Noname manuscript No.
(will be inserted by the editor)

Evaluation of Messaging Middleware for High Performance Cloud Computing

Roberto R. Expósito · Guillermo L. Taboada · Sabela Ramos · Juan Touriño · Ramón Doallo

Submitted: March 3, 2012

Abstract Cloud computing is posing several challenges, such as security, fault tolerance, access interface singularity, and network constraints, both in terms of latency and bandwidth. In this scenario, the performance of communications depends both on the network fabric and its efficient support in virtualized environments, which ultimately determines the overall system performance. To solve the current network constraints in cloud services their providers are deploying high-speed networks, such as 10 Gigabit Ethernet. This paper presents an evaluation of High Performance Computing message-passing middleware on a cloud computing infrastructure, Amazon EC2 cluster compute instances, equipped with 10 Gigabit Ethernet. The analysis of the experimental results, confronted with a similar testbed, has shown the significant impact that virtualized environments still have on communications performance, which demands more efficient communication middleware support to get over current cloud network limitations.

Keywords Cloud Computing · High Performance Computing · Virtualization · 10 Gigabit Ethernet · Message-Passing Middleware · Performance Evaluation

1 Introduction

Cloud computing is a model that enables convenient, on-demand and self-service access to a shared pool of highly scalable, abstracted infrastructure that hosts applications, which are billed by consumption. This computing paradigm is changing rapidly the way enterprise computing is provisioned and managed, thanks to the commoditization of computing resources (e.g., networks, servers,

Roberto R. Expósito · Guillermo L. Taboada · Sabela Ramos · Juan Touriño · Ramón Doallo
Dept. of Electronics and Systems, University of A Coruña, A Coruña, Spain.
E-mail: {rreye,taboada,sramos,juan,doallo}@udc.es
Corresponding author: Roberto R. Expósito

storage and applications) which provide cost-effective solutions and efficient server virtualization [5]. Moreover, cloud computing technologies can be useful in a wide range of applications such as email, file storage or document clustering [39], among other domains [20]. However, this model is posing several challenges, such as security [35], heterogeneity of the cloud management frameworks, and handling network constraints, both in terms of latency (the cost of sending a message with minimal size through the network) and bandwidth, that limit the scalability of the cloud resources.

In cloud computing the performance of communications depends both on the network fabric and its efficient support in cloud middleware, which ultimately determines the overall system performance. Cloud infrastructures typically relied on a virtualized access to Gigabit Ethernet and the use of TCP/IP stack, a combination that provides poor performance [33], especially when the underlying infrastructure consists of systems with an increasing number of cores per processor due to the poor ratio between CPU power and network performance. To solve these network constraints in cloud services their providers are deploying high-speed networks (e.g., 10 Gigabit Ethernet and InfiniBand).

High-speed networks have been traditionally deployed in High Performance Computing (HPC) environments, where message-passing middleware is the preferred choice for supporting communications across distributed memory systems. MPI [1] is the standard interface for programming message-passing applications in languages compiled to native code (e.g., C/C++ and Fortran), whereas MPJ [7, 31] is the messaging middleware for Java applications. These HPC applications, whose scalability depends on low-latency communications [37], generally achieve good performance on clusters with high-speed networks, whereas they suffer significant performance bottlenecks on virtualized environments, especially networking overheads, at public cloud infrastructures.

Amazon Elastic Compute Cloud (Amazon EC2) [21] provides users with access to on-demand computational resources to run their applications. EC2 allows scalable deployment of applications by providing a Web service through which a user can boot an Amazon Machine Image (AMI) to create a custom Virtual Machine (a VM or "instance"). Cluster compute instances [22], a resource introduced in July 2010, provide the most powerful CPU resources with increased network performance, which is intended to be well suited for HPC applications and other demanding network-bound applications. This is particularly valuable for those applications that rely on messaging middleware like MPI for tightly coupled inter-node communication.

This paper presents an evaluation of HPC message-passing middleware on a cloud computing infrastructure, Amazon EC2 cluster compute instances, with a high-speed network, 10 Gigabit Ethernet, in order to assess their suitability for HPC. Nevertheless, the experimental results have shown several performance penalties, such as the lack of efficient network virtualization support and a proper VM-aware middleware adapted to cloud environments.

The structure of this paper is as follows: Section 2 introduces the related work. Section 3 describes the network support in virtualized environments, the building block in cloud infrastructures. Section 4 introduces the message-passing middleware considered in this work. Section 5 analyzes the performance results of the selected middleware on HPC cluster instances of Amazon EC2 cloud, compared to our private cloud testbed with a similar configuration. These results have been obtained from a micro-benchmarking of point-to-point primitives, as well as an application benchmarking in order to analyze the scalability of HPC applications on cloud services. Section 6 summarizes our concluding remarks and future work.

2 Related Work

Typically, computationally intensive codes present little overhead when running on virtualized environments, whereas I/O bound applications, especially network intensive ones, suffer significant performance losses [14]. Thus, message passing applications whose scalability heavily depends on start-up latency performance were initially highly inefficient in such virtualized environments.

There have been many studies of virtualization techniques in the literature, including performance enhancements focused on reducing this I/O overhead. Paravirtualization [36] was introduced in order to reduce the performance overhead associated with emulated I/O access to virtual devices. Liu et al. [19] describe the I/O bypass of the Virtual Machine Monitor (VMM), or hypervisor, using InfiniBand architecture, extending the OS-bypass mechanism to high-speed interconnects. The use of this technique has shown that Xen hypervisor [8] is capable of near-native bandwidth and latency, although it has not been officially integrated in Xen so far. Nanos et al. [27] developed Myrixen, a thin split driver layer on top of the Myrinet Express (MX) driver to support message passing in Xen VMs over the wire protocols in Myri-10G infrastructures.

A VM-aware MPI library was implemented in [15], reducing the communication overhead for HPC applications by supporting shared memory transfers among VMs in the same physical host. Mansley et al. [24] developed a direct data path between the VM and the network using Solarflare Ethernet NICs, but its operation is restricted to these devices. Raj et al. [28] describe network processor-based self via specialized network interface cards to minimize network overhead.

Hybrid computing concept was studied in [25]. They have examined how cloud computing can be best combined with traditional HPC approaches, proposing a hybrid infrastructure for the predictable execution of complex scientific workloads across a hierarchy of internal and external resources. They presented the `Elastic Cluster` as a unified model of managed HPC and cloud resources.

Additionally, some works have already evaluated the performance of cloud computing services, such as Amazon EC2. Wang et al. [34] present a quantitative study of the end-to-end networking performance among Amazon EC2 medium instances, and they observed unstable TCP/UDP throughput caused by virtualization and processor sharing. In [33], Walker evaluates the performance of Amazon EC2 for high-performance scientific applications, reporting that Amazon has much worse performance than traditional HPC clusters. Walker used only up to 32 cores from Amazon EC2 high-CPU extra large instances, the most powerful CPU instances in 2008, with a standard Gigabit Ethernet interconnection network. His main conclusion was that the cloud computing service was not mature for HPC at that moment.

The suitability for HPC of several virtualization technologies was evaluated in [29], showing that operating system virtualization was the only solution that offers near native CPU and I/O performance. They included in their testbed four Amazon EC2 cluster compute instances, interconnected via 10 Gigabit Ethernet, although they focused more on the overall performance of the VM instead of the scalability of HPC applications.

3 Network Support in Virtualized Environments

The basic building blocks of the system (i.e., CPUs, memory and I/O devices) in virtualized environments are multiplexed by the Virtual Machine Monitor (VMM), or hypervisor. Xen [8] is a popular high performance VMM, used by Amazon EC2 among other cloud providers. Xen systems have a structure with the Xen hypervisor as the lowest and most privileged layer. Above this layer comes one or more guest operating systems, which the hypervisor schedules across the physical CPUs. The first guest operating system, called in Xen terminology domain 0 (dom0), boots automatically when the hypervisor boots and receives special management privileges and direct access to all physical hardware by default. The system administrator can log into dom0 in order to manage any further guest operating systems, called domain U (domU) in Xen terminology.

Xen supports two virtualization technologies:

- **Full Virtualization (HVM):** this type of virtualization allows the virtualization of proprietary operating systems, since the guest system's kernel does not require modification, but guests require CPU virtualization extensions from the host CPU (Intel VT [16], AMD-V [4]). In order to boost performance fully virtualized HVM guests can use special paravirtual device drivers to bypass the emulation for disk and network I/O. Amazon EC2 cluster compute instances use this Xen virtualization technology.
- **Paravirtualization (PV):** this technique requires changes to the virtualized operating system to be hypervisor-aware. This allows the VM to coordinate with the hypervisor, reducing the use of privileged instructions that are typically responsible for the major performance penalties in full virtualization. For this reason PV guests usually outperform HVM guests. Paravirtualization does not require virtualization extensions from the host CPU.

VMs in Xen usually do not have direct access to network hardware, except using PCI passthrough technique (see next paragraph) or with third-party specific support (like [19] for InfiniBand and [27] for Myrinet). Since most existing device drivers assume a complete control of the device, there cannot be multiple instantiations of such drivers in different guests. To ensure manageability and safe access, Xen follows a split driver model [11]. Domain 0 is a privileged guest that accesses I/O devices directly and provides the VMs abstractions to interface with the hardware. In fact, dom0 hosts a backend driver that communicates with the native driver and the device. Guest VM kernels host a frontend driver, exposing a generic API to guest users. Guest VMs need to pass the I/O requests to the driver domain to access the devices, and this control transfer between domains requires involvement of the VMM. Therefore, Xen networking is completely virtualized. A series of virtual Ethernet devices are created on the host system which ultimately function as the endpoints of network interfaces in the guests. The guest sees its endpoints as standard Ethernet devices, and bridging is used on the host to allow all guests to appear as individual servers.

PCI passthrough [17] is a technique that provides an isolation of devices to a given guest operating system so the device can be used exclusively by that guest, which eventually achieves near-native performance. Thus, this approach benefits network-bounded applications (e.g., HPC applications) that have not adopted virtualization because of contention and performance degradation through the hypervisor (to a driver in the hypervisor or through the hypervisor to a user space emulation). However, assigning devices to specific guests is also useful when those devices cannot be shared. For example, if a system included multiple video adapters, those adapters could be passed through to unique guest domains.

Both Intel and AMD provide support for PCI passthrough in their more recent processor architectures (in addition to new instructions that assist the hypervisor). Intel calls its option Virtualization Technology for Directed I/O (VT-d [2]), while AMD refers to I/O Memory Management Unit (IOMMU [3]). For each case, the new CPUs provide the means to map PCI physical addresses to guest virtual addresses. When this mapping occurs, the hardware takes care of access (and protection), and the guest operating system can use the device as if it were a non-virtualized system. In addition to this mapping of virtual guest addresses to physical memory, isolation is provided in such a way that other guests (or the hypervisor) are precluded from accessing it.

Xen supports PCI passthrough [38] for PV or HVM guests, but dom0 operating system must support it, typically available as a kernel build-time option. For PV guests, Xen does not require any special hardware support, but PV domU kernel must support the Xen PCI frontend driver for PCI passthrough in order to work. Hiding the devices from the dom0 VM is also required, which can be done with Xen using pciback driver. For HVM guests, hardware support (Intel VT-d or AMD IOMMU) is required as well as pciback driver on dom0 kernel. However, domU kernel does not need any special feature, so

PCI passthrough with proprietary operating systems is also possible with Xen using HVM guests.

4 Messaging Middleware for High Performance Cloud Computing

Two widely extended HPC messaging middleware, OpenMPI [12] and MPICH2 [26], were selected for the performance evaluation of native codes (C/C++ and Fortran) carried out on Amazon EC2. In addition, FastMPJ [32] was also selected as representative Java messaging middleware.

