Harri Hämäläinen

Evaluation of the network performance in a high performance computing cloud

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 21.05.2014

Thesis supervisor:

Prof. Jukka Manner

Thesis advisor:

M.Sc. (Tech.) Risto Laurikainen

Aalto University
School of Electrical
Engineering

AALTO-YLIOPISTO
SÄHKOTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Harri Hämäläinen

Työn nimi: Suurteholaskentapilven verkon suorituskyvyn evaluointi

Päivämäärä: 21.05.2014 Kieli: Englanti Sivumäärä:7+55

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoverkot Koodi: S-38

Valvoja: Prof. Jukka Manner

Ohjaaja: DI Risto Laurikainen

Pilvipalvelut mahdollistavat resurssien joustavan käytön. Erityisesti niin sanoituissa Infrastructure-as-a-Service -pilvipalveluissa käyttäjän voivat virtualisoinnin kautta ajaa sovelluksiaan omissa virtuaalikoneissaan ja siten muokata sovellusten ajoympäristöä omien tarpeidensa mukaan. Näissä palveluissa käytettävä virtualisointi lisää yleisrasitetta, joka heikentää sekä laskennan että I/O-laitteiden suorituskykyä.

Tässä työssä evaluoidaan tällaisen pilvipalvelun verkon suorituskykyä. Palvelussa käytetty verkkoteknologia pohjautuu InfiniBand-arkkitehtuuriin, joka on yleinen teknologia erityisesti suurteholaskennassa käytettävissä klusterijärjestelmissä. Evaluointimenetelmät tutkivat verkon latenssia ja läpisyöttöä (engl. throughput) eri skenaarioissa, joissa suureita tutkitaan sekä ilman virtualisointia että virtualisoinnin kanssa. Skenaarioiden tarkoituksena on kartoittaa yleisrasitteeseen voimakkaimmin vaikuttavia tekijöitä. Tämän lisäksi työssä evaluoidaan erityistä SR-IOV-teknologiaa, joka mahdollistaa fyysisen laitteen esittämisen joukkona virtuaalikoneisiin liitettäviä virtuaalilaitteita. Teknologian avulla voidaan yleisesti tehostaa I/O laitteiden suorituskykyä virtuaalikoneissa. Tämän evaluoinnin yhteydessä käytettävissä InfiniBand laitteissa on SR-IOV-tuesta ollut kehitysversio, jota on testettu evaluoitavassa järjestelmässä.

Evaluoinnin tulokset osoittavat käytettävän tunnelointiprotokollan sekä virtualisoinnin I/O-tuen puutteen aiheuttavan suurimmat suorituskyvyn menetykset evaluoiduissa skenaarioissa. Evaluoitu SR-IOV-teknologia on tulosten perusteella kaikissa tapauksissa suositeltava käyttöönotettava teknologia suorituskyvyn parantamiseksi.

Avainsanat: InfiniBand, pilvipalvelut, SR-IOV, virtualisointi

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Harri Hämäläinen

Title: Evaluation of the network performance in a high performance computing cloud

Date: 21.05.2014 Language: English Number of pages:7+55

Department of Communications and Networking

Professorship: Data Networks Code: S-38

Supervisor: Prof. Jukka Manner

Advisor: M.Sc. (Tech.) Risto Laurikainen

The cloud services enable a flexible use of resources. Especially in so called Infrasturcture-as-a-Service style cloud services the users can run their own applications in their own virtual machines and so customize the whole execution environment as needed. However the virtualization introduces an overhead which decreases the performance of computation and I/O-device access.

This work contains a network performance evaluation of this kind of cloud service. The service uses InfiniBand as its network interconnect solution, a technology often used in high performance computing clusters. The evaluation methods study the network latency and throughput in different scenarios. In these scenarios the metrics are studied with and without virtualization. The purpose of these scenarios is to study the major contributing sources for the introduced overhead. This work also contains an evaluation of SR-IOV technology, which enables the mapping from physical device into multiple virtual functions which can be assigned directly to virtual machines. The technology can be used to improve the performance of I/O devices. In this work the SR-IOV technology is studied with InfiniBand devices which are currently having an experimental support for SR-IOV.

The evaluation results show that the tunneling protocol used and the lack of hardware support for virtualized I/O are causing the biggest performance losses in the evaluated scenarios. The evaluated SR-IOV technology is, based on the evaluated scenarios, desired in all cases to improve the performance.

Keywords: InfiniBand, cloud services, SR-IOV, virtualization

Preface

I want to thank my instructor, Risto Laurikainen, and supervisor, Prof. Jukka Manner, for the feedback regarding this work. I also want to thank Pekka Lehtovuori for the chance to work on this topic, as well as the all the members of old CS and CE teams at CSC for the support I've received and especially for the great discussions I've had there.

Otaniemi, 21.05.2014

Harri Hämäläinen

Contents

Abbreviations

10 GigE	10 Gigabit Ethernet
BIOS	Basic Input/Output System
CPU	Central Processing Unit
DC	Direct Current
DMA	Direct Memory Access
FDR	Fourteen Data Rate
GigE	Gigabit Ethernet
HCA	Host Channel Adapter
HPC	High-perforamance computing
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure-as-a-Service
IB	InfiniBand
IBA	InfiniBand Architecture
IBTA	InfiniBand Trade Association
IOV	PCI-SIG I/O Virtualization
IPoIB	IP over InfiniBand
I/O	Input / Output
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
MPI	Message Passing Interface
MTU	Maximum Transmission Unit
NIC	Network Interface Card
OFED	OpenFabrics Enterprise Distribution
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
RDMA	Remote Direct Memory Access
SDP	Sockets Direct Protocol
SDR	Single Data Rate
SR-IOV	Single-Root I/O Virtualization
ULP	Upper Layer Protocol
VLAN	Virtual Local Area Network
VMM	Virtual Machine Monitor
vNIC	Virtual Network Interface
VT-d	Virtualization Technology for Directed I/O

1 Introduction

High performance computing (HPC) is a term used to refer a set of computing solutions which can be applied to heavy computational problems. Most commonly this means the usage of special computing hardware, such as a super computer or a super cluster. These system are designed to process the computation tasks requiring the amount of resources not available on commodity hardware such as basic workstation computers. The HPC systems are today used to solve wide range of problems from weather forecasting to DNA sequencing and from financial market analysis to fundamental questions in physics.

The superior computational power comes naturally with some drawbacks. The high initial price and operating costs or the lack of expert knowledge to setup and administer the HPC installations prevents some organizations from acquiring these systems for their own use. In these cases the organizations can still have possibility to use shared HPC resources offered by another organization taking the financial and administrative responsibilities of the system. Also the shared HPC system are static by nature and the customization of the computation environment must usually be carried by a system administrator who controls which applications and libraries are installed. For the users of the systems this means either to use the tools available in the system or to request an installation of the tools required, which starts to cause troubles for administration of the system when large number of users are to be served.

While Infrastructure-as-a-Service (IaaS) cloud services are also usually larger systems with lots of computational capacity they are generally not thought as HPC platforms. The purpose of such IaaS cloud is to offer an on-demand access to a pool of low-level resources, usually a hardware or virtual machines managed by a cloud middleware services. These middleware services consist of the underlying host operating systems, hypervisors running the virtual machines, and accounting and control processes managing the resource access from machines.

The on-demand access means that the user can request an allocation of system resources for himself and the request is served if the system can fulfill the request. The resources are allocated for that user until they are explicitly released back to the pool of available resources. By allocating a larger set of resources the user of IaaS cloud system can also compute problems usually solved only on traditional HPC environments. Compared to HPC systems the allocated cloud resources are usually given to user with higher degree of freedom. User can select which operation system to run and which applications to use. The HPC systems are usually shared systems among multiple users without superuser access to the system. Using arbitrary applications on these systems might require negotiation with system operators.

However the improved flexibility of cloud computing approach has its costs. Virtualization used with the cloud services adds overhead which might degrade the performance of the system. Also serving the different users in a fair and secure way has challenges in these environments.

In HPC systems the system performance is affected by multiple factors. Traditionally the

HPC workloads have been limited by the available CPU and memory resources i.e. these workloads are said to be CPU intensive. In super-cluster environments the computation power usually comes from multiple host machines interconnected with a fast network allowing efficient parallel execution of computation processes. To fully utilize the fast interconnection between multiple CPU resources also the applications must support efficient communication and computation. Software optimization for HPC is a complex area requiring deep expert knowledge. Several libraries implement a message passing interface (MPI) to provide standard and efficient interfaces for inter-machine communication in a distributed environment, which makes the process of implementing efficient communication less error prone.

While traditional HPC systems usually outperform virtualized cloud systems, an HPC cloud has several use cases where the increased flexibility is more important than the performance lost due to the virtualization layer. In addition to the flexibility provided by virtualized on-demand service the virtualization layer can be used to provide an alternative platform for legacy applications requiring e.g. a certain operating system configuration. In these cases the virtualized cloud service offers a plan of continuation for the usage of such applications even if the original hardware and operating systems are phased out.

1.1 Problem Statement

HPC cloud environment could offer a flexible way of accessing HPC resources. The user of IaaS cloud system can either build a system highly customized for his needs or use generic prebuilt system usually provided by a cloud operator. Because of the possibility of customization the user can run both the legacy systems as well as the modern systems on the same virtualized hardware.

The biggest question in this however is how much the virtualization affects the system performance. Especially in case of the HPC the interest is in the virtualization overhead of the network fabric optimized for rapid message passing and required by many parallel HPC applications to achieve their full potential. This is due to the recent developments in the CPU virtualization support making the effect of virtualization for CPU intensive computing on single host less severe compared to systems where the computation is distributed over multiple hosts requiring large number of I/O operations from the network hardware. In these cases the performance of virtualized network I/O has been lacking behind until recently with development of technologies such as the single root I/O virtualization (SR-IOV).

To understand the effect of virtualization for network performance in detail and to discover potential bottleneck components in the system an analysis is required on multiple layers in the system. So far mostly the suitability of IaaS cloud for HPC is studied on commercial IaaS clouds such as Amazon's Elastic Compute cloud [11]. In these cases however it is hard to get any deeper insight from the system as only the virtualized layer is visible to user.

The high network throughput is usually always desired as long as it does not affect the

network latency too much. With bigger data flows this can happen for several reasons like processing, queuing and congestion delays. As a consequence some transfer mediums can have a high throughput but a very poor latency and vice versa.

The analysis is based on two general characteristics of network performance: latency and bandwidth. The latency is used to report the delay introduced by endpoints and the network connecting the them. In case of the HPC systems the ultra low latency is usually key requirement due to the message passing between the system nodes. In practice this means the latency being in microsecond range. The bandwidth reports the throughput or amount of traffic which flows from the endpoint to another.

The problem addressed in this thesis is to get an insight from the different parts of the network stack affecting the network performance. The focus is on comparing the virtualized and the native network performance on top of InfiniBand network infrastructure. A test system having InfiniBand adapters is configured with latest firmware from the hardware supplier and a series of tests are executed to evaluate the recent development in hardware based technologies which are promising better I/O performance characteristics for virtualization.

1.2 Objectives and Scope

The problem analysis described in this thesis is carried for specific infrastructure-as-a-service (IaaS) cloud service called Pouta and operated by CSC - IT Center for Science. The exact details of the system under study and the analysis methods are given in detail in Chapter 5.

For the analysis the goal is to understand the system performance limitations on multiple levels in the system. This knowledge provides insight for the developers of a such system to understand two key aspects:

1. How much slower the networking is in a virtualized cloud environment when compared to a system without virtualized hosts.
2. Which system components are the greatest contributors for degraded network performance.

The motivation behind this goal is to find potential places for improvements allowing the system to provide higher service performance for users. The analysis is based on the data gathered from the system under the study with several tools and applications reporting certain network performance characteristics. As a proposal for a solution an evaluation of Single-Root I/O Virtualization (SR-IOV) technology is tested and the results are compared to the results from a current HPC cloud deployment.

In this thesis actual HPC applications are not used and the focus is on the lower layer of communication protocols offered by the system. This is because the implementation and usage of such applications requires deep knowledge of the specific application and how it

uses the resources it has. By studying the performance characteristics on the lower layer we can still give certain bounds for the characteristics.

Based on the results of this thesis the cloud system administrators should be able to reason about the suitability of the InfiniBand fabric for virtualized cloud environment. As with the most technologies also the ones used in this thesis are under constant development, which is why in this thesis the question about the suitability is studied with the current technologies available.

1.3 Related Work

The suitability of cloud computing systems for high performance computing (HPC) has been studied by several groups [46, 29, 38, 65, 32, 75, 27, 56, 69]. The results from these studies indicate the system interconnection to be one of the main reasons for inferior performance compared to traditional HPC systems. While the interconnection options available in the most public cloud services are not as far efficient as the fast HPC interconnects the flexibility and vast number of resources available in cloud services have opened an interesting questions about their suitability for HPC like computation.

Many studies are comparing different systems built on top of Amazon EC2 instances against some HPC system [38, 29, 32, 69]. The problem in these approaches is that the underlying infrastructure is not controlled by the researchers which is why for example the networking characteristics can have variation based on how the virtual machines are placed into the cloud [76].

In a virtualized cloud environments the virtualization overhead of I/O operations is the other natural reason for degraded performance due to interconnection overhead. Wang and Ng [71] have analyzed the effect of virtualization for network performance with Amazon EC2 computing cloud instances are they reported high delay variation with unstable throughput which are likely to affect the scientific workloads in such an environment. The effect of different hypervisors for network performance is also studied by Walters et al. [70]. Especially with such software comparisons the actual results from their studies might already be outdated, but their findings are interesting as different hypervisors are having very different characteristics for network performance.

One of the prominent technologies to improve the I/O access from virtual machines is so called Single-Root I/O Virtualization (SR-IOV). This thesis includes tests using SR-IOV with InfiniBand network interface cards. So far only few studies have been published about the actual performance improvement when SR-IOV is used [18, 26].

From these studies it is clear that the HPC cloud services are studied up to some extent, but extensive and complete studies are missing. Studies comparing traditional HPC services against generic virtual machines in the cloud will not do justice for HPC cloud services in general as the another one is designed to have high performance and other one to have high flexibility.

1.4 Outline

The rest of this thesis is organized as follows. In Chapter 2 the definitions of high performance computing (HPC) and cloud computing are given. For the cloud computing also several use cases are described. The chapter contains also an introduction to concept of virtualization. The development of virtualization technologies are described from historical viewpoint to point out how different requirements and limitations have guided the work. Current virtualization technologies are introduced with their basic principles. Hardware support technologies for virtualization, like single-root I/O virtualization used in this thesis work, are introduced.

In Chapter 3 the technologies used to build Infrastructure-as-a-Service (IaaS) cloud services are introduced. The discussion looks for alternative solutions for both the offered service and the technology solutions used to build such a cloud service.

In Chapter 4 the InfiniBand Architecture (IBA) and its usage in the high performance computing are introduced. The methods of accessing InfiniBand fabric from applications are given. Also the performance effect of different access methods are described.

In Chapter 5 the system under study is described and the evaluation scenarios are defined and the results from these scenarios are reported. The installation procedures required by some evaluation scenarios are described.

Final conclusions from this work are given in Chapter 6.

2 Background

In the first section of this chapter the basics of high performance computing (HPC) and the cloud computing are described. While both are relevant concepts to the subject of this thesis these are only discussed briefly here as both topics are containing huge number of studies and active research.

The virtualization technologies discussed in this Section 2.2 make it possible to run one or multiple virtualized environments on top of single host. This host is here referred as hypervisor even though terms like virtual machine monitor or VMM for short are also often used. First the section discusses the history of virtualization after which some virtualization models and hardware support developed especially to make the I/O operations more efficient in virtualized environments are discussed. Especially the SR-IOV technology discussed here is used in the evaluations discussed in Chapter 5.

In the second section the previous research related to the field of this thesis is introduced.

2.1 Cloud computing

The National Institute of Standards and Technology (NIST) has given an often used definition for the cloud computing [39] which states the following:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction."

This definition however includes a wide range of different services which are discussed in detail in Subsection 2.3. There are however a few common characteristics for these services which are associated as a crucial features for any cloud computing service. These characteristics are described next.

The ubiquitous service is a service which appears to be accessible from everywhere and virtually with any kind of device. The service itself is not clearly bound to a specific host, and the computation takes place in a physical location, which is not necessarily exactly known by the user. For the cloud computing this means that in ideal case the computation resources are universally available to users regardless of their local systems used to access the service. In reality some limitations might exists such as poor network connectivity preventing the user from accessing the remote cloud resources or the lack of capabilities required to access the cloud computing interfaces. Given the current popularity of different cloud services these limitations are not having a huge effect on the offered services.

The on-demand nature of cloud services allows users to request and allocate cloud resource as they need without human interaction to grant the resources or manage the resource access. This allows rapid provisioning i.e. allocation of requested resources and scaling the resource allocation to match the variations in the service usage.

The definition for convenience is more tricky. Machine processable application program interfaces (APIs) allow a convenient way of controlling the cloud resources and most cloud services offer management APIs over Hypertext Transfer Protocol (HTTP). But like the possibility of accessing the service virtually from anywhere at any time is surely also very convenient. Some cloud computing services also offer customization and service flexibility at much larger extent than similar non-cloud computing services. This flexibility is one of the major contributing factors why cloud computing has recently drawn the attention of the public.

Most often some kind of virtualization solutions are used to provide more control and flexibility for the management of the infrastructure. The concept of virtualization is discussed in detail in Section 2.2. For the cloud computing the improved flexibility means decoupling the cloud applications from physical hardware with virtualization. This decoupling allows the virtualized system to start new virtual machines with minimal overhead and usually completely without human interaction. With the decoupling services can also be migrated from one physical host to another more easily which can also improve fault tolerance in case of hardware failures.

The virtualization layer is not a strict requirement. Recently especially in HPC cloud environments so called bare metal [74] cloud solutions have gained more attention. In the bare metal cloud environments the virtualization layer is completely missing which eliminates the virtualization overhead, but reduces the amount of flexibility gained from the virtualization. In the bare metal environments the operating system is provisioned to a physical system through some management interface which handles the provisioning of the system.

2.2 Virtualization

Virtualization technologies started to develop over 40 years ago [24]. The motivation to develop virtualization technologies in the era of mainframe computers was to provide ways allowing multiple operating systems to be operated on top of a physical system. Traditionally these physical systems were allocated for one user at a time as the user was executing commands in privileged mode after which the system was reconfigured for the next user. With virtualization it became possible to run multiple operating systems in virtual machines on top of a host operating system often called a virtual machine monitor (VMM) or a hypervisor.

During the 1980s and 1990s the research focus shifted away from virtualization as modern multitasking systems replaced big mainframe systems [58]. However in the beginning of 21st century virtualization technologies started again to draw the attention of the public as the x86 architecture turned out to be very well available with reasonable price and the system performance development continued to increase.

While the server hardware is now cheap and easily available the need for computational services has also increased drastically. To fulfill these needs with straightforward approach of setting up new physical hardware installation with its software stack the out-

come will not only take lots of physical space and power, but it also requires lots of management from the service provider whose responsibility is to keep the software stacks updated and manage the user access in these machines. Virtualization as a solution provides means of decoupling the software stack from hardware while it also can make the space and power consumption more efficient. Also with virtualization it is easier to deploy systems with distributed components deployed in the virtual machines rather than having a separate physical host for each system component [72].

Today well known use cases for virtualization are the different Infrastructure-as-a-Service (IaaS) cloud services discussed in Section 2.1. Such services offer computing resources to be allocated from a public service and the customers can run their systems and applications on top of the allocated resources. This not only removes the need to own physical computing system in order to offer computing services but also provides flexible ways to scale up the size of the service with the system usage.

Another virtualization trend today is the virtualized networking. With virtual networking multiple virtual machines can be connected with virtual network topology. The direct consequence from virtual network topology is an improved service flexibility as multiple configurations can be assigned without the need to physically reconnect the links. This eases the physical management work in data centers, improves the flexibility of provided services and allows automated control of the virtualized networking infrastructure. In addition to this also different link characteristics could be virtually assigned to links such as the bandwidth or latency of the links.

But the x86 processor architecture was not designed to support virtualization. Due to this it is assumed that the operating system is running on the bare-metal hardware [68]. With virtual machines running on top of single hypervisor this is not the case anymore. Multiple virtual machines are sharing the resources and these machines are supposed to be isolated or sand boxed from each other. The sand boxing tries to guarantee that a virtual machine cannot access the resources allocated for another virtual machine or prevent another virtual machine from accessing the resources assigned to it. The implementation of this resource assignment and access control is usually a responsibility carried by the hypervisor, but lately some of these techniques are offloaded to special hardware circuits to make the virtual execution more efficient. In some of these cases the hypervisor is merely controlling the assignment of the resources like instructing the memory management unit to map certain memory regions to virtual machines. Some of these techniques are discussed in Section 2.2.

There are multiple models to implement the details of virtualization. The most drastic trade-off between these models is in performance and flexibility of the virtualized environment. The models used most often are:

- Full virtualization, full hardware simulation, guests require no modifications
- Paravirtualization, hardware is not simulated, guests require modifications

The full virtualization usually implements almost fully simulated hardware so that the guest system running on this virtual environment does not have to have any modifications

for virtualized environment. This efficiently make the virtualization layer completely transparent for guest which makes the migration and portability of the virtual machine easier while improving the isolation of virtual machines.

One implementation of full virtualization approach is so called binary translation. Binary translation, used in VMWare products, allows guest system to run directly on host CPU but it traps the privileged x86 instructions and the hypervisor then translates these to emulated instructions.

The paravirtualization approach adds a specific interface to hypervisor which the guest operating systems use to request certain actions (such as direct memory accesses). This allows otherwise slow virtualized operation to be executed faster under the hypervisors control. The execution speed up is gained because neither a full device virtualization nor instruction trapping are needed in hypervisor level. The downside of this methods is that the hypervisor interface requires a special support to be present in the kernel of the guest system, which adds extra complexity for the usage of the system.

To make the virtualization more efficient the processor manufacturers started to include develop virtualization support for x86 architecture which in 2005 [55] resulted in two widely used processor virtualization technologies, namely VT-x from Intel and AMD-v from AMD. These technologies are today found on most processors and the focus has shifted to developing better I/O virtualization that required less or no involvement from hypervisor to manage the I/O device usage. With these extensions the method of full virtualization has become more efficient and is used e.g. in the Kernel-based Virtual Machine (KVM) [48] which has been included in the mainline Linux [35] kernel since the version 2.6.20.

To understand the performance problems with hypervisor managed the I/O operations one can think a virtualized system with a hypervisor running multiple virtual machines connected to some network as shown in Figure 1a. These virtual machines are having a very intensive traffic consisting of packet flows coming from the physical network link and routed to specific virtual machine by a software switch of the hypervisor. Each time a new packet has arrived from network to network interface card (NIC) a processor interrupt is required first to check the incoming packet. Based on the information in the packet another interrupt is created for the processor core responsible to run the virtual machine the packet was destined to.

The performance problem in this case is that for most of the incoming packets two interrupts are created. One for the core responsible to handle the incoming packets and another for the core responsible to handle the packet at the destination. In the networks with high bandwidth available the interrupt processing starts to limit the maximum achievable bandwidth. To make the processing more efficient less interrupts must be generated. For example with the Intel VT-D or with the AMD's AMD-Vi pass-through technologies a PCI device on the host can be attached directly into the virtual machine running on the host. In addition to I/O device assignment these technologies must also implement address translation for DMA addresses as well as handle the remapping of all device interrupts [49]. This way the device can be later accessed from the virtual machine with

no overhead. The limitation here however is that a single device can only be assigned to single virtual machine at time, a limitation which is only recently lifted by PCI-SIG I/O virtualization (IOV).

Intel VMDq [15] is a hardware-based virtualized traffic management technology which makes it possible to offload parts of the data traffic management. In practice this means that hypervisor can configure multiple hardware level I/O queues for its virtual machines and the traffic targeting one of the virtual machines is sorted in hardware level. Each of the queues have their own interrupt which allows the workload to be spread among multiple processor cores and offloaded classifying and sorting reduces the number of CPU cycles required to process the incoming packets. The hypervisor must still handle the data transfer to the address spaces of virtual machines from the NIC and the other way around.

Single-Root I/O Virtualization (SR-IOV) [52] is a PCI-SIG group [50] specification defining a standard means for virtual machines to bypass the involvement of the hypervisor in the I/O device data management. SR-IOV makes it possible to turn any PCI device supporting SR-IOV into multiple virtual devices which then can be associated with processes like virtual machines running on the system. For these processes the associated virtual function behaves like the real physical device.

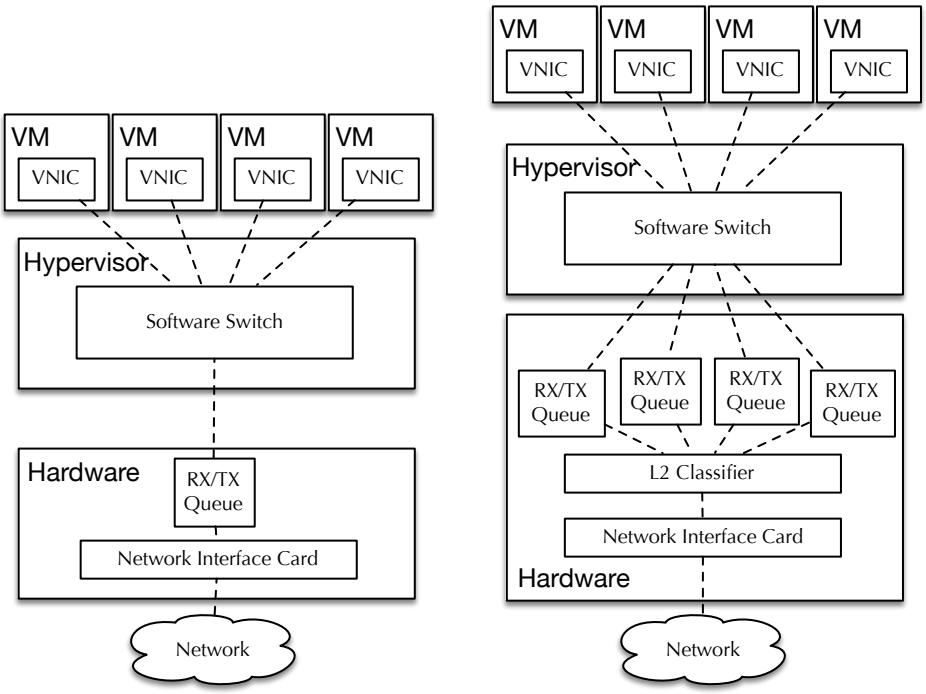
PCIe devices having SR-IOV capabilities are called physical functions and these devices are used to manage the set of virtual functions. Each of the virtual functions can be mapped to a single virtual machine which sees the virtual function as a normal PCIe device. With the SR-IOV the device operations can be made directly to the address space of the device without hypervisor's assistance. This also means fewer context switches and better I/O performance.

2.3 Use cases for cloud services

This section introduces the most often used categorization for different cloud services. This categorization to distinct different cloud service types is given in Table 1 though multiple variations also exists. For the user of the services the difference is in the level of interfaces offered. This interfaces range from the low-level virtual machine access to web applications.

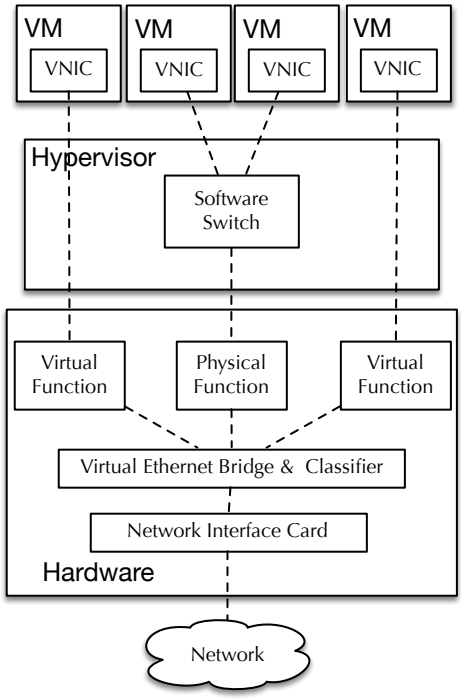
The Software as a Service (SaaS) service model provides an on-demand access for an user to application services. Users starting to use a SaaS service can usually start using the software easily without special installation process as the whole stack from the hardware up to the software and the date are hosted by the cloud service providers. Such a conceptual stack is given in Figure 2.

Well known examples of SaaS cloud services are different business applications such as the customer relationship management application Salesforce [6]. The user can buy the SaaS service from the provider and can usually start to use the application immediately without first setting up the hardware system or running low level application setup and



(a) Hypervisor managed access

(b) VMDq



(c) SR-IOV

Figure 1: Architectural view on virtual machine network access with generic hypervisor managed access, VMDq, and SR-IOV

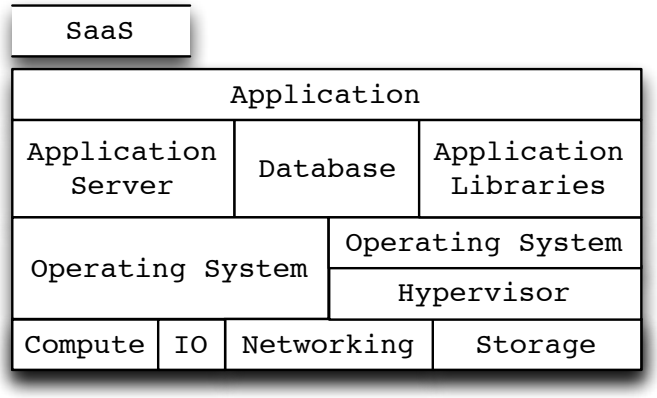


Figure 2: Conceptual cloud stack for Software-as-a-Service

maintenance. The SaaS cloud services are usually accessed with *thin clients*, such as web browsers, which communicate with the service APIs over HTTP while the computation and data handling are happening in the cloud service.

Model	Description	Example
SaaS	The cloud providers offers an application services and users can access the application	Salesforce [6]
PaaS	The cloud provider offers a computing platform service which usually includes a operating system environment with some pre-configured environment on top of which the user can run the applications	Heroku [2]
IaaS	The cloud provider offers physical or virtual machines to user with some basic low-level infrastructure services such as networking. User is responsible to setup the rest of the system such as operating systems.	Amazon's EC2 [11]

Table 1: Often used cloud service models with well-known examples of that specific service type

With Platform as a Service (PaaS) style cloud service the cloud service provider offers a platform for users who can then implement their own software on top the provided platform stack like the one in Figure 3. The platform can provide application development and runtime environments, data storage services like databases, and ready-made solutions for easily achievable scalability.

An example of PaaS style cloud service one could take Heroku [2], a cloud service platform offering application runtime hosting on its cloud platform. The platform offers application runtime environments with often used middleware services such as database access, ready-made solutions for application scaling, and tools for application usage ac-

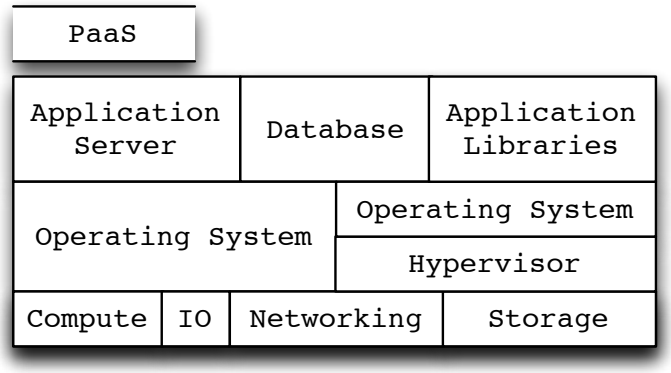


Figure 3: Conceptual cloud stack for Platform-as-a-Service

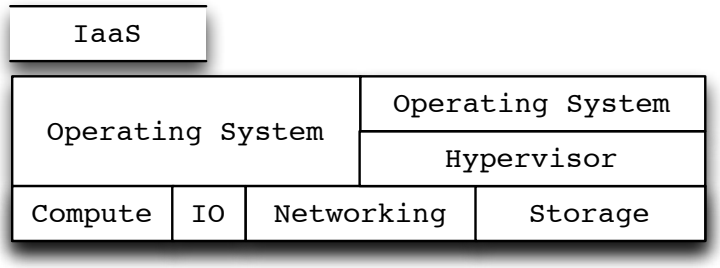


Figure 4: Conceptual cloud stack for Infrastructure-as-a-Service

counting. The services like Heroku make the process of starting application development more light-weight as the tools and required libraries are provided without need to have a local setup for these.

The lowest level of cloud service models is so called Infrastructure as a Service (IaaS) model as shown in Figure 4. The IaaS model provides a set of physical or virtualized resources for users. Users are to execute operating systems or applications directly on top these resources or on top of hypervisor. IaaS style cloud services usually also provides means to attach storage services or configure virtualized network infrastructure for the virtual machines allowing machines to interconnect with each other and communicate to other services and clients over Internet.

The most well known example of this cloud service type is the Amazon's EC2 cloud which sells virtual machine resources to customers. The customer can select a desired amount of computational power and the operating system platform to be used. Basically the customer is free to set up any kind of system configuration to be executed on his virtual machine setup. The virtual machine instance is launched upon user request and terminated when the user wants to stop the usage of the virtual machine. During the time the virtual machine is active the physical computation resources are reserved for the user. In some cases the service might also use over-committing of resources meaning e.g. that the same processor core is allocated for multiple users at the time, effectively lowering the performance if both users want to utilize the core simultaneously.

Another classification for cloud services is whether the service is intended for public to use or not. The public cloud services are usually provided by a service provider whose business is to sell the cloud resources, like Amazon's EC2. Another approach is the private cloud services which can be seen as intended for organization's internal use.

2.4 High Performance Computing

High performance computing is a generic term used to refer to systems which are suitable to process large computation tasks notably faster compared to common personal computers (PC) i.e. workstations and laptops. Some distributed systems consisting of multiple PCs can be though as a HPC system as it is with projects such as Folding@home [1] which, according to their statistics, is currently having a computational capacity over 36 x86 petaflops[1]. However more often HPC as a concept is associated with physically big computer system installations consisting of huge number of central processing units (CPU) and vast amount of memory available. As the HPC problems usually are solved by utilizing the parallel execution on multiple CPUs the interconnection between the CPUs and the memory access has a requirement for low latency [44].

With traditional HPC systems the requested resources are reserved and dedicated to one application at time for the whole life time of the application. This usually prevents any competition on resources between different applications in the system and hence results

[1] http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats2

in no need for performance isolation [9]. When there are more resource requests in the system than there are resources available the application execution will wait its turn in job queue until resources are available. This property of HPC systems is one of the major differences between the HPC and the cloud systems, where resources are usually pooled and shared between multiple applications and workloads simultaneously.

Often the HPC systems are targeted to offer maximum computational capability rather than capacity. This characteristic is desired for systems aimed to solve single large problems one at the time rather than share the resources among multiple problems leading to longer execution times. The fair-share scheduling distributing the available resources evenly for every user in the system might lead to resource contention, a situation where users in the system are competing for shared resources, which is likely to cause race conditions or thrashing[2]. The resource contention and the performance degradation it causes are a severe problems especially in HPC systems, which are supposed to provide high performance by the definition.

To manage the resource allocation in the HPC systems a special job queuing subsystem can be utilized. The purpose of the job queue is to schedule the user processes, often referred as jobs, so that the system resources are not over-utilized[31]. In a shared HPC environment the jobs are submitted into the job queue. Jobs are taken into execution from the queue when there are resources available to fulfill the resource requirements of the job. The queue is controlled by a queue manager which can also implement scheduling policies and enforce other restrictions for jobs to guarantee fair treatment among the users. Such guarantees might limit the maximum life time of a computation job or the maximum amount of resource for one job. When the user must

Designing and utilizing HPC systems with maximal capability in mind is a challenging task. Several problems raise when the performance gain is received from the multiple interconnected processors in the system. For the system design this means designing for low latency so that the delays are causing minimal impact on the system performance. To achieve this the HPC systems are utilizing high-end hardware components with specialized interconnection technologies which can provide sub-microsecond latency with the aid from specialized interconnection topology, transport engine offloads, and lock-free switching. A switched fabric topology as shown in Figure 5 is often used to increase the total available bandwidth with multiple transport paths.

The large number of processors in typical HPC system are often so called multiple instruction, multiple data (MIMD) architectures. The processors in MIMD system can work independently from each other, with different instructions and with different data. The memory used in these systems can either be shared with every processor (Figure 6) or distributed among processors (Figure 7).

Also the physical system environment has more weight in HPC system design. While the large HPC systems still require more space and power capacity the biggest problem

[2]Thrashing in computer science means an event in computer system when misses in virtual memory or in cache are constantly occurring due to high number of access requests to different locations. Due to this the performance gain from the usage of the virtual memory or cache is lost.

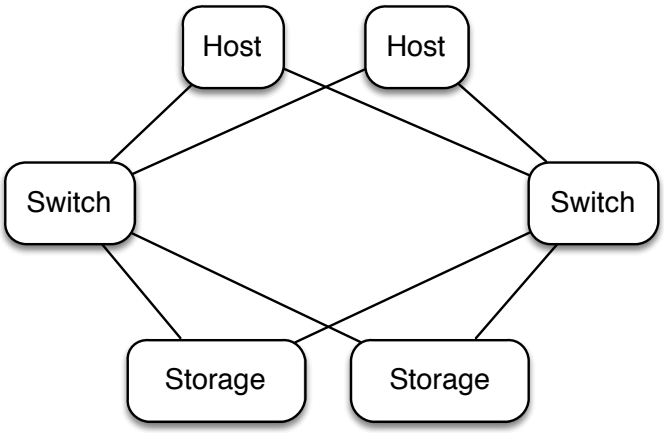


Figure 5: Switched fabric topology

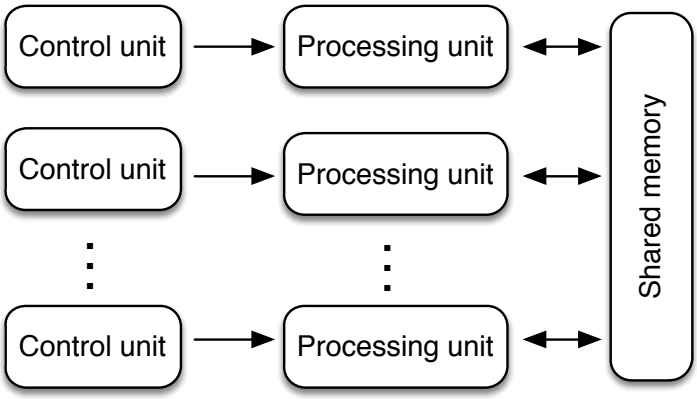


Figure 6: MIMD with shared memory

when large number of processing units are placed closely to each other is the heat the components start generate. When the system temperature grows too high it will hurt the system performance which is why the system must have a proper cooling solution. To remove the hot air from the system either a liquid or an air cooling can be used.

From the user perspective the biggest issue is the problem of writing applications which scale up efficiently with the increasing number of added processor. The traditional tools from the research field of concurrent programming, such as threads and tools for interprocess communication, are hard to apply correctly and might require extensive debugging. To ease the programming process of massively parallel systems several application program interfaces (APIs) with associated libraries are invented. One of the most well-known example of such is the message passing interface (MPI). The MPI is still after twenty since its definition the *de facto* way of programming the parallel systems in a scalable, portable, and efficient way [61].

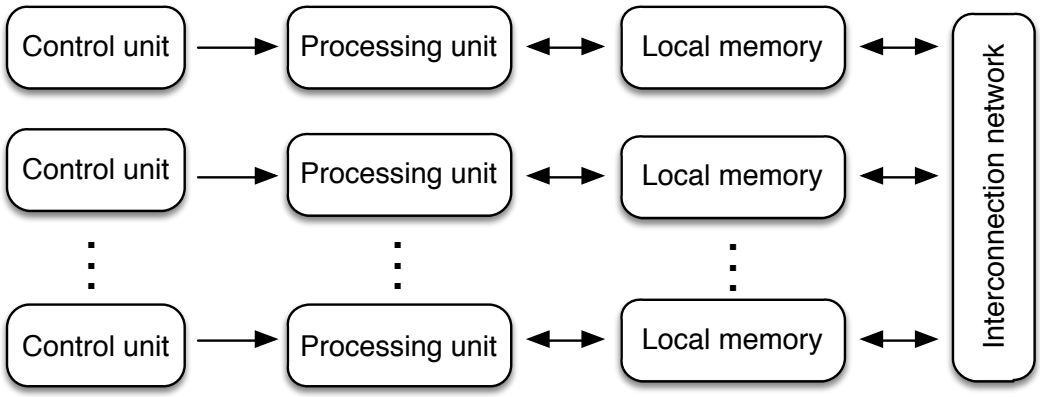


Figure 7: MIMD with distributed memory

Up to this point the typical application profiles for HPC and cloud computing systems have been very different. For the HPC applications the typical workload is CPU intensive job requiring lots of computation such as different modeling problems in physics, engineering, and bio sciences [63, 62, 47]. These HPC systems are usually having specialized hardware components optimized for heavy computation providing e.g. low latency network access to exchange data with processors on remote nodes.

Scientific cloud computing is characterized by different problems revolving around the concept of big data [37] Such problems are dealing with large amounts of data which are processed with tools like MapReduce [17] and Hadoop [60]. While

In most of the public cloud services the hardware or virtual interface for hardware is similar what available on markets today so that most applications can be executed on these systems. Today services like Amazon web services [11] offer also HPC cloud resources which are aiming to offer HPC performance characteristics.

2.5 Summary

In this chapter the concepts of cloud computing, virtualization, and HPC are introduced. The term cloud computing is used to refer a huge variety of different computing services, which share certain characteristics. Especially the cloud services usually can provide improved service flexibility, on-demand service, and convenient usage.

To achieve the promised high flexibility the resource in the cloud should be decoupled from the hardware. This often means decoupling the hardware from the offered service. One way to achieve this is to utilize virtualization, which allows running virtual machines on top of physical system. Like with bare metal provisioning, the virtualization is not strict requirement for all of the possible services defined as cloud services. However in the most cases virtualization is one of the most prominent cloud enabling technologies.

There exists several levels of different cloud services. This chapter introduced most often

used categorization for these different services, namely the SaaS, PaaS, and IaaS types of services.

Finally this chapter introduced the characteristics for HPC. The large computational scientific problems require these specialized system offering high computational performance. Designing HPC systems is very different from designing commodity hardware systems due to latency and throughput requirements.

In this thesis the focus is on the IaaS style of cloud service offering virtualized HPC resources. The focus is especially on how much the virtualization costs in terms of system performance. As shown in this chapter the question of feasibility of HPC cloud is to what extent the users of HPC cloud resources are willing to sacrifice the system performance in order to get more flexible system.

3 Design of HPC Cloud

This chapter discusses about the technology alternatives used to build IaaS cloud services with HPC use case in mind. The focus is on hardware and software alternatives. The chapter starts with a look into other IaaS services available and their usability to fulfill the need for HPC cloud service. Then the focus is shifted on technology alternatives comparing different hardware and software options commonly available and used.

To build a reasonable HPC IaaS cloud the system developer must meet the goals of building both a reasonable IaaS service and reasonable HPC service. This somewhat trivial statement however makes it clear why the goals of HPC IaaS cloud are hard to meet because of the goals and requirements of the different kind of services are combined. As discussed in section

3.1 Service Alternatives for HPC Cloud

Rather than building a whole new HPC service any rational organization is going to check first the availability and possible suitability of such a service before starting the service planning. The high success of the Amazon's EC2 computing cloud has caused multiple of similar service competitors to appear. At the moment the competition of the IaaS market shares is still greatly dominated by Amazon's service [14].

Initially Amazon's EC2 cloud has offered only generic virtual machine instances serving purpose of generic computing. This approach sets neither guarantees nor deterministic bounds for computation performance or network latency. Due to unfortunate scheduling decision or limited resources the several virtual machines might end up being physically separated with multiple network link hops leading to increased latency.

In the end of year 2013 Amazon announced new cloud virtual machine types for HPC applications. These included both compute-optimized C3 and memory-optimized R3 instance types and a new G2 type for graphics processing unit (GPU) computation. The C3 and R3 instance types include SR-IOV enabled network access from virtual machines. The network infrastructure used in EC2 services is using 10 Gigabit Ethernet.

Glenn Lockwood has analyzed the network characteristics of C3 instance in his blog post [23]. In terms of latency in the communication between virtual machines his findings show performance improvements for small MPI message sizes from 65 microseconds to 40 microseconds when SR-IOV is used. These results are still far from the native, non-virtualized latency of 20 microseconds. The reported throughput performance improvement with SR-IOV is minimal. From these characteristics it is a questionable whether or not the service is actually providing HPC resources. Strict definitions do not exists but certainly some other services using alternative network technologies can provide greatly better service quality.

From the economic point of view when the service billing is based on reservation time of the resources the best returned value is gained for the invested funds when resources are

constantly fully utilized. If billing is based on the resource utilization, like used CPU time, the invested funds are spent according to service usage. In some cases the computational needs are occurring periodically with longer idle periods between the periods with higher load. In these cases the outsourced service provider can provide beneficial cost savings. For the cases where the number of heavy computation tasks is constantly higher than the available resource to compute these the service is going to be constantly on full load. In these cases the ownership of such system can be rationalized.

For some use cases outsourcing the cloud service and the data handling could cause privacy issues and problems in data handling laws. Such might be the case for example with business critical data or with medical patient data, which usually must be processed without exposing the data into public. In these cases pushing the data to service located in different country with different data privacy laws can be seen as a violation of this policy. Having a private cloud installation is likely going to require more resources to set up and to run but can circumvent some of these issues.

3.2 Software stack alternatives

The software stack of the IaaS style cloud consists of the host operating systems running several services and interfaces. These services are database instances, hypervisor processes, interprocess messaging subsystems, distributed filesystem daemons, and accounting processes just to name a few. The interfaces to different services allow different system components to interact with each other and with the users of the system.

A more concrete view into different services present in IaaS style cloud service can be obtained from the following list of core services present in OpenStack cloud middleware [5].

OpenStack Nova
: The OpenStack Nova component manages the virtual machine instances. Nova service can abstract multiple hypervisors under its interfaces.

OpenStack Keystone
: The OpenStack Keystone is the authentication service which validates user credentials and provides authentication token used to access different components requiring authentication.

OpenStack Neutron
: The OpenStack Neutron provides virtualized networking services used by virtual machines.

OpenStack Glance
: The OpenStack Glance is the virtual machine image register. It is basically just a library of available disk images used to launch new virtual machines.

OpenStack Swift
: The OpenStack Swift is an object storage service used to store data which can be

read by multiple clients.

OpenStack Cinder
The OpenStack Cinder is a block storage service used to abstract away the details of different block devices and provide unified interface for virtual machines to access these devices.

OpenStack Telemetry
The OpenStack Telemetry is a metering component used to collect measurements from the virtual machines like used CPU seconds.

The purpose of the cloud middleware software like OpenStack is to tie together the different services required to provide IaaS cloud service. For example with the OpenStack Nova the actual virtualization can be provided by any supported hypervisor implementation like KVM or Xen. The differences between the hardware assisted full virtualization of KVM and para-virtualization approach used in Xen are discussed more in Section 2.2. For the users the KVM approach is going to be more convenient as the guest operating system in virtual machine can be executed without any modifications. While these two hypervisors are having different approach for virtualization the performance characteristics are pretty much the same [43].

The OpenStack being an open source product has gathered huge number of active contributors to back its development. While there exists other open source cloud middleware alternatives, like OpenNebula [20], OpenStack is currently providing more advanced features and is more focused to provide a public cloud solution, like Amazon's EC2, while OpenNebula can be seen more like a solution for datacenter virtualization [30]. In addition to open source public cloud solutions multiple companies are offering commercial public cloud solutions or commercial support for OpenStack based public clouds.

The choice of the operating system running the cloud middleware is almost fully dominated by different version of Linux distributions. The same goes with the TOP500 list of the fastest HPC systems in the world [7]. Majority of the HPC systems are running some Linux based operating system.

In the near future container based virtualization technologies like Docker [42] might provide interesting alternatives also for scientific workloads. For example the Docker allows lightweight packaging and deploying of applications while it also allows packaging of all the necessary dependencies for the application. In the ideal case a complex application with its dependencies has to be assembled once after which it can be executed in all environments supporting Docker containers.

3.3 Interconnection alternatives

The role of network interconnection is crucial in HPC environments. With poor interconnection performance characteristics the communication will cause long delays for computation that will dominate the overall performance and cause the system not to behave in a fashion typical to HPC systems.

Two performance characteristics are crucial for the HPC interconnect. Namely the communication latency and bandwidth. The latency means the time delay it takes from the cause until the effect of the cause happens. For example the cause might be an event of message being sent by the sender while the effect is the event of message being received by the receiver.

For any communication happening over electromagnetic signals the wave propagation speed will set a theoretical minimum latency $t = s/v_m$, where s is the distance the signal must travel and v_m is the wave propagation speed in the medium convoying the signal. While for electromagnetic wireless signals traveling in vacuum this wave propagation speed equals to the speed of light the value will be less for signals traveling in some other medium like optical fiber or copper wire.

In addition to the propagation delays also the transmission of the message will cause additional delays as well as the processing and possible queuing which can take place in the sender or in the middleboxes participating in the forwarding of the message.

While the latency of the communication defines the delays of the communication the bandwidth or throughput of the communication defines how much data or messages can be transferred within a certain time unit. Multiple factors affect the maximum achievable throughput. As the digital communication is at some point converted to analog transmission there will always be noise sources present. Any practical communication system also has a limited bandwidth available. These two factors lead to Shannon-Hartley theorem [59] providing an upper bound for the achievable throughput.

The desired properties of an ideal HPC interconnect solution therefore are low latency, high bandwidth, and negligible CPU load. In addition to these the interconnect must work in a reliable manner and be constantly available. From the system builder's viewpoint the ideal solution should also be cost efficient while the users of the system are willing to use systems which are convenient to use.

The distribution of different interconnect technologies in the Top500 listing of HPC systems is shown in Figure 8. While the top 5 of the list is dominated by custom interconnect technologies the most common ones on the list are different InfiniBand, 1 Gigabit Ethernet (GigE), and 10 Gigabit Ethernet (10GigE) technologies.

The InfiniBand and most of the other less often used HPC interconnect options are utilizing specialized hardware and communication protocols to scale up to system installations consisting of large number of nodes with high bandwidth between the nodes. The downside of this approach is the requirements for specialized hardware solutions not commonly available and usually the high cost of hardware acquisition. The custom communication protocols have also the downside of not being compatible with TCP/IP stack. Like with InfiniBand some tunneling protocol can be used to carry TCP/IP traffic within the custom communication protocol, but this is likely to render useless the performance gain from the custom protocol.

The InfiniBand architecture is described in more detail in Chapter 4. As studied by Vienne et al. when 1GigE and 10GigE technologies are compared against the InfiniBand

the InfiniBand has an advance both in terms of latency and bandwidth [67]. It is also shown that InfiniBand scales better with the number of concurrent flows than the Ethernet alternative [33].

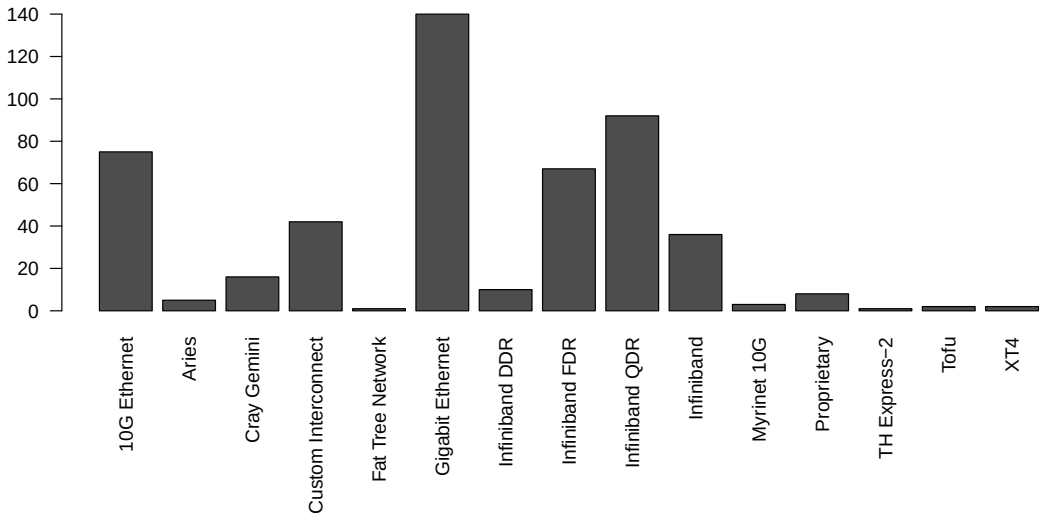


Figure 8: HPC interconnect distribution among HPC systems in the Top500 listing [7]

As the retransmissions of lost or corrupted packets are source of long delays and decreased throughput some of the HPC interconnection technologies are completely removing the need for retransmissions in the first place. If reliable transport is required with the unreliable and connectionless service provided by the Ethernet the upper layer protocol like TCP must be used to provide the reliability and in order delivery of the packets sent. With technologies like InfiniBand the in order delivery and reliable message transport can be enabled on the link level and offloaded to hardware from the operating system kernel.

But as the Top500 list shows many HPC systems are using some Ethernet variant. Especially useful this might be if the system is supposed to run generic applications requiring TCP/IP communication. However designing an Ethernet based interconnect for large number of host nodes requires good planning so that the network topology does not cause limitations for scaling [10]. Basically the idea of providing efficient Ethernet based interconnection is to aggregate links into a fat tree topology as shown in Figure 9. The aggregation means that links closer the root in the topology are having more bandwidth i.e. either more links or higher bandwidth links than the the links connecting to the host nodes.

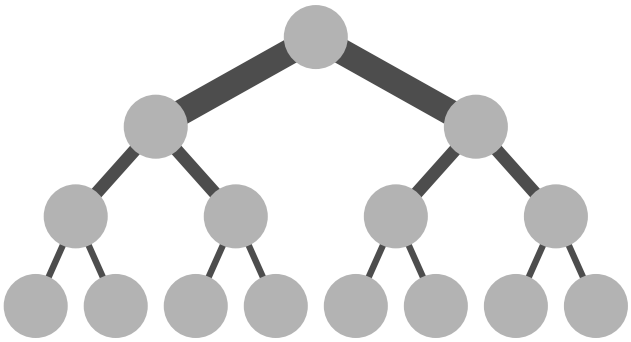


Figure 9: An example of fat tree topology

3.4 Summary

This chapter has discussed about different HPC IaaS technologies and services. At the moment there are many public IaaS providers, some of which are claiming to provide HPC IaaS resources. These services are most likely suitable for certain kind of HPC tasks, but initial tests show that for example low latency requirements are a bit off from what can be achieved with real HPC systems without the cloud service layer. The biggest network related question in HPC system design is whether the interconnect is going to use standard Ethernet or something else like InfiniBand. This design decision is fundamentally a choice between the performance and flexibility of use.

4 InfiniBand Networking

InfiniBand Architecture (IBA) is a network technology used in many HPC systems. The work related to IBA Standard [12] originates to 1999 when the InfiniBand Trade Association (IBTA) defined the first InfiniBand standard for switched fabric I/O architecture for interconnected devices. The driving motivator for a completely new inter-server communication architecture was to improve reliability, availability, scalability, and performance to levels not easily achieved with shared bus I/O architectures [51]. The resulting IBA standard includes over 1500 pages and it's clear that this thesis is unable to cover all the detail of IBA. Hence the focus here is to introduce the differences which have made the IBA one of the most used interconnection type in certain areas of networking.

By looking the ranking of the fastest supercomputer sites in the world [7], one can see that currently 205 sites out of 500 are using InfiniBand as the networking interconnect solution of the system. For a comparison the list has 75 systems using 10 Gb Ethernet interconnection and over 140 systems using 1 Gb Ethernet interconnection.

In Section 4.1 the principles and relevant details of IBA are introduced and these are compared to other I/O architectures used as device interconnects. In Section 4.2 several methods of interfacing the IBA from software are introduced.

4.1 InfiniBand Characteristics

InfiniBand Architecture (IBA) provides low-latency and high-bandwidth interconnect with low processing overheads and focus on transport offload. The link bandwidth ranges from 2Gb/s 1 x Single Data Rate (SDR) up to 300Gb/s with 12 x Enhanced Data Rate (EDR). The end-to-end latency on InfiniBand links are usually up to few microseconds depending on the InfiniBand hardware used and its data rate while the typical switching delay in the middle boxes of the subnet is in the nanoseconds scale. To achieve the latency this small the IBA uses hardware offload to move the I/O processing from the CPU to network hardware. Based on these characteristics it is suitable for system area networking in data centers and HPC clusters.

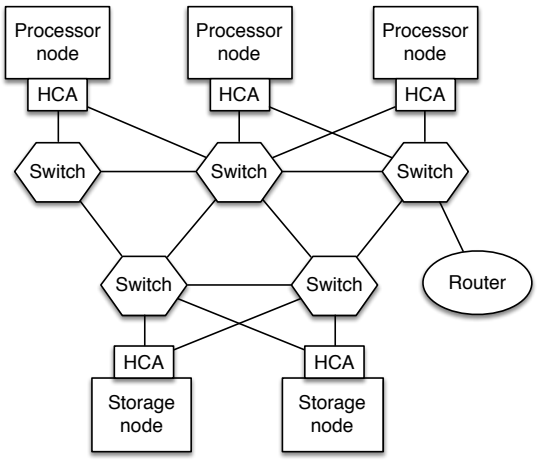


Figure 10: InfiniBand Network

The network topology shown in Figure 10 looks similar to widely used in current Ethernet networks. The switched fabric topology which allows the traffic to be spread among multiple paths connecting the source and target nodes for a communication flow which improves the throughput and provides resilience against network breaks as other paths provide fail-over possibilities.

Nodes in InfiniBand subnets can be interconnected directly or InfiniBand switches can be used. Multiple subnets can be interconnected with InfiniBand routers allowing bigger deployments. Routers can also be used to connect the IB fabric to other networking infrastructures. The layered stack of IBA for end hosts and middleboxes is shown in Figure 11.

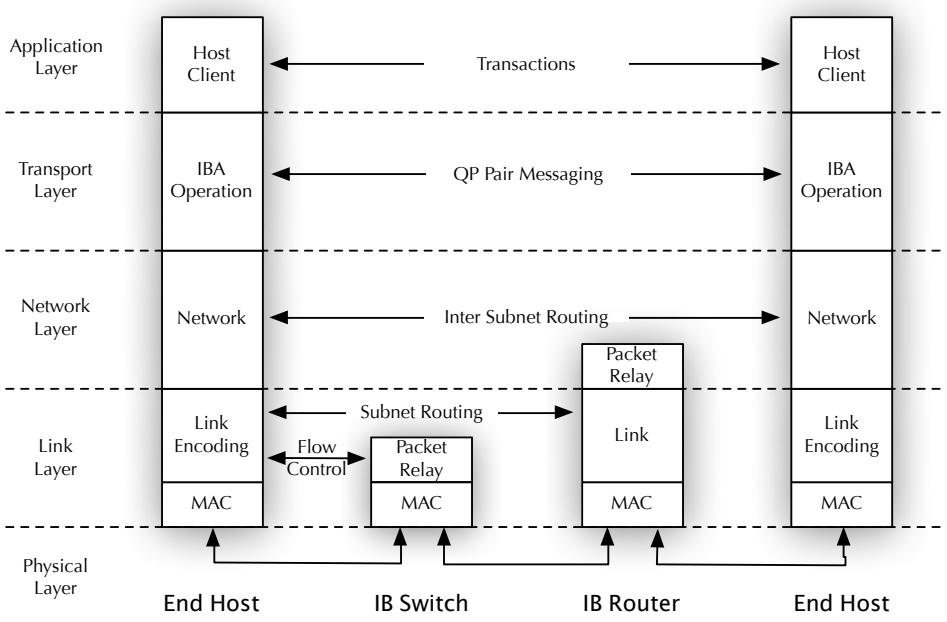


Figure 11: Layers in InfiniBand stack

4.1.1 Operation Semantics

The communication model of IBA is bases on operations which can be inserted into queues which the IB hardware executes. These queues are always created as pairs, one for send and another for receive operations, and the tuple they form is called a queue pair (QP). The operations happening through these QPs are also called as verbs.

The operations used with send queues are defined in the IBA specification [12] and are listed below:

- SEND: Specifies a block of data to be sent from sender's memory space to receiver who places the data into a memory location selected by the receiver
- RDMA: Remote direct memory access operations are operations to remote memory location which include read, write, and atomic operations. With these operations the sender specifies also a memory location the operation is targeted on the receiver's side which is why RDMA operations does not require actions from the receiver.
- MEMORY BINDING: With memory binding operations the memory locations shared with other nodes can be altered and handles to locations can be created which are then used with RDMA operations.

The communication with these operations happens with Work Queue Requests (WQR) which are placed on the send queue. The requests are queue turns to Work Queue Elements (WQE) which are executed by the Host Channel Adapter (HCA) i.e. the IB hardware. One the operation is completed a notification is placed into Completion Queue

(CQ) from where applications can check the status of the operation. The only receive operation is to receive the data which is sent with send operation. With this operation the recipient can copy the data sent by a remote sender to a memory location selected by the recipient.

The IBA connection endpoints are identified by a tuple consisting of a local ID (LID) and QP identifier. The LID is used to specify the host and QP is used to select service used. One can see this as an analogy to a tuples consisting of an IP address and port number used in TCP/IP connections.

4.1.2 Service Types

The IBA specification lists multiple service types for an IBA connection. A connection can either be connection oriented or a datagram style connection. In case of connection oriented service mode the QP used is associated with exactly one another remote QP and the communication flows only between these two QPs. In case of datagram service mode the QP can be used to communicate with multiple other QPs.

Another attribute for service types is whether or not the operations are acknowledged. When the acknowledged mode is used each operation is supposed to return an acknowledgement from the remote QP with either positive or negative acknowledgement. With unreliable service type no acknowledgement is returned, which also means that the operations are not guaranteed to complete and data associated with the operation is not necessarily coherent.

With the service types described above four different kind of combinations can be created listed in Table 2.

Service type	Connection oriented	Acknowledged
Reliable Connection (RC)	yes	yes
Unreliable Connection (UC)	yes	no
Reliable Datagram (RD)	no	yes
Unreliable Datagram (UD)	no	no

Table 2: InfiniBand architecture service types

4.1.3 Congestion and Flow Control

In traditional HPC the low latency is one of the key requirements. In a lossless network without a proper congestion control the network links might become fully utilized leading to effect known as congestion. The congestion increases delays and might lead to packets being dropped because the packet buffers in the devices delivering the packets run out of free space leading to re-transmissions of dropped packets. The congestion therefore could lead to heavy performance degradation.

To avoid the congestion the transmission control protocol (TCP) has several well defined congestion control mechanisms. The most common ones of these, namely TCP New Reno [19], TCP CUBIC [57], and Compound TCP [64] are based on the idea the an observed packet loss indicates congestion situation after which the congestion control mechanisms reacts by reducing the data transmission rate. While no congestion indicators are reporting congestion the transmission rate might be increased in order to reach better utilization of free resources in the network.

To prevent these costly re-transmissions from happening in the first place the InfiniBand interconnect uses a credit based flow control on IB links which guarantees that the link cannot end up being congested. The credit based flow control achieves this so that packets cannot be forwarded to link which is lacking buffer space to receive the packets.

4.1.4 Link Encoding

The link level communication between IB HCAs uses 8b/10b and 64b/66b encoding schemes to enable clock recovery [73]. With the clock recovery the sender does not need to send the clock signal along with the data stream. Upon receiving the data stream the receiver can extract the clock information from the data stream with approximated frequency reference source and phase-locked loop.

With SDR, DDR, and QDR the links using 8b/10b encoding. The use of this encoding means that each 10-bit output symbol carries 8 bits of payload data and 2 bits are used to allow an injective mapping from 8-bit of payload symbols to these 10-bit symbols. With long enough input the mapped symbols have a practical properties such as that the output bit string has the difference between the counts of ones and zeros is no more than two and that maximum length of a substring containing only ones or zeros is five. This also effectively prevents DC bias of the mapped symbols. Other communication protocols using 8b/10b encoding include such technologies like USB 3.0 and Gigabit Ethernet.

Because the 8b/10b encoding adds an extra overhead 25% to input symbol length this encoding scheme is not used with higher data rates namely FDR-10, FDR, and EDR. Instead these data rates use encoding known as 64b/66b encoding which is also used in protocols like 10 Gigabit Ethernet. Compared to 8b/10b encoding the resulting encoded symbols can carry a much higher ratio of input data to encoded data namely the overhead is just 3.125%. The strict guarantee for DC bias prevention is however lost while the practical consequences of this are negligible and can be treated on receiver circuits.

The theoretical maximum transmission speed of the IB link can be calculated when the data rate and used link aggregation factors are known. The signaling rate of a single SDR link is 2.5 gigabits per second (Gb/s) which means that due to 8b/10b encoding the actual payload transmission rate, also known as the goodput, is 2.0 Gb/s. On an IB network the links can also be aggregated so that either four or twelve links are bundled which multiplies the resulting goodput respectively. With Double Data Rate (DDR) the goodput rate is doubled and with Quad Data Rate (QDR) the goodput is four times the SDR i.e. 8 Gb/s with single link aggregation. Again the data rates with FDR-10, FDR, and EDR does

not follow this formula. The signaling rate with Fourteen Data Rate (FDR) is 14.0625 Gb/s and due to encoding the goodput rate is $(1 - \frac{2b}{66b}) * 14.0625\text{Gb/s} = 13.6364\text{Gb/s}$.

4.2 Interfaces for InfiniBand Architecture

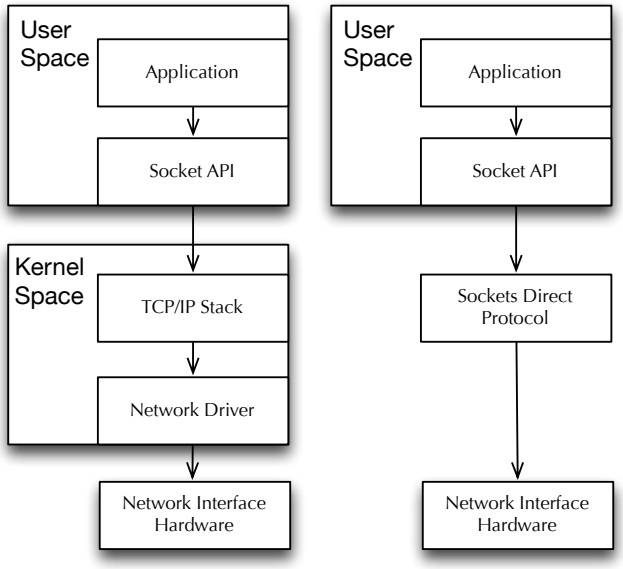
InfiniBand Architecture has no standard API specification but a set of *verbs* used to interact with the transport service [25]. On top of the verbs the Open Fabrics Enterprise Distribution package (OFED) provides multiple upper layer protocols (ULPs) for accessing the underlying InfiniBand (IB) interface. Some of these ULPs are going to be discussed next.

The easiest of running generic IP applications on top of IB networking is to use IP over InfiniBand (IPoIB) protocol. IPoIB encapsulates IP packets and tunnels these packets over the InfiniBand link. The resulting InfiniBand frames are transported by using either the Reliable Connected (RC) [34] delivery mode or the Unreliable Connected (UC) [16] delivery mode.

The IPoIB is a very convenient way of allowing the applications to communicate over IB fabric without any modifications. This means that basically any application using standard TCP/IP communication can work on top of system providing the IPoIB connectivity. However the IPoIB protocol has its drawback. Most notably it has inferior performance characteristics when compared to native IB communication. The reason for this is the processing overhead introduced by the IP protocol. For example with a high throughput IPoIB flow with small packet size will hurt the communication latency due to the IP state management happening in operating system kernel [13]. To prevent the state management processing from forming a bottle neck in the system a proper offload hardware support for I/O is required. Another reason for the overhead introduced by the IPoIB protocol is the presence of IP headers which increases the required message size and reduces the space which can be used to carry the actual application communication payload.

Because the the IPoIB is basically tunneling network layer protocols it cannot handle the link layer protocols. This is especially a problem when virtual local area networks (VLAN) are used. One workaround for this problem is to the Ethernet of IB (EoIB) protocol, which encapsulates the link layer frames in the InfiniBand datagrams. The transformation is completely transparent to applications and allows also the usage of VLANs with the communication happening on top of IB network.

Sockets Direct Protocol (SDP) [12] is a protocol originally developed for TCP communication over IB but later extended to be a general protocol on top of Remote Direct Memory Access (RDMA). SDP offers a way to utilize the RDMA capabilities from applications using BSD stream sockets. In practice with IB this means that with SDP the applications can still use the same BSD socket API, but the underlying implementation is replaced so that the offload capabilities and RDMA semantics of the IB stack are used. As shown in Figure 12 this means bypassing the kernel part of the network stack as the RDMA can be used to read and write data directly from and into the application buffers. Bypassing the TCP/IP stack means also that the communication is utilizing the reliable,



(a) Traditional model (b) Model with SDP

Figure 12: Differences between the traditional network stack model and the stack model when SDP is used

hardware based transport semantics of the IB devices. As the result the applications using TCP sockets on IB enabled hosts can gain major performance boost when compared to applications relying on IPoIB. This is due to the increased latency and more efficient CPU utilization as less CPU cycles are wasted for redundant TCP flow management. The SDP still has dependencies for the IP stack, especially what comes to address resolution. This also means that the system using SDP must have a working IP stack in place, which is replaced by SDP stack only if that is possible.

On Linux based system supporting SDP and with RDMA capabilities to execute a program with SDP the program is started with an additional SDP shared library loaded which will trap the libc socket calls for AF_INET stream socket calls with its own implementation for AF_SDP address family. By default, if the libsdp.so library is loaded and the system has SDP capability, the SDP will be implicitly used to replace the TCP/IP if possible. A developer can also use the address family explicitly. With this approach the application will either use SDP or fail if no SDP capability is available.

Message Passing Interface (MPI) is standard interface for portable and efficient parallel computation [36]. The communication semantics of MPI allow its usage on top of HPC network interconnects like IBA. After almost two decades MPI is still widely used for that purpose, especially in the field of scientific computation.

In addition to the protocols discussed above there exists several others with similar goals of providing efficient communication on top of IB network. Protocols like SCSI RDMA Protocol (SRP) or ISCSI Extensions for RDMA (iSER) are targeted to be used with storage systems attached to IB fabric.

4.3 Summary

This chapter described the characteristics of InfiniBand architecture. InfiniBand uses very different approach compared to more commodity networking technology solutions like Ethernet. As a consequence InfiniBand can provide very low latency and high throughput performance. However the specialized hardware and protocol causes limitations for the InfiniBand usage. To mitigate these limitations some upper layer protocols, like IPoIB, exists to provide interoperability with applications using TCP/IP.

5 Evaluation Methods and Tests

In this chapter the evaluation environment and the test suite are described. The reader is supposed to be familiar with the concepts related to virtualization introduced in Chapter 2.2 and InfiniBand architecture introduced in Chapter 4.

5.1 System Under Test

The system under study here is a HPC IaaS cloud called Pouta, which is operated by CSC - IT Center for Science. The system is physically located in the CSC data center in Kajaani. During the analysis described here the cloud service is going trough final pilot testing before it goes into production. The pilot testing status means that while there are some real usage having an effect on the system performance the system is still in quite a idle state.

The high level presentation of the deployment is given in Figure 13. In this figure three hosts are connected to IB fabric. One of the hosts is using SR-IOV functionality to provide IB connectivity to virtual machine running on the host while the two others are using Linux bridges.

The cloud hardware consists of HP ProLiant SL230s servers referred from now on as hosts or host nodes. Each host node is having Intel Xeon 2.6 GHz E5-2670 processor with 16 cores and 64 gigabytes of main memory. The host nodes are connected to InfiniBand fabric with Mellanox ConnectX3 InfiniBand Host Channel Adapters (HCA) which operate in Fourteen Data Rate (FDR). The IB links are actually bundled so that and promise link signaling rate up to 56 Gb/s with four times link aggregation (4x FDR) while the theoretical maximum goodput for this link is 54.556 Gb/s. On these hosts the large receive offload [28] is turned on to push part of the networking processing to network hardware.

The hardware choice for the host nodes was largely dominated by the another cluster system system sharing the hardware with Pouta. As a consequence this kind of hybrid system can be configured to dynamically move host nodes from cluster system into cloud system and vice versa.

The hosts are running CentOS Linux distribution with Linux kernel 2.6.32. All the virtual machines are executed on hosts, which are dedicated only to run virtual machines on top of the hypervisor called Kernel-based Virtual Machine [48] (KVM). KVM provides full virtualization on x86 hardware and supports both supports both Intel VT and AMD-V technologies. KVM hypervisor is a kernel module which upon loading adds support for kernel to schedule virtual machines as any other processes in the system. Because of this KVM has to implement smaller set of features compared to majority of other hypervisor solutions available. For example the KVM does not need to implement the process scheduler by itself as it can use the Linux kernel process scheduler to implement this functionality.

Initially the network links had maximum transmission unit (MTU) of 1500 bytes which

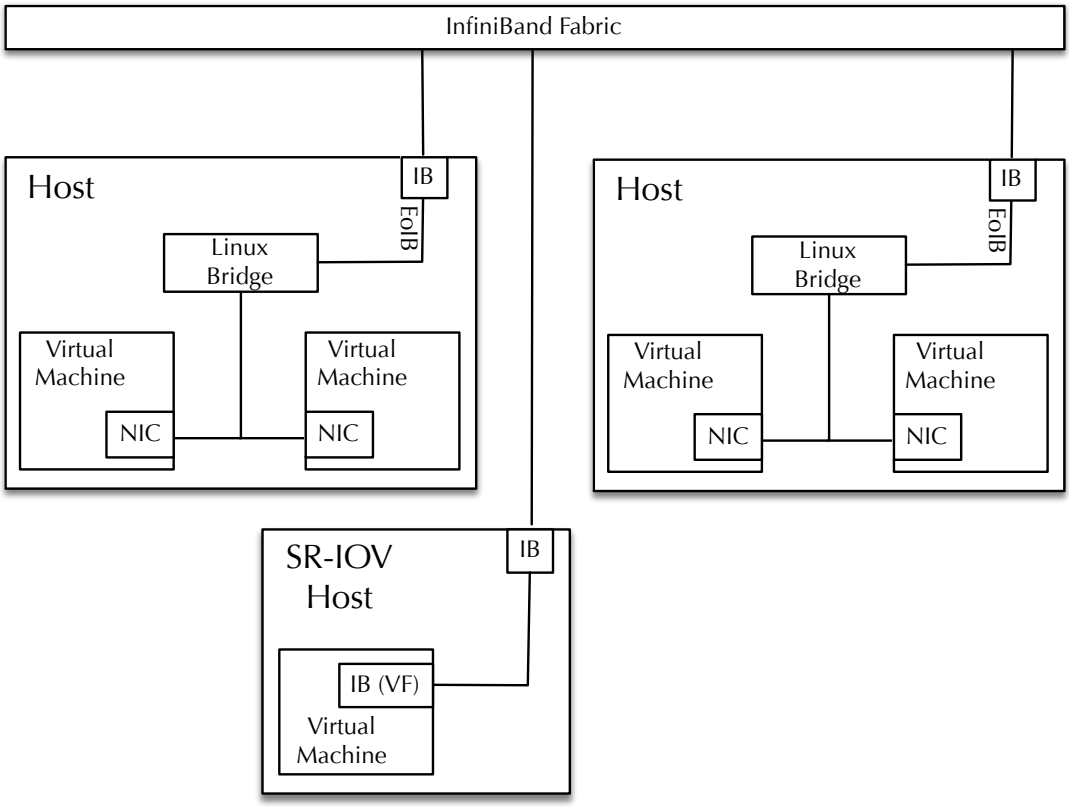


Figure 13: Conceptual view of a small part of Pouta system

was later during the evaluation experiments updated to 4034 bytes to achieve better bandwidth usage. The MTU limit of 4034 bytes is due to Ethernet over InfiniBand (EoIB) protocol and VLAN header insertion overhead [40] which are used on compute hosts.

The official support for Single Root I/O Virtualization (SR-IOV) described in Section 2.2 is added to official Mellanox Open Fabrics Enterprise Distribution (Mellanox OFED) versions since revision 2.0-3.0.0 released in 2013. The InfiniBand performance with SR-IOV enabled host was however tested prior to this evaluation with firmware version still in development. The results from these tests indicated that the achieved throughput were known to be poor for SR-IOV enabled hosts and virtual machines running on these hosts.

To use SR-IOV functionality with InfiniBand network interface card the system Basic Input/Output System (BIOS) must be configured with SR-IOV and Intel Virtualization technology must be enabled. After this the kernel must be booted with kernel parameters shown in Listing 1. After this the kernel can support SR-IOV technology and map the physical devices into a set of virtual function devices.

Listing 1: Kernel parameters to boot a SR-IOV enabled kernel

```
intel_iommu=true iommu=pt
```

Next a firmware support for SR-IOV is required to be burnt on the device. This firmware can be installed with tools provided with the OFED distribution. After a successful installation the system should have the original PCI device and multiple virtual function devices of the original device as shown in the lspci output of the system in Listing 2.

Listing 2: The output of lspci command after the SR-IOV capable InfiniBand device namely Mellanox ConnectX-3 is installed

```
# lspci  | grep -i ConnectX
06:00.0 Network controller: Mellanox Technologies MT27500 Family [
   ConnectX-3]
06:00.1 Network controller: Mellanox Technologies MT27500 Family [
   ConnectX-3 Virtual Function]
06:00.2 Network controller: Mellanox Technologies MT27500 Family [
   ConnectX-3 Virtual Function]
...
```

Configuring SR-IOV enabled device to be used with virtual machines can be tricky. Without these options described above the hypervisor might be unable to assign virtual function device to virtual machines or the interrupt handling might require tremendous CPU utilization to process the network traffic. This was noticed during the evaluations described in this thesis and the issue was reported to Mellanox Technologies which later updated their installation documentation instructing the usage of iommu=pt parameters setting.

Multiple other configuration options for performance tuning are documented in the Mellanox guide for IB performance tuning [41]. The biggest performance gain for these single flow tests was from enabling the large receive offload.

The tests using SR-IOV are executed on the same hardware which is available on Pouta cloud but the system and the network are separate systems. In these tests the network i.e. the InfiniBand fabric is having minimal background utilization compared to Pouta cloud where noise sources are present due to the pilot testing.

OpenStack Neutron is the networking component of OpenStack [5] cloud middleware used in Pouta cloud installation. The component is responsible for creating the virtual networking infrastructure for virtual machines to use. This includes e.g. procedures for assigning addresses for virtual machines and managing the subnet allocation and access in the system. While Neutron is mostly acting as a controller the underneath implementation it uses a set of widely used networking tools and technologies. One of these technologies is the Linux bridge implementation, which acts as a software switch connecting multiple local area networks. The Linux bridging is known to have some processing overhead due to switching state management and some alternatives for this have been evaluated recently [66]. Neutron creates a special virtual TAP device interface for each virtual machine which is connected to bridge. Multiple virtual machines on a single host can then be attached to single bridge which happens e.g. if the virtual machines belonging to the same same customer are scheduled to run on the same host.

A part of the network setup used in Pouta system is modeled in Figure 13. In the figure there are three physical host machines connected to IB fabric, two of which are both having two virtual machines running on them are participating to standard network setup configured by Neutron. The third one is a host with SR-IOV configuration and with one virtual machine running on it. Currently Neutron has no off-the-shelf support for SR-IOV devices. Also in a production environment of a public cloud like Pouta the security considerations are still open for SR-IOV devices.

5.2 Test Suite Description

In this evaluation a test runner is implemented to handle the scheduled execution of test suite. The test suite runner is a client-server application implemented in Python programming language [53]. The test suite can be downloaded from its GitHub repository[3].

All the tests are active network tests which means that they generate extra traffic in the network and the measurements are taken from that traffic. Most of these tests require both a server and a client applications and the results are reported either on the server or the client end.

The latency on links between virtual machines and hosts using IPoIB is tested with ping application from iputils [4] suite. For each measurement a set of fifty samples is taken with 0.25 second interval. The throughput on these links is measured with iperf [3] tool. The effect of different message sizes is tested with ØMQ (ZeroMQ) messaging library [8] by sending one hundred messages of a specific size from sender to receiver.

[3] https://github.com/hhamalai/ducking-octo-wallhack

On hosts and virtual machines that have direct access to the InfiniBand interface the link characteristics are also tested with applications from the OFED *perftest* package to measure network RDMA latency and throughput for read and write operations. These test applications are using RDMA communication instead of IPoIB protocol and the results are therefore expected to have an increased performance. For the hosts this should match the native InfiniBand performance. For the virtual machines using SR-IOV the performance is having an overhead due to virtualization and due to SR-IOV virtual functions used.

Due to the experimental state of Mellanox OFED the Socket Direct Protocol (SDP) could not be tested thoroughly in this evaluation. For the evaluation discussed in this thesis the SDP could be used for communication between hosts or virtual machines using the SR-IOV technology or a mix of these. As discussed in chapter 4 SDP might provide some performance gain for IPoIB flows when TCP is used.

The test runner is a client-server application where the client process has the test suite description of all tests supposed to be executed. As most of the tests are having both the server and the client endpoints the client process is first supposed to connect to server process and to tell the server process which test to execute. Upon receiving this message from client the server process forks the new test process requested by the client and confirms to client that the test is running. The client process can now execute its part of the test which is supposed to be terminated within a finite period of time. When the client test process is finished the client requests the server process to terminate the server test process, after which the server can receive new test request from the same or another test runner client.

As the suite is a collection of well-known test applications they have different formats and styles to report the results. Some applications report the results on the server process while others report the results on client side process or both. The test runner collects the output from both, the server and the client processes, and stores them for later processing with the command used to execute the test so that it is possible later to return and verify the test runs.

The test runner runs the suite once in a specified interval with default being one hour. This means that for a single test there are multiple executions of it producing multiple results for a single test case.

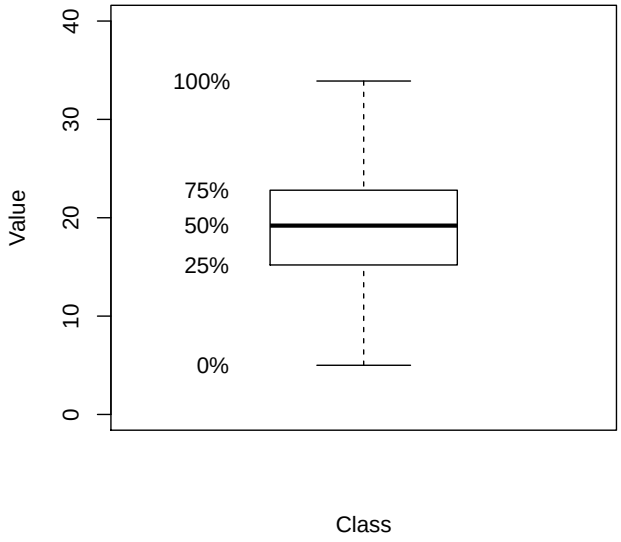


Figure 14: An example of statistics visualized with a box plot

The results from these tests are processed with Bash [22] shell scripts and R toolkit [54]. When applicable, box plots are used to visualize measured statistics, which provide a convenient ