Virtual InfiniBand Clusters for HPC Clouds

Marius Hillenbrand
System Architecture Group
KIT, Germany
m.hillen@kit.edu

Viktor Mauch
Steinbuch Centre for Computing
KIT, Germany
mauch@kit.edu

Jan Stoess
System Architecture Group
KIT, Germany
stoess@kit.edu

Konrad Miller
System Architecture Group
KIT, Germany
miller@kit.edu

Frank Bellosa
System Architecture Group
KIT, Germany
bellosa@kit.edu

ABSTRACT

High Performance Computing (HPC) employs fast interconnect technologies to provide low communication and synchronization latencies for tightly coupled parallel compute jobs. Contemporary HPC clusters have a fixed capacity and static runtime environments; they cannot elastically adapt to dynamic workloads, and provide a limited selection of applications, libraries, and system software. In contrast, a cloud model for HPC clusters promises more flexibility, as it provides elastic virtual clusters to be available on-demand. This is not possible with physically owned clusters.

In this paper, we present an approach that makes it possible to use InfiniBand clusters for HPC cloud computing. We propose a performance-driven design of an HPC IaaS layer for InfiniBand, which provides throughput and latency-aware virtualization of nodes, networks, and network topologies, as well as an approach to an HPC-aware, multi-tenant cloud management system for elastic virtualized HPC compute clusters.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network Management—*InfiniBand, Isolation*; D.4.4 [**Operating Systems**]: Communications Management—*InfiniBand Virtualization*

General Terms

Design

Keywords

HPC, InfiniBand, Cluster, Cloud Computing, Virtualization

1. INTRODUCTION

Today's High Performance Computing (HPC) clusters are typically operated and used by single organizations. A cluster operator deploys a predefined, fixed runtime and systems environment in a known-good configuration. Such physically owned clusters have two major drawbacks: First, the resource demands of HPC workloads are often fluctuating, leaving the cluster under-utilized or overloaded. Second, application developers are faced with a restricted runtime environment, which allows them at best to change application libraries, but not the underlying operating system or core runtime libraries, such as the job scheduler.

HPC in the Cloud promises increased flexibility and efficiency in terms of cost and energy consumption. Large providers of HPC Infrastructure as a Service (IaaS) can reduce personnel costs through high degrees of automation and a better overall utilization than in privately operated clusters. For end users, elastic virtual clusters provide precisely the capacity that suits demand, without the need to purchase and operate own hard- and software. Since virtual resources can be fully granted to users, they can choose or customize the OS or runtime environment according to their demands.

As an example, consider a team of researchers using fluid dynamics simulations with varying demand for compute capacity: They employ coarse test runs at the beginning of their project and highly detailed simulations when completing their publication (continuous high demand). With an HPC cloud, they could scale the capacity of their virtual cluster according to the stage of their project. During periods of little or no use, there would be little or no running expenses. *Pay-per-use* allows to associate the costs directly with the resource usage (and the project's budget). The researchers use an experimental simulation package with special libraries and therefore customized the software on their virtual nodes.

The Challenge of HPC-as-a-Service

HPC applications differ from standard cloud workloads in their much higher demands on the underlying resources and their guaranteed and timely delivery. Those requirements, however, are particularly hard to achieve in a virtualized environment with its high I/O-overhead and jitter. It is unclear, whether the elasticity and standardization of cloud environments, which are both achieved by means of virtualization, can be achieved along with the predictability associated with HPC. Further, and in contrast to server workloads, HPC jobs are typically CPU-intensive and task synchronization points (e.g., in the MPI communication library) often require equal-paced CPU resources on all available cores. Therefore,

it is an open question whether the consolidation benefits of the cloud model can really be achieved in HPC environments.

Former research on HPC applications in contemporary IaaS environments has identified network quality of service (QoS) as the primary hindrance for virtual HPC clusters [6, 10], which can be relieved by providing virtual machines (VM) with direct access to an high-speed cluster interconnect [22]. Despite those initial findings, there had been no answer to the question how cloud computing can achieve high levels of isolation and elasticity, while, at the same time, delivering guaranteed performance to virtualized HPC applications.