OpenMPI is an open source MPI-2 implementation developed and maintained by a consortium of academic, research, and industry partners. Open MPI's Modular Component Architecture (MCA) allows for developers to implement extensions and features within self-contained components. The Byte Transfer Layer (BTL) framework is designed to provide a consistent interface to different networks for basic data movement primitives between peers. This favor the quick and efficient support of emerging network technologies. Therefore, adding native support for a new network interconnect is straightforward, thus OpenMPI now includes several BTL implementations for TCP, Myrinet, InfiniBand and shared memory support, among other (see Figure 1).

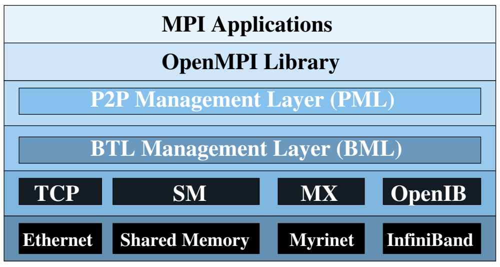

Fig. 1 OpenMPI software layers

MPICH2 is a high-performance and open source implementation of the MPI standard (both MPI-1 and MPI-2). Its goal is to provide an MPI implementation that efficiently supports different computation and communication platforms including commodity clusters, high-speed networks, and proprietary high-end computing systems. The ADI-3 (Abstract Device Interface) layer is a full featured low-level interface used in the MPICH2 implementation to provide a portability layer that allows access to many features of a wide range of communication systems. It is responsible for both the point to point and one sided communications. The ADI-3 layer can be implemented on top of the CH3 device, which only requires the implementation of a dozen functions but provides many of the performance advantages of the full ADI-3 interface. In order to support a new platform in MPICH2, only the CH3 channel has to be implemented. Several CH3 channels already offer support for TCP, Myrinet, Infiniband and shared memory (see Figure 2).

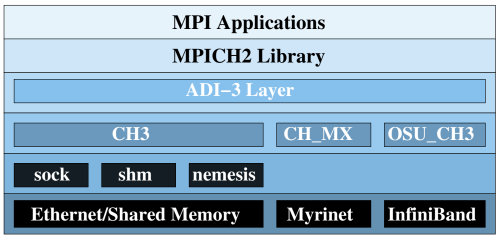

Fig. 2 MPICH2 software layers

Nemesis [9] is a new generic communication subsystem designed and implemented to be scalable and efficient both in the intranode communication context using shared-memory and in the internode communication case using high-performance networks. Nemesis has been integrated in MPICH2 as a CH3 channel and delivers better performance than other dedicated communication channels.

FastMPJ is a Java message-passing implementation which provides shared memory and high-speed networks support on InfiniBand and Myrinet. FastMPJ implements the mpiJava 1.2 API [10], the most widely extended MPJ API, and includes a scalable MPJ collectives library [30]. Figure 3 presents an overview of the FastMPJ communication devices on shared memory and high-speed cluster networks. From top to bottom, the communication support of MPJ applications with FastMPJ is implemented in the device layer. Current FastMPJ communication devices are implemented on JVM threads (`smpdev`, a thread-based device), on sockets over the TCP/IP stack (`iodev` on Java IO sockets and `niodev` on Java NIO sockets), on Myrinet and InfiniBand.

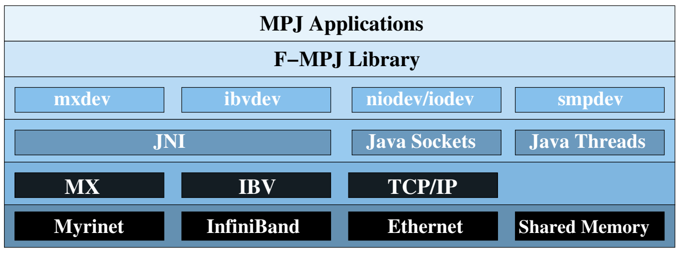

Fig. 3 FastMPJ communications support overview

5 Performance Evaluation

This section presents a performance evaluation of native (C/C++ and Fortran) and Java message-passing middleware for HPC on a cloud computing infrastructure, Amazon EC2 cluster compute instances, whose access to the high-speed network, 10 Gigabit Ethernet, is virtualized. In order to analyze the impact of the cloud network overhead in representative HPC codes, a testbed with similar hardware has been set up. This evaluation consists of a micro-benchmarking of point-to-point data transfers, both inter-VM (through 10 Gigabit Ethernet) and intra-VM (shared memory), at the message-passing library level and its underlying layer, TCP/IP. Then, the significant impact of virtualized communication overhead on the scalability of representative parallel codes, NAS Parallel Benchmarks (NPB) kernels [6], has been assessed. These results indicate that more efficient communication middleware support is required to get over current cloud network limitations.

5.1 Experimental Configuration

The evaluation has been conducted on sixteen cluster compute instances of the Amazon EC2 cloud [21], which have been allocated simultaneously in order to obtain nearby instances, and two nodes from our private cloud infrastructure (CAG testbed). Performance results using up to 8 processes have been obtained in a single node, whereas *processes*/8 nodes have been used in the remaining scenarios.

The Amazon EC2 cluster compute instances are a resource introduced in July 2010 with 23 GBytes of memory and 33.5 EC2 Compute Units (according to Amazon one EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor), running a Linux OS. For these instances, Xen HVM guests, the provider details the specific processor architecture (see Table 1), two Intel Xeon X5570 (2.93 GHz) quad-core Nehalem processors, hence 8 cores per instance, to allow the performance tunning of applications on this processor. These systems are interconnected via 10 Gigabit Ethernet, which is the differential characteristic of this resource. In fact, this EC2 instance type has been specifically designed for HPC applications and other demanding latency-bound applications. However, unfortunately the network interface card is not available via PCI passthrough in these instances, so its access is virtualized. The JVM used is OpenJDK 1.6.0_20 (Amazon Linux 2.6.35 Cluster virtual machine). There is no control on the location of the requested resources as the provider does not support yet the allocation of several instances connected to the same physical switch.

Regarding CAG testbed (see Table 1 again), each node has two Intel Xeon E5620 (2.40 GHz) quad-core processors (thus, 8 cores per node) and 16 GBytes of memory, interconnected via 10 Gigabit Ethernet (Intel PRO/10GbE NIC). These machines run Xen PV guests whose OS is Linux Debian with kernel 2.6.35, and the JVM is Sun JDK 1.6.0_20. These VMs access directly the

Table 1 Description of the specific hardware details of the two clouds used

	Amazon EC2	**CAG**
CPU	2 × Intel Xeon X5570 Nehalem @2.93 GHz	2 × Intel Xeon E5620 Westmere @2.40 GHz
#Cores	8 (16 with HT)	8 (16 with HT)
Memory	23 GB DDR3-1333 MHz	16 GB DDR3-1066 MHz
Memory Bandwidth	32 GB/s	25.6 GB/s
#Memory Channels	3	3
QPI Speed	6.4 GT/s	5.86 GT/s
#QPI Links	2	2
L3 Cache size	8 MBytes	12 MBytes
Interconnect	10 Gigabit Ethernet	10 Gigabit Ethernet
Virtualization	Xen HVM	Xen PV

network interface card through PCI passthrough, using this approach in 10 Gigabit Ethernet for the first time to the best of our knowledge. The motivation behind enabling PCI passthrough in this cloud is to analyze its impact on the efficient support of high-speed networks in virtualized environments.

The evaluated message-passing libraries are OpenMPI 1.4.3 and MPICH2 1.4, used with GNU C/Fortran compiler 4.1.2 and Intel C/Fortran compiler 12.1 (both compilers with -O3 flag), as well as FastMPJ 0.1 (labeled F-MPJ in graphs). The performance differences observed between GNU and Intel compilers were reduced, below 4%, when no network communications are involved, and completely negligible when network traffic is considered due to the virtualized network I/O overhead. As some works have already stated that GNU C compiler is generally the most efficient and reliable under Linux OS [18], only GNU compiler results are shown for clarity purposes. The point-to-point micro-benchmarking results have been obtained with Intel MPI Benchmarks suite (IMB, formerly Pallas) and its MPJ counterpart communicating byte arrays (hence, with no serialization overhead). The NPB implementations are the official NPB-MPI version 3.3 and the NPB-MPJ implementation [23]. The metric considered for the evaluation of the NPB kernels is MOPS (Millions of Operations Per Second), which measures the operations performed in the benchmark, that differ from the CPU operations issued. Moreover, NPB Class B workloads have been selected as they are the largest workloads that can be executed in a single Xen PV VM in CAG testbed, due to a Xen bug that limits the amount of memory to 3 GBytes when using PCI passthrough.

Finally, the performance results presented in this paper are the mean of several measurements, generally 1000 iterations in ping-pong benchmarks and 5 measurements for NPB kernels. The results show some variability due to the scheduling of the processes/threads on different cores within a node (the pinning of threads to specific cores has not been considered in this work).

5.2 Point-to-point Micro-benchmarking

Figures 4 and 5 show point-to-point latencies (for short messages) and bandwidths (for long messages) of message-passing transfers using the evaluated message-passing middleware on 10 Gigabit Ethernet and shared memory, respectively. Here, the results shown are the half of the round-trip time of a pingpong test or its corresponding bandwidth. Each figure presents the performance results of Amazon EC2 HVM and CAG native and CAG PV testbeds.

MPI (MPICH2 and OpenMPI) obtains 55-60 μs start-up latency and up to 3.7 Gbps bandwidth for Xen HVM point-to-point communication on Amazon EC2 over 10 Gigabit Ethernet (top graph in Figure 4). Here both MPI libraries rely on TCP sockets, which according to the raw TCP communication test from the Hpcbench suite [13], show poor start-up latency, around 52 μs, similar to MPI start-up latency. However, TCP sockets obtain higher bandwidth than MPI, up to 5.5 Gbps. These results, both MPI and Hpcbench, are quite poor, caused by the overhead in the virtualized access of Xen HVM to the NIC. However, the communication protocol also presents a significant influence as MPI is not able to achieve as much bandwidth as Hpcbench. This assessment is confirmed by the Java results, which show even poorer performance than MPI, with about 140 μs start-up latency and below 2 Gbps bandwidth for FastMPJ using niodev. Here Java sockets operation suffers a significant performance penalty.

The communication overhead caused by the virtualized access of Xen HVM to the NIC can be significantly alleviated through the use of Xen PCI passthrough (middle graph in Figure 4). Thus, MPI and Hpcbench achieve start-up latencies around 28-35 μs and bandwidths up to 7.2 Gbps. It is noticeable that Hpcbench and OpenMPI obtain quite similar results on this scenario, which suggests that the overhead incurred by network virtualization on Xen HVM scenario without PCI passthrough limits long message performance. Java performance on Xen PV also outperforms its results on Xen HVM, although the performance is still far from the MPI results.

In order to assess the impact of Xen PV virtualization overhead on the previous results, the performance of the communications has been measured on the CAG testbed running a non virtualized environment, thus obtaining the native performance of the system (bottom graph in Figure 4). The main conclusions that can be derived from these results are that Xen PV incurs an overhead of around 11 μs in start-up latency (25 μs overhead in the case of FastMPJ), whereas native long message performance achieves up to 8.2 Gbps for Hpcbench and almost 8 Gbps for MPI, which are reduced in Xen PV down to 5.3 Gbps for MPICH2 and 7.2 Gbps for Hpcbench. These results are still a little far from the theoretical performance of 10 Gigabit Ethernet, which suggests that TCP processing is the main performance bottleneck, for both short and long message performance. Regarding Java results on the non virtualized scenario, long message performance is similar to Xen PV results, only showing a small 10% improvement due to the high overhead of the operation of its NIO sockets implementation.

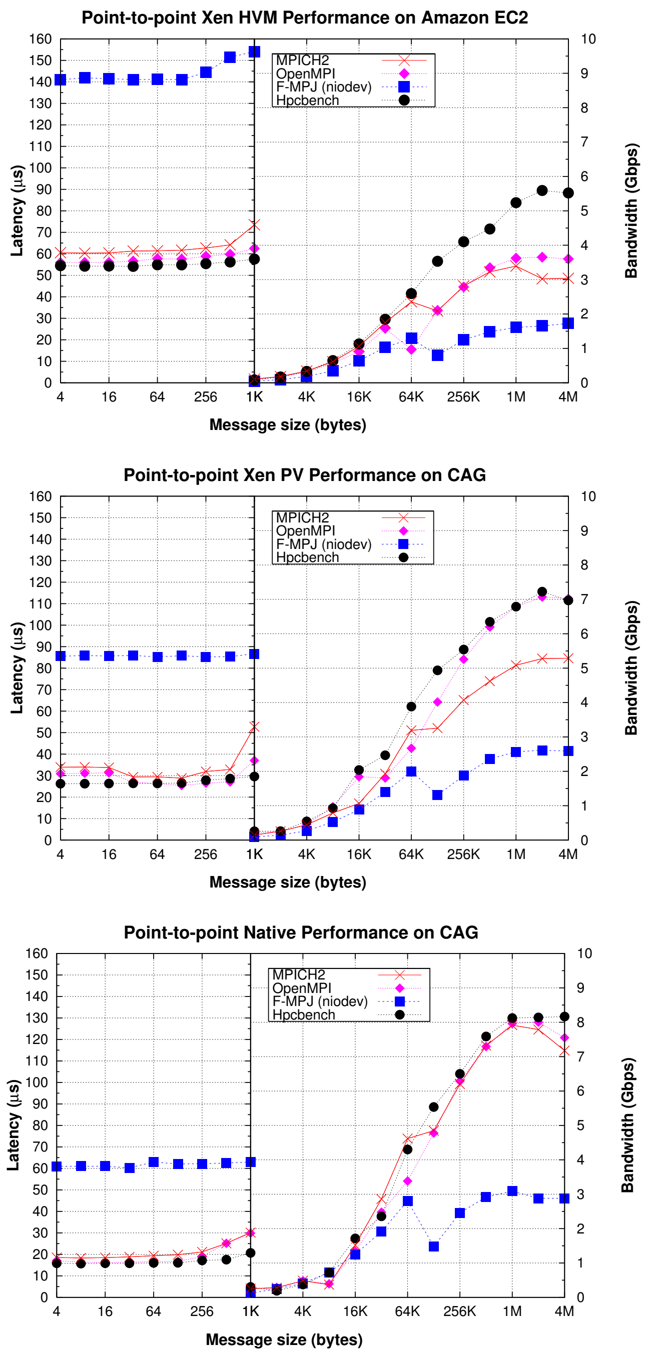

Fig. 4 Point-to-point communication performance on the analyzed testbeds over 10 Gigabit Ethernet

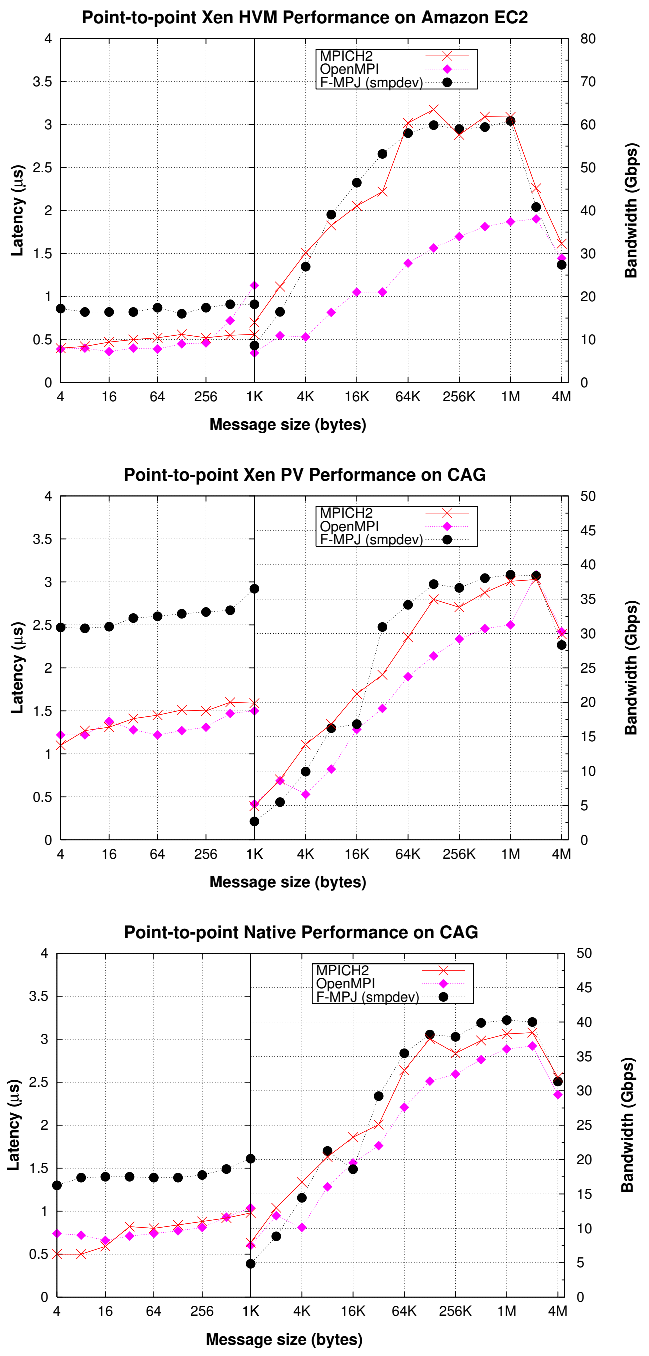

Fig. 5 Point-to-point shared memory communication performance on the analyzed testbeds

As the access to multi-core systems is a popular option in cloud computing, in fact each Amazon EC2 cluster computing instance provides two quad-core processors, the performance of communications on such shared memory systems has been also considered. Here communications are done generally within a single VM, without accessing the network hardware or their communication support (i.e., the communication protocol is not assisted by the NIC). In fact, the use of several VMs per compute node is inefficient, especially in terms of start-up latency, as the data transfers must pass through the hypervisor and the domain0 VM.

The shared memory performance results of message-passing middleware on Amazon EC2 compute cluster instances (top graph in Figure 5) are significantly superior than on 10 Gigabit Ethernet. Thus, MPI shows start-up latencies below 0.5 μs, thanks to the use of Nemesis in MPICH2 and the shared memory BTL in OpenMPI, whereas Java shows latencies below 1 μs. Regarding long message bandwidth, both FastMPJ(smpdev) and MPICH2, which relies on Nemesis, achieve up to 60 Gbps due to their efficient exploitation of multithreading, whereas OpenMPI gets up to 38 Gbps.

These high performance results confirms that Xen obtains close to native performance results for CPU and memory intensive operations, when no I/O activity is involved. In order to prove this statement the performance results of our CAG testbed with the Xen PV and native configurations (middle and bottom graphs in Figure 5) have been analyzed. These results show very low start-up latencies, below 1 μs for MPI and 1.5 μs for FastMPJ in the native scenario, which are slightly increased in approximately 0.5 μs for MPI and 1 μs for FastMPJ due to the Xen PV overhead. Moreover, the performance for long messages is similar for the three evaluated middleware, which suggests that the main memory subsystem is the main performance bottleneck, not the communication protocol. The drop in performance for large messages is due to the effect of cache size, as the storage needs exceed the L3 cache size (L3 cache size per processor is 8 MBytes in EC2 and 12 MBytes in CAG and is shared by all available cores for both cases). Thus, as the two processes involved in this micro-benchmark are scheduled in the same processor, the performance drops when the message size is equals or higher than a quarter of the L3 cache size due to the one-copy protocol implemented for large messages by these middleware. These results confirm the high efficiency of shared memory message-passing communication on virtualized cloud environments, especially on Amazon EC2 cluster compute instances, which is able to obtain two times better start-up latency and around 40% higher bandwidth than in CAG mainly thanks to its higher computational power and the higher performance of the memory.

5.3 Impact of Virtualized Networking on Applications Scalability

The impact of the virtualized network overhead on the scalability of HPC applications has been analyzed using the MPI and MPJ implementations of

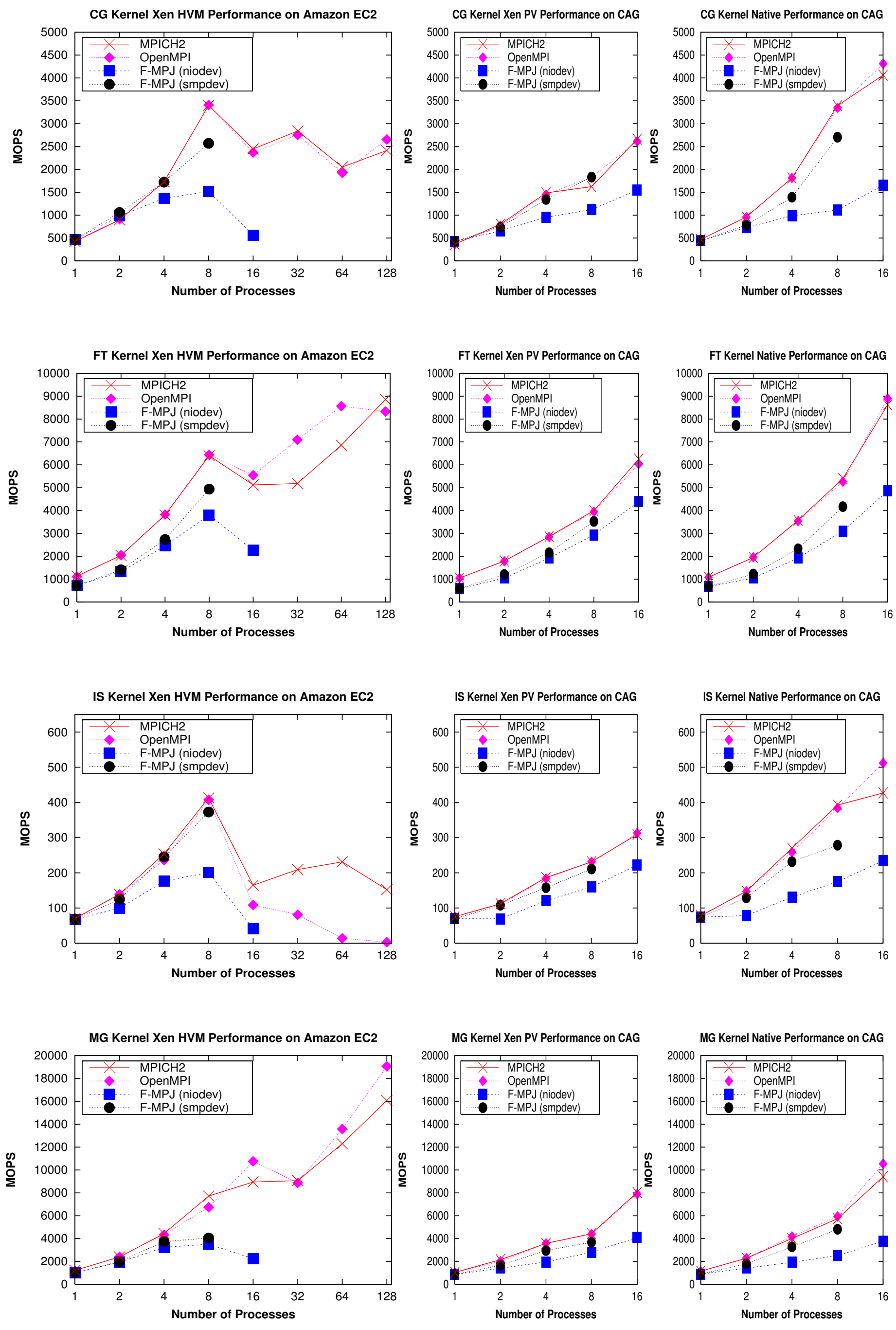

Fig. 6 NPB performance on Amazon EC2 HVM, CAG PV and CAG native testbeds

the NPB, selected as they are probably the benchmarks most commonly used in the evaluation of languages, libraries and middleware for HPC. In fact, there are additional implementations of the NPB for Grid middleware, Java Threads, OpenMP and hybrid MPI/OpenMP.

Four representative NPB codes have been evaluated: CG (Conjugate Gradient), FT (Fourier Transform), IS (Integer Sort) and MG (Multi-Grid). Moreover, Class B workloads have been selected due to the technical limitation of Xen PV as well as they present limited scalability, with performance results highly influenced by the efficiency of the communication middleware, which favor the comparison of the evaluated middleware.

Figure 6 presents the performance, in terms of Millions of Operations Per Second (MOPS), of CG, FT, IS and MG using up to 128 processes on Amazon EC2 and using up to 16 processes on CAG. Regarding CG kernel, which is a communications-intensive code that only includes point-to-point communication primitives, the evaluated middleware is not able to scale when using more than one node on Amazon EC2, due to their poor communications performance on 10 Gigabit Ethernet as it has shown a very high start-up latency. This statement has been proved analyzing CG results on CAG, where MPI and FastMPJ take advantage of the use of 16 processes (2 nodes), both for Xen PV and native scenarios. However, the higher performance of shared memory data transfers on Amazon EC2 allows their cluster compute instances to obtain the best performance results on 8 processes, whereas the CAG testbed shows a small performance penalty due to the higher start-up latency of Xen PV than the native scenario for shared memory communications. This behavior has also been appreciated in the remaining NPB kernels under evaluation. FastMPJ results on 32 and further number of processes are not shown for CG and the other kernels due to the poor performance they achieve.

Regarding FT results, Amazon EC2 suffers significantly the network overhead due to the extensive use of Alltoall primitives, showing similar results for one and four/eight nodes. Only using 16 nodes this kernel clearly outperforms the results obtained in a single node. Nevertheless, native CAG testbed using 16 processes (2 nodes) outperforms Amazon EC2 results on 128 processes (16 nodes) due to its higher network performance that allows this kernel to scale. IS kernel is a communications-intensive code whose scalability is highly dependent on Allreduce and point-to-point communication latency. Thus, native CAG is able to outperform Amazon EC2, which is able to scale only within a single node using up to the all available processes (8) thanks to its good shared memory performance and the avoidance of network traffic. Finally, MG is a less communication intensive kernel that is able to scale on Amazon EC2. Nevertheless, CAG results are quite competitive, achieving around 10000 MOPS with only 2 nodes (16 processes). Regarding FastMPJ results, the shared memory device (smpdev) generally shows the best performance although it is limited to shared memory systems.

The performance evaluation presented in this section has shown that communication bound applications would greatly benefit from the direct access to the NIC in virtualized environments. This is especially true for applications sensitive to network start-up latency, that therefore can take advantage from the flexibility, elasticity, and economy of cloud services provided that an efficient communication middleware for virtualized cloud environments would be made available.

6 Conclusions

The scalability of HPC applications on cloud infrastructures relies heavily on the performance of communications, which depends both on the network fabric and its efficient support in cloud middleware. To solve the current latency and network limitations in cloud services their providers are deploying high-speed networks (10 Gigabit Ethernet and InfiniBand), although without the proper middleware support as they rely on TCP/IP stack and a virtualized access to the NIC.

This paper has presented an evaluation of HPC message-passing middleware on a cloud computing infrastructure, Amazon EC2 cluster compute instances, equipped with 10 Gigabit Ethernet. The analysis of the performance results obtained, confronted with the experimental results measured in a similar testbed, a private cloud infrastructure, has shown the significant impact that virtualized environments still have on communications performance. This fact demands more efficient communication middleware support to get over current cloud network limitations, such as TCP/IP stack replacement on high-speed Ethernet networks.

The analysis of the measured performance results has shown significant scalability increases when supporting the direct access to the underlying NIC through PCI passthrough, reducing the communication processing overhead and the associated data copies. In fact, thanks to this technique, our experimental two-node cloud testbed is able to outperform the results obtained on sixteen Amazon EC2 compute cluster instances.

Acknowledgements This work was funded by the Ministry of Science and Innovation of Spain under Project TIN2010-16735 and an FPU Grant AP2010-4348 and by an Amazon Web Services (AWS) LLC research grant.

References

1. A Message Passing Interface Standard (MPI) http://www.mcs.anl.gov/research/projects/mpi/ [Accessed July 2012]
2. Abramson D et al (2006) Intel Virtualization Technology for Directed I/O. Intel Technology Journal 10(3):179–192
3. Advanced Micro Devices (AMD) I/O Virtualization Technology (IOMMU). http://support.amd.com/us/Processor_TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf, February 2009 [Accessed July 2012]

4. Advanced Micro Devices (AMD) Virtualization Technology (AMD-V). http://sites.amd.com/us/business/it-solutions/virtualization/Pages/amd-v.aspx [Accessed July 2012]
5. Baek SJ, Park SM, Yang SH, Song EH, Jeong YS (2010) Efficient Server Virtualization Using Grid Service Infrastructure. Journal of Information Processing Systems 6(4):553–562
6. Bailey DH et al (1991) The NAS Parallel Benchmarks. Int Journal of High Performance Computing Applications 5(3):63–73
7. Baker M, Carpenter B (2000) MPJ: A Proposed Java Message Passing API and Environment for High Performance Computing. In: Proc. 15th IPDPS Workshops on Parallel and Distributed Processing (IPDPS'00), Cancun, Mexico, LNCS, vol 1800, pp 552–55
8. Barham P, Dragovic B, Fraser K, Hand S, Harris TL, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the Art of Virtualization. In: Proc. 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing (Lake George), NY, USA, pp 164–177
9. Buntinas D, Mercier G, Gropp W (2006) Design and Evaluation of Nemesis, a Scalable, Low-Latency, Message-Passing Communication Subsystem. In: Proc. 6th IEEE Intl. Symposium on Cluster Computing and the Grid (CCGRID'06), Singapore, pp 521–530
10. Carpenter B, Fox G, Ko S, Lim S (2002) mpiJava 1.2: API Specification. http://www.hpjava.org/reports/mpiJava-spec/mpiJava-spec/mpiJava-spec.html [Accessed July 2012]
11. Fraser K, Hand S, Neugebauer R, Pratt I, Warfield A, Williamson M (2004) Safe Hardware Access with the Xen Virtual Machine Monitor. In: Proc. 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS'04), Boston, MA, USA
12. Gabriel E et al (2004) Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In: Proc. 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, pp 97–104
13. Huang B, Bauer M, Katchabaw M (2005) Hpcbench - A Linux-Based Network Benchmark for High Performance Networks. In: Proc. 19th Intl. Symposium on High Performance Computing Systems and Applications (HPCS'05), Guelph, Ontario, Canada, pp 65–71
14. Huang W, Liu J, Abali B, Panda DK (2006) A Case for High Performance Computing with Virtual Machines. In: Proc. 20th Intl. Conference on Supercomputing (ICS'06), Cairns, Queensland, Australia, pp 125–134
15. Huang W, Koop MJ, Gao Q, Panda DK (2007) Virtual Machine Aware Communication Libraries for High Performance Computing. In: Proc. ACM/IEEE Conference on Supercomputing, Reno, NV, USA, pp 1–12
16. Intel Corporation (2006) Virtualization Technology (Intel VT). http://www.intel.com/technology/virtualization/technology.htm?iid=tech_vt+tech [Accessed July 2012]
17. Jones T (2009) Linux virtualization and PCI passthrough. http://www.ibm.com/developerworks/linux/library/l-pci-passthrough/ [Accessed July 2012]

18. Karna AK, Zou H (2010) Cross Comparison on C Compilers Reliability Impact. Journal of Convergence 1(1):65–74
19. Liu J, Huang W, Abali B, Panda DK (2006) High Performance VMM-bypass I/O in Virtual Machines. In: Proc. USENIX'06 Annual Technical Conference, Boston, MA, USA, pp 29–42
20. Liu ST, Chen YM (2011) Retrospective Detection of Malware Attacks by Cloud Computing. Int Journal of Information Technology, Communications and Convergence 1(3):280–296
21. Amazon Web Services LLC (AWS LLC) Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2 [Accessed July 2012]
22. Amazon Web Services LLC (AWS LLC) High Performance Computing Using Amazon EC2. http://aws.amazon.com/ec2/hpc-applications/ [Accessed July 2012]
23. Mallón DA, Taboada GL, Touriño J, Doallo R (2009) NPB-MPJ: NAS Parallel Benchmarks Implementation for Message-Passing in Java. In: Proc. 17th Euromicro Intl. Conf. on Parallel, Distributed, and Network-Based Processing (PDP'09), Weimar, Germany, pp 181–190
24. Mansley K, Law G, Riddoch D, Barzini G, Turton N, Pope S (2007) Getting 10 Gb/s from Xen: Safe and Fast Device Access from Unprivileged Domains. In: Proc. Workshop on Virtualization/Xen in High-Performance Cluster and Grid Computing (VHPC'07), Rennes, France, pp 224–233
25. Mateescu G, Gentzsch W, Ribbens CJ (2011) Hybrid Computing-Where HPC Meets Grid and Cloud Computing. Future Generation Computer Systems 27(5):440–453
26. MPICH2 (2005) High-performance and Widely Portable MPI. http://www.mcs.anl.gov/research/projects/mpich2/ [Accessed July 2012]
27. Nanos A, Koziris N (2009) MyriXen: Message Passing in Xen Virtual Machines over Myrinet and Ethernet. In: Proc. 4th Workshop on Virtualization in High-Performance Cloud Computing (VHPC'09), Delft, The Netherlands, pp 395–403
28. Raj H, Schwan K (2007) High Performance and Scalable I/O Virtualization via Self-Virtualized Devices. In: Proc. 16th Intl. Symp. on High Performance Distributed Computing, Monterey, CA, USA, pp 179–188
29. Regola N, Ducom JC (2010) Recommendations for Virtualization Technologies in High Performance Computing. In: Proc. 2nd Intl. Conference on Cloud Computing Technology and Science (CloudCom'10), Indianapolis, IN, USA, pp 409–416
30. Taboada GL, Ramos S, Touriño J, Doallo R (2011) Design of Efficient Java Message-Passing Collectives on Multi-Core Clusters. Journal of Supercomputing 55(2):126–154
31. Taboada GL, Ramos S, Expósito RR, Touriño J, Doallo R (2012) Java in the High Performance Computing arena: Research, practice and experience. Science of Computer Programming (In press, http://dx.doi.org/10.1016/j.scico.2011.06.002)

32. Taboada GL, Touriño J, Doallo R (2012) F-MPJ: Scalable Java Message-Passing Communications on Parallel Systems. Journal of Supercomputing 60(1):117–140
33. Walker E (2008) Benchmarking Amazon EC2 for High-Performance Scientific Computing. LOGIN: The USENIX Magazine 33(5):18–23
34. Wang G, Ng TSE (2010) The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In: Proc. 29th Conference on Information Communications, San Diego, CA, USA, pp 1163–1171
35. Wang X, Sang Y, Liu Y, Luo Y (2011) Considerations on Security and Trust Measurement for Virtualized Enviroment. Journal of Convergence 2(2):19–24
36. Whitaker A, Shaw M, Gribble SD (2002) Denali: Lightweight Virtual Machines for Distributed and Networked Applications. In: Technical Report 02-02-01, University of Washington, USA
37. Won C, Lee B, Park K, Kim MJ (2008) Eager Data Transfer Mechanism for Reducing Communication Latency in User-Level Network Protocols. Journal of Information Processing Systems 4(4):133–144
38. Xen Org (2005) Xen PCI Passthrough. http://wiki.xensource.com/xenwiki/XenPCIpassthrough [Accessed July 2012]
39. Ye Y, Li X, Wu B, Li Y (2011) A Comparative Study of Feature Weighting Methods for Document Co-Clustering. Int Journal of Information Technology, Communications and Convergence 1(2):206–220