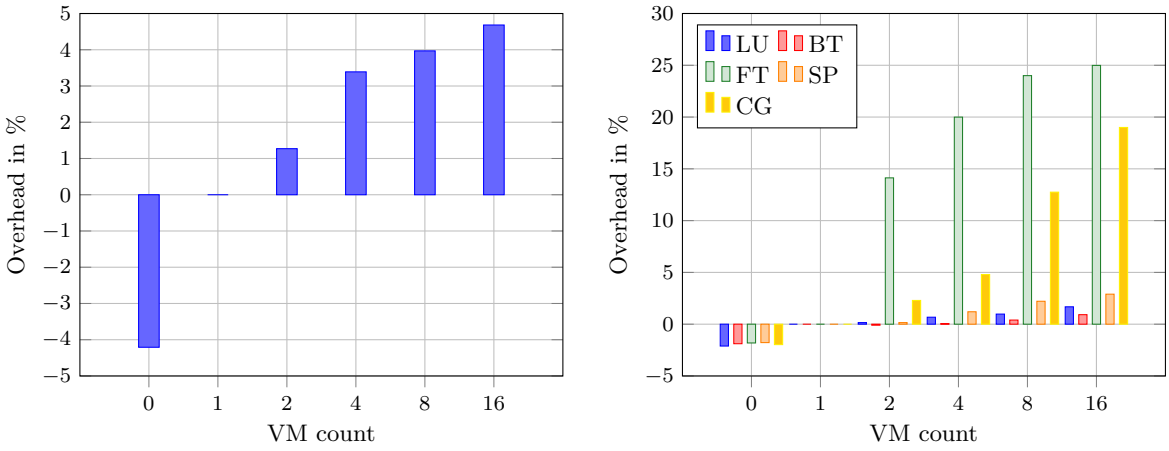
The previously shown microbenchmarks suggest that co-scheduling using VMs on HPC clusters result in significant performance penalties if the MPI library does not come along with efficient intra-host inter-VM communication. In this section we evaluate the performance hit for the applications described in Sect. 4.

In the first test case, 32 MPIBlast processes run equally distributed on a different number of VMs. All VMs run on the same compute node and use InfiniBand as inter-VM communication channels. Figure 3a shows the performance differences between the various configurations. The usage of 0 VMs means that all processes run natively on the host system communicating over shared-memory segments. As MPIBlast is a compute-bound application and the communication channel does not constitute its bottleneck, the performance differences between the configurations are rather small.

Figure 3b shows the results of a similar test scenario, in which the MPI-versions of the NPBs were divided equally over VMs. Overall, the usage of multiple VMs is not a performance drawback decreasing the performance of the pseudo applications by only 1 % to around 3 % for the 16 VM case related to the 1-VM case. The slower communication interface between VMs in comparison to native execution is only clearly noticeable in the more communication intensive kernels CG and FT.

The next test case is the *multi-zone* version of the NPB. They were started on one compute node for native (SHM) and within multiple VMs on that node for VM (IB), i. e., the process count equals the VM count for the latter case. However, the relationship between processes and threads changed between every run. Figure 2 shows that the best performance can be achieved if the benchmark uses more processes rather than threads. The usage of a message passing interface reduces the number of side effects such as *False Sharing* and contention on synchronization primitives. Furthermore, the memory allocation strategy is simplified. The processes are bound to a single NUMA node and always allocate the memory on its node guaranteeing local memory accesses. In the case of using one process and 16 threads, applications have to use NUMA-aware allocation strategies to achieve best performance [11].

In the case, that the benchmarks are running natively on the host system, a shared memory region is used for inter-process communication. If the MPI processes run within different VMs, a shared memory interface is missing and IB is used as communication channel for the inter-VM com-



(a) MPIBlast (32 Processes)

(b) NPB Class C (16 Processes)

Figure 3: Overhead when running MPI-only applications across multiple VMs compared to the execution within one VM.

munication. Figure 2 reveals that the performance degradation by using multiple VMs is small. Consequently, the performance of the multi-zone version of NPB does not depend on small messages (smaller than the cache size), which clearly is more efficient by using a shared memory interface (cf. Fig. 1).

6. RELATED WORK

Overall there has been not much research on intra-host inter-VM communication. Typically, studies focus either on the comparison of different virtualization solutions in general [19], or they investigate the impacts of I/O virtualization on inter-node communication [14, 12].

Zhang et. al proposed a design of a locality-aware MPI library [21, 20]. Their implementation extends MVAPICH2 [9] by a *locality detector* enabling communication over shared-memory segments between processes residing in different VMs on the same host. Focusing on the performance benefits of Inter-VM Shared Memory (IVShmem) over SR-IOV communication they perform a comprehensive performance evaluation of inter-VM communication using either of the two mechanisms.

7. CONCLUSION

This paper explores the applicability of virtualization as driver for co-scheduling applied to HPC. We estimate the impact of the VM size on the performance of HPC applications by conducting scalability studies over different VM counts but with a fixed amount of processes, i. e., the varying VM granularity has a direct influence on the ratio between shared-memory and IB communication. Depending on the application's characteristics, a scheduler might decide to host multiple VMs of the same job on one node without taking high performance losses.

However, especially the latency of collective operations suffer from the IB communication channel between VMs. Therefore, we plan to work on locality-awareness of the MPI layer. This should not only comprise inter-VM communication over shared-memory but also adoptions of the communication channels to dynamic re-schedules that might occur during runtime.

8. ACKNOWLEDGMENTS

This research and development was supported by the Federal Ministry of Education and Research (BMBF) under Grant 01IH13004B (Project FAST).

9. REFERENCES

[1] Intel MPI Benchmarks. Technical report, Intel Corporation, 2014.

[2] Intel Virtualization Technology for Directed I/O. Technical report, Intel Corporation, 2014.

[3] D. H. Bailey et al. The NAS Parallel Benchmarks. *Int. Journal of High Performance Computing Applications*, Sept. 1991.

[4] P. Barham et al. Xen and the Art of Virtualization. *SIGOPS Oper. Syst. Rev.*, 2003.

[5] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference, FREENIX Track*, 2005.

[6] J. Breitbart et al. Case Study on Co-Scheduling for HPC Applications. In *Proc. Int. Workshop Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS 2015)*, Sept. 2015.

[7] A. Darling et al. The design, implementation, and evaluation of mpiBLAST. In *Proc. of ClusterWorld*, 2003.

[8] J. Dongarra. Impact of Architecture and Technology for Extreme Scale on Software and Algorithm Design. In *Euro-Par 2010 – Parallel Processing*, Aug. 2010. Euro-Par 2010 Keynote.

[9] W. Huang et al. Design of High Performance MVAPICH2: MPI2 over InfiniBand. In *Proc. 6th IEEE Int. Symp. Cluster Computing and the Grid*, CCGRID '06, 2006.

[10] Intel LAN Access Division. PCI-SIG SR-IOV Primer. Technical Report 2.5, Intel Corporation, Jan. 2011.

[11] S. Lankes et al. Node-Based Memory Management for Scalable NUMA Architectures. In *Proc. 2nd Int. Workshop Runtime and Operating Systems for Supercomputers (ROSS 2012) in conjunction with 26th Int. Conf. Supercomputing (ICS 2012)*, 2012.

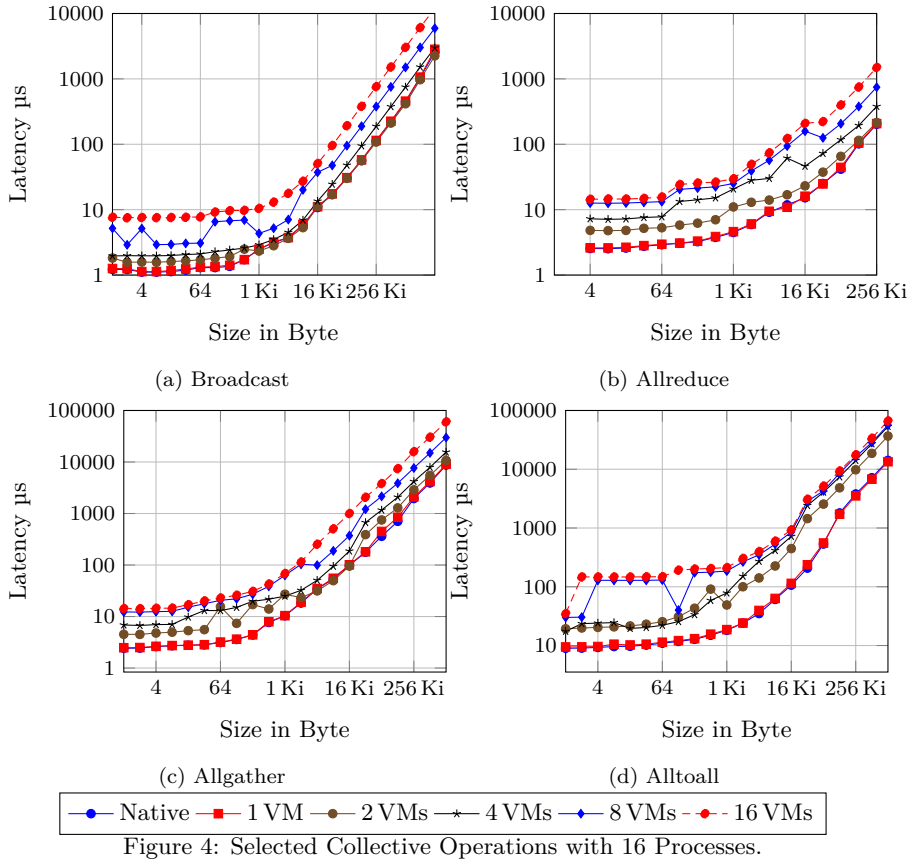


Figure 4: Selected Collective Operations with 16 Processes.

[12] J. Liu et al. High Performance VMM-Bypass I/O in Virtual Machines. In *USENIX Annual Technical Conference, General Track*, 2006.

[13] Y.-M. Ma et al. InfiniBand virtualization KVM. *CloudCom*, 2012.

[14] M. Musleh et al. Bridging the Virtualization Performance Gap for HPC Using SR-IOV for InfiniBand. *IEEE CLOUD*, 2014.

[15] L. Nussbaum et al. Linux-based virtualization for HPC clusters. In *Proc. Linux Symposium*, July 2009.

[16] S. Pickartz et al. Migration Techniques in HPC Environments. In *Euro-Par 2014: Parallel Processing Workshops*, Lecture Notes in Computer Science. 2014.

[17] R. Uhlig et al. Intel Virtualization Technology. *Computer*, 38, May 2005.

[18] R. F. Van der Wijngaart and H. Jin. Nas parallel benchmarks, multi-zone versions. *NASA Ames Research Center, Tech. Rep. NAS-03-010*, 2003.

[19] A. J. Younge et al. Analysis of Virtualization Technologies for High Performance Computing Environments. In *Cloud Computing (CLOUD), 2011 IEEE Int. Conf.*, 2011.

[20] J. Zhang et al. Can Inter-VM Shmem Benefit MPI Applications SR-IOV Based Virtualized Infiniband Clusters? In *Euro-Par 2014 Parallel Processing*. Jan. 2014.

[21] J. Zhang et al. High performance MPI library over SR-IOV enabled infiniband clusters. In *2014 21st Int. Conf. High Performance Computing (HiPC)*, 2014.