In this paper, we present a comprehensive architecture for an HPC cloud model. Our architecture follows three design goals: (i) isolation of individual users in the cluster network to achieve **multi-tenancy**; (ii) automatic allocation and deployment of virtual HPC clusters, including the configuration of the cluster network, to achieve **dynamic provisioning**; and (iii) guaranteed network performance at all times, to enable HPC-class **service level agreements**. Our approach focuses on InfiniBand (IB), a common cluster interconnect [19]. We present a performance-driven design of an HPC IaaS layer for IB, which provides throughput and latency-aware virtualization of nodes, networks, and network topologies. It includes an HPC-aware, multi-tenant cloud management system for elastic virtualized HPC clusters. Our approach enables privately operated HPC clusters to be run more cost-efficiently in the cloud, due to new sharing opportunities [3].

The rest of the paper is structured as follows: we first provide background information and give an overview over related work in Section 2. We then discuss our approach for an HPC cloud in Section 3. We present early experiences in Section 4. Finally, we conclude and point out ongoing work towards a prototype in Section 5.

2. BACKGROUND AND RELATED WORK

Our design is based on several existing building blocks which we introduce in this section. Further, we present related work, such as concepts for virtualized clusters.

Cloud computing is an ongoing trend in the IT industry, referring to a model for ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources such as servers, storage, applications, and services [17]. Users move away from owning and operating applications and hardware themselves. Instead, they start utilizing services and computing infrastructure that are offered by cloud providers via web services. Payment is made according to consumption [2]. Compared to user-owned infrastructure, cloud computing offers more flexibility and scalability. Growing demand can be answered by adding resources from the cloud almost instantly; previously you had to purchase and configure new hardware. Cloud computing heavily relies on virtualization, with the main goal being the abstraction of physical resources, while still retaining their *interfaces*. Smith and Nair present an overview of system-level virtualization in [24].

2.1 Cloud Computing and High-Performance Computing

Virtualization offers several benefits for HPC, which have been pointed out already by previous research [8, 11, 18]. Some of the promised benefits are:

Customization: Users can configure the virtual clusters according to their own needs.

Isolation: Operators can grant (virtual) administrator privileges to users, while retaining full ownership of the physical nodes and networks.

Reliability: Virtual machines can be monitored without modification of the observed VM. Reliability can be improved with checkpoint/restart schemes for VMs.

Scalability: Users can debug and test run their algorithms on virtual nodes.

Virtualized clusters have already been proposed for grid computing [9] and several frameworks have been developed to manage such clusters (e.g., Nimbus [15]). One essential feature, called *contextualization*, allows to configure the software inside virtual clusters in an automated way and thus simplifies cluster management. In contrast, contemporary commercial IaaS clouds such as Amazon EC2 do not provide support for clustering VMs; each VM is a single entity [4]. As an alternative, there are tools such as StarHPC [13], which provide sophisticated mechanisms for cluster setup on top of these offerings. However, information about the cluster is scattered between cloud provider and StarHPC in this case.

Gupta and Milojicic have evaluated contemporary IaaS clouds [10]; they conclude that they are not cost-effective for communication-intensive applications, but suitable for loosely-coupled applications with little communication. The Amazon EC2 *cluster compute instances*[1] are a popular IaaS offer targeted at HPC applications. In contrast to regular EC2 instances, cluster compute VMs are assigned to dedicated nodes with 10GBit/s Ethernet. Multiple instances can be placed close together to improve network performance within a cluster. Several studies confirm a performance comparable to that of HPC clusters [3,4]. However, you still need external tools (such as StarHPC) to setup virtual clusters on top of EC2 *cluster compute instances*.

Our approach goes beyond existing research on both cloud systems and virtualized HPC. We propose a cloud management framework that specifically supports virtual HPC clusters: It provides high-performance resources and InfiniBand interconnectivity for virtualized clusters, supports dynamic reconfiguration of nodes and network topologies for cloud-like elasticity, and provides *contextualization* for automated deployment of HPC workloads.

2.2 The InfiniBand Architecture

InfiniBand (IB) is a high-performance network technology which is in wide-spread use in compute clusters. We provide a short overview of the IB architecture [12] and introduce the mechanisms that we employ. Compared to network technologies such as Ethernet, IB has a substantial performance advantage through aggressive protocol offloading; all layers up to the transport layer are handled completely in network adapters. Moreover, IB directly interfaces applications to the network hardware. The OS is involved only in establishing connections and registering memory buffers to ensure protection. Applications bypass the OS to trigger actual communication operations and poll for their completion, by directly accessing device memory. As a result, an application can handle complete send/receive cycles independently and without latency from the intervention of the OS. Send requests and receive buffers are posted to queues (*pairs* of send

[1] `http://aws.amazon.com/hpc-applications/`

and receive queues). These *queue pairs* form communication endpoints. IB supports four transport types: connection- and datagram-oriented, each in a reliable and an unreliable variant. In addition to send and receive primitives, IB supports *remote direct memory access (RDMA)* operations.

An IB network comprising only end nodes and switches is called a *subnet* and forms an administrative domain. Two types of node addresses are assigned dynamically by management software: *Local identifiers (LIDs)* are used by layer-2 switches, and *global identifiers (GIDs)* are used by layer-3 routers to forward packets between *subnets*.

2.3 InfiniBand Isolation and Quality of Service

IB has mechanisms for restricting communication and for traffic-shaping, enabling the enforcement of isolation and Quality-of-Service (QoS) policies. The isolation mechanisms form groups of nodes called *partitions* and restrict communication to within these partitions. Each node can be a member of one or multiple partitions, as stored in its *membership table* by management software. Switches can be configured to filter packets based on partitions and thereby enforce isolation. The QoS mechanisms allow to schedule network bandwidth in a flexible way by differentiating between up to 15 traffic classes, called *service levels*. In each port, out-bound packets are sorted into up to 15 send queues, called *virtual lanes*, based on their traffic class. The outgoing link is multiplexed based on a configurable weighted round-robin schedule.

Both isolation and QoS mechanisms work at the granularity of network ports. They are not sufficient to separate VMs running on the same host and thereby using the same network port. We take care of this in our approach in Section 3.2.

2.4 InfiniBand Virtualization

There are three ways to virtualize IB with near-native performance:

PCI Pass-through grants a VM direct access to a dedicated HCA. It requires an I/O Memory Mapping Unit (IOMMU) to ensure memory protection between different VMs [25]. A guest OS uses regular drivers.

Para-Virtualization for IB has been proposed by Liu et al. in [16]. It requires ongoing modifications of drivers in host and guest with respect to changes of the underlying hardware and OS.

Single Root-I/O Virtualization (SR-IOV) is a standard for virtualization support in hardware [20]. It allows a PCI Express device to appear as multiple virtual devices which guests can access via PCI Pass-through.

In all three cases, an application can circumvent all layers of system software, OS, and hypervisor. Communication operations thus suffer no virtualization overhead.

3. APPROACH TO HPC CLOUDS

Our goal is automated provisioning of elastic virtual compute clusters for HPC workloads as an IaaS cloud service. This section presents our approach towards this goal. We first introduce our overall architecture, before we discuss its three aspects in detail: virtualization of each node (Section 3.1), virtualization of the cluster interconnect (Section 3.2), and orchestration with a cloud management framework extended to provide elastic virtual HPC clusters (Section 3.3).

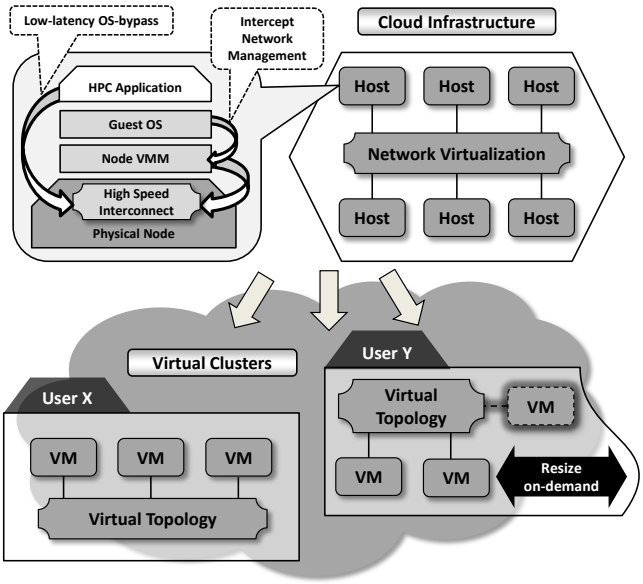

Figure 1: Virtualization of hosts and the cluster interconnect to provide elastic virtual clusters.

In contrast to physical clusters, our approach includes *elasticity*, the possibility to dynamically grow or shrink the size of a virtual cluster, to reflect the current demand for compute capacity. Each virtual cluster gets the impression of using the interconnect by itself, albeit with reduced bandwidth. A user can employ existing management tools to configure advanced IB features in his virtual cluster's view on the network. As an advantage over physical IB networks, a user can employ a network pre-configured by the HPC cloud provider, instead of implementing or hand-tuning his own configuration. We extend an existing cloud computing framework to manage both the virtual clusters and the IB interconnect accordingly. Figure 1 provides an overview of our architecture.

HPC applications have much stronger requirements on QoS than traditional server workloads. They typically match the bulk-synchronous single-program multiple-data (SPMD) application model [5, 14] which can be simplified to a cycle through two phases: Phases with purely local computation alternate with synchronization phases, which often require communication between all processes of a parallel job. When some processes use more time for the computation phase, all others have to wait before all processes can start the synchronization phase together. Such delays limit the scalability of parallel applications significantly [7, 21]. The inhomogeneous computation time in different processes is mainly caused by OS background activity, but parallel application performance will also suffer whenever one of the involved processes receives fewer computation resources. In addition, the time required for synchronization directly depends on communication latency in the cluster network. Thus, an HPC cloud has to guarantee resource allocation for processes (i.e., CPU and memory) and network QoS (mainly, a bounded delay) within the virtual cluster.

3.1 Node Virtualization

We provide a user with a collection of VMs that function as *virtual cluster nodes*. A physical node provides these VMs

with basic resources, such as CPU, memory, and disk storage, and with access to the IB cluster interconnect.

We want to minimize the background activity in the host system (OS and hypervisor) to reduce their negative influence on application performance. We dedicate a physical CPU core to each logical CPU of a virtual cluster node and we allocate physical memory for all virtual memory assigned to a VM. The alternatives, timesharing and memory overcommitment, would lead to preemption of HPC jobs, paging activity, and ultimately jitter in the execution speed of HPC jobs – thereby reducing parallel application performance [7, 21].

We use an existing hypervisor for commercial and server workloads, the kernel-based virtual machine (KVM) running on Linux. We made this decision for the following reasons:

Compatibility to Linux, POSIX: Compared with specialized lightweight kernels, a POSIX compatible OS such as Linux supports a variety of programming languages and libraries. Linux is the de facto standard OS for HPC clusters and supercomputers [19].

Availability of Source Code: In contrast to commercial offerings, such as VMware ESX, the KVM source code is available and allows modifications.

Convergence of host and guest OS: Optimizing Linux for HPC workloads will benefit both host and guest OS. Using different systems would double the effort.

Compatibility to Cloud Solutions: Many cloud management frameworks already support KVM, in contrast to specialized research OSs.

Compatibility to InfiniBand: The OpenFabrics Alliance, formed of IB hardware manufacturers and others, provides and supports IB drivers for several GNU/Linux distributions. Thus, there is no porting required for our selection of host and guest OS.

SR-IOV is our preferred solution for IB access due to the standardization of SR-IOV and announcements of IB SR-IOV support by several vendors such as QLogic and Mellanox. With SR-IOV, several VMs can use a single adapter. We employ existing (as of early 2012, still announced) drivers, whereas para-virtualization would require to adapt and maintain drivers per device and OS/hypervisor combination. We based our approach on early SR-IOV patches for Mellanox IB adapters that have been posted to the *linux-rdma* mailing list in June 2010 and in December 2011. We expect our approach to work with the released drivers as well.

3.2 Network Virtualization

We want to share a physical IB network between multiple virtual HPC clusters and, at the same time, isolate them from each other. We have to guarantee QoS properties such as minimum bandwidth and bounded latency. In addition, a user should be able to use all configuration options of an IB network inside his own virtual cluster. All mechanisms we employ to reach these goals shall not influence performance.

We redirect a user's management actions to a *virtual network view*, a state machine that resembles the physical network and reacts to regular IB management protocols. That way, a user can employ existing management tools to customize the following settings for his virtual cluster:

Packet routing: A user can fine-tune routing to match the communication pattern of his HPC application.

Isolation: A user may partition his virtual cluster into further sub-partitions.

QoS policy: A user may employ his own QoS policy within his bandwidth share.

However, a user is strictly blocked from modifying the configuration of the physical network. The configuration from a user's *virtual network view* is implemented in the physical network only after being approved and adapted by the cloud management. Thus, the cloud provider can always ensure and enforce proper isolation and bandwidth allocations. As an advantage over physical IB infrastructure, the HPC cloud provider supplies a reasonable default setup, so that users do not have to run management tools if they do not need a custom network configuration.

IB Network Management Interface

We strictly isolate the users from the physical network's management interfaces. We virtualize these interfaces to allow users to manage the parts of the network connecting their virtual clusters. We use the SR-IOV architecture of the Mellanox IB adapters: There, only the host OS can send valid *management datagrams (MAD)* to alter the configuration of network nodes, such as switches and adapters. A guest OS may only pass management traffic through the host via a channel provided by the SR-IOV drivers: We can easily replace this forwarding mechanism with a redirection to the state machine implementing our *virtual network view*.

In addition, we employ a second layer of protection: IB supports to protect the network configuration with a secret key. Once this mechanism is enabled, each network node (such as a switch or a network adapter) ignores any configuration requests without the correct key. The cloud management framework initially configures this protection, assigns each node a separate random key, and keeps all keys secret. As a result, only cloud management and the cloud provider can configure the physical network directly. Legitimate configuration changes requested by a user are redirected and transformed through cloud management (the *virtual network view* we discussed above), which applies them on behalf of the user.

If a user breaks out of his VM and gains full access to an IB adapter, he can circumvent the redirection of his management operations. However, the key-based protection in every node will keep him from actually modifying the network configuration. A brute-force attack is not practical, as it would take more than 3000 years (64-bit secret key, 40 Gbit/s link speed). Even if the user can extract or guess one protection key (e.g., from the adapter in the host of his VM), he cannot gain access to any other node's configuration, because all nodes are protected by different keys.

Network Traffic Isolation

We want to isolate virtual clusters and make communication between nodes located in different virtual clusters impossible. For this purpose, we extend the IB isolation mechanism (see Section 2.3) to work with VMs.

IB *partitions* work at the granularity of network ports (in adapters and switches). With several VMs sharing one adapter, this is insufficient: An adapter is a member of several partitions (typically one per VM), but we cannot allow every VM to use every assigned partition, as the basic mechanism would do. Partition filtering in switches does not help, as a switch can only differentiate between ports—that is, complete hosts—and not between individual VMs using

that port. So, we have to enforce correct partition usage in hosts, where individual VMs can be distinguished.

In the IB software interface, an application specifies the partition to use during connection establishment. With SR-IOV IB access, such operations must be passed through the host OS driver. Thus, the host can ensure that applications only use partitions assigned to their VM. After a connection is established, the application can send and receive using OS-bypass and there is no further virtualization overhead. However, it cannot alter the used partition. The unreliable datagram transport is different and cannot be supported in this way: The partition is specified per datagram directly via OS-bypass. Thus, we must emulate *queue pairs* for unreliable datagrams, disable OS-bypass, and thereby reduce performance. However, HPC applications (i.e., MPI) use reliable transports over IB and thus are unaffected.

With *partitioning* being effective for VMs, we can implement our isolation policy based on this mechanism: By default, each virtual cluster will be provided with a separate partition. When a user defines further isolation inside his virtual cluster, we assign him as many partitions in the physical network, as defined inside his *virtual network view*. Up to 32768 (physical) partitions can be defined per subnet, which should be sufficient for most cases. The host OS translates the partition in each connection request (if valid) to the partition used in the physical network, completely transparent to the guest OS and its applications.

Network Performance Isolation

HPC services require bandwidth and latency guarantees, especially when some users fully utilize their bandwidth shares. Alfaro et al. provide algorithms for implementing QoS policies using the mechanisms provided by IB in [1]. We consider each virtual cluster to be an *uncooperative user* that tries to utilize all available bandwidth. Thus, we assign each virtual cluster a distinct traffic class to strictly separate its traffic from others. Using this strict separation, we can employ the IB QoS mechanisms to assign minimum bandwidth shares and maximum latencies per hop. Based on the IB specification [12] and the work of Alfaro et al. [1], we can determine appropriate settings. We expect that actual service level guarantees can also be derived this way.

With our policy of strict traffic separation, we face a limitation: IB can only differentiate 15 traffic classes, so we can share a physical network link only between 15 virtual clusters. However, each virtual cluster will typically occupy just a fragment of the physical cluster, so each link will be used only by a subset of all virtual clusters. This subset depends on VM placement, the physical network topology, and the routing algorithm used. Consequently, clever VM placement that respects the network topology can support more than 15 virtual clusters with performance isolation: Virtual clusters with non-overlapping placement can be assigned the same traffic class.

The restriction to 15 traffic classes also limits how detailed a user's custom QoS policy can be applied in the network: We first ensure performance isolation between virtual clusters and implement custom QoS settings only when there are unassigned traffic classes left. Otherwise, a user's traffic is handled less differentiated than the user intended (e.g., as only one traffic class instead of several).

At the IB software interface, we employ the same approach as with network isolation: The host OS intercepts connection establishment and maps the traffic class from the user's view to that assigned in the physical network (specified by an identifier called *service level*). That way, a user is restricted to utilize the bandwidth share assigned to his virtual cluster. As with isolation, we must emulate unreliable datagrams and deny OS-bypass.

3.3 Cloud and Cluster Network Management

A cloud management framework orchestrates the mechanisms we described to provision virtual HPC clusters automatically. We are currently completing an appropriate framework. In this section, we present our goals and the design principles we follow.

A management framework for HPC clouds has several tasks: At a cluster-wide scale, the framework configures the cluster network to provide network isolation and QoS. It incorporates the network topology in the VM placement algorithms to allocate VMs that comprise a virtual cluster close to each other and to comply with restrictions imposed by the IB QoS mechanisms (see Section 3.2). Locally on each node, it grants VMs access to the cluster interconnect by assigning them an SR-IOV virtual device and by setting up network virtualization in the host OS (i.e., the *virtual network view* we introduced in Section 3.2).

We employ an existing cloud management framework that already supports virtual clusters (Nimbus [9, 15]) and add the functionality to manage the cluster interconnect. We integrate support for virtual clusters and the cluster network with cloud management, instead of adding it on top, for three reasons: (1) providing VMs with access to the cluster interconnect requires modifications to the hosts; (2) virtual clusters form the basis of our network isolation and QoS policies and thus have to be integrated with cluster network management; and (3) virtual clusters should be considered in VM placement, which is the task of cloud management.

4. EARLY EXPERIENCES

In this section, we present experiences with an early prototype. We use a 4-node cluster equipped with Mellanox ConnectX-2 IB adapters, and an IB DDR switch. Since SR-IOV IB drivers are not yet available, we revert to using PCI Pass-through to dedicate an adapter to a single VM. Based on a review of alpha SR-IOV patches, we expect to get a reasonable estimate for the performance with SR-IOV.

With regard to isolation between virtual clusters, we have verified the following properties of our design by experimentation: (1) We can restrict communication to within a virtual cluster, and (2) no user can modify the network configuration, even when user VMs have full access to an IB adapter.

We have compared communication latencies between Amazon EC2 cluster compute instances and VMs in our prototype using the SKaMPI [23] MPI microbenchmark. In EC2, data transfers have to pass through several layers of system software, such as the guest TCP/IP stack or the ethernet driver in Xen dom0. In contrast, an application can directly access the IB adapter in our HPC cloud prototype, thereby bypassing all layers of system software. As a result, we observed significantly lower latencies in our prototype ($3.4\,\mu s$ instead of $77.5\,\mu s$ for 4-byte messages). Early experiments with HPC application benchmarks show promising results. They indicate that we can improve the default setup of common Linux distributions significantly with moderate changes (e.g., reducing timer frequency).

5. CONCLUSION AND OUTLOOK

An HPC IaaS cloud model has many benefits: Elastic virtual clusters provide capacity on-demand and the pay-as-you-go principle avoids the huge initial investments of physically owned clusters. We present a novel architecture for such HPC clouds based on the IB cluster interconnect. We provide each user with the impression of a dedicated physical network. We provide strict isolation between virtual clusters in the physical network (for multi-tenancy) and incorporate a user's settings as far as QoS mechanisms permit.

We are currently working on completing a prototypic HPC cloud, which will incorporate SR-IOV IB access. We will publish a thorough evaluation as soon as SR-IOV drivers are publicly available. The placement of virtual nodes in the physical IB cluster and the physical network topology determine communication latencies within a virtual cluster (e.g., number of hops). In addition, a certain placement may render physical nodes unusable for other virtual clusters, because of limited QoS resources (see Section 3.2). We are currently working on strategies for this scheduling problem. In contrast to contemporary cloud infrastructure, VMs with access to IB cannot be live-migrated in a transparent way – an additional challenge that we will face in the future.

Acknowledgements

We want to thank the Steinbuch Centre for Computing and Mellanox Technologies for access to test infrastructure and software support for our work.

6. REFERENCES

[1] F. J. Alfaro, J. L. Sánchez, M. Menduiña, and J. Duato. A formal model to manage the infiniband arbitration tables providing qos. *IEEE Trans. Computers*, 2007.

[2] M. Armbrust et al. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[3] A. G. Carlyle, S. L. Harrell, and P. M. Smith. Cost-effective hpc: The community or the cloud? In *Second International Conference on Cloud Computing Technology and Science*. IEEE, 2010.

[4] P. Church and A. Goscinski. Iaas clouds vs. clusters for hpc: A performance study. In *CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, 2011.

[5] A. Dusseau, R. Arpaci, and D. Culler. Effective distributed scheduling of parallel workloads. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM, 1996.

[6] C. Evangelinos et al. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. In *Cloud Computing and Its Applications (CCA 2008)*, 2008.

[7] K. Ferreira, P. Bridges, and R. Brightwell. Characterizing application sensitivity to os interference using kernel-level noise injection. In *ACM/IEEE Conference on Supercomputing*. IEEE Press, 2008.

[8] R. Figueiredo, P. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *23rd Intern. Conf. on Distributed Computing Systems*. IEEE, 2003.

[9] I. T. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang. Virtual clusters for grid communities. In *CCGRID*. IEEE, 2006.

[10] A. Gupta and D. Milojicic. Evaluation of hpc applications on cloud. Technical Report HPL-2011-132, HP Laboratories, 2011.

[11] W. Huang, J. Liu, B. Abali, and D. Panda. A case for high performance computing with virtual machines. In *20th Annual International Conference on Supercomputing*. ACM, 2006.

[12] *InfiniBand Architecture Specification Volume 1, Release 1.2.1*. InfiniBand Trade Association, 2007.

[13] C. Ivica, J. T. Riley, and C. Shubert. Starhpc - teaching parallel programming within elastic compute cloud. In *ITI*. IEEE, 2009.

[14] T. Jones et al. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In *ACM/IEEE SC2003 Conference on High Performance Networking and Computing*, 2003.

[15] K. Keahey and T. Freeman. Contextualization: Providing one-click virtual clusters. In *Fourth IEEE International Conference on eScience*, USA, 2008. IEEE.

[16] J. Liu, W. Huang, B. Abali, and D. K. Panda. High performance vmm-bypass i/o in virtual machines. In *USENIX Annual Technical Conference*, 2006.

[17] P. Mell and T. Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6), 2009.

[18] M. Mergen, V. Uhlig, O. Krieger, and J. Xenidis. Virtualization for high-performance computing. *ACM SIGOPS Operating Systems Review*, 40(2):8–11, 2006.

[19] H. Meuer. The top500 project: Looking back over 15 years of supercomputing experience. *Informatik-Spektrum*, 31(3):203–222, 2008.

[20] PCI-SIG Single-Root I/O Virtualization Specification, `http://www.pcisig.com/specifications/iov/single_root/`.

[21] F. Petrini, D. J. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q. In *ACM/IEEE SC2003 Conference on High Performance Networking and Computing*, 2003.

[22] N. Regola and J.-C. Ducom. Recommendations for virtualization technologies in high performance computing. In *2nd International Conference on Cloud Computing Technology and Science*. IEEE, 2010.

[23] R. Reussner et al. Skampi: A detailed, accurate mpi benchmark. In V. Alexandrov and J. Dongarra, editors, *Recent advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 1999.

[24] J. E. Smith and R. Nair. The architecture of virtual machines. *IEEE Computer*, 38(5):32–38, 2005.

[25] B.-A. Yassour, M. Ben-Yehuda, and O. Wasserman. Direct device assignment for untrusted fully-virtualized virtual machines. Technical report, IBM Research, 2008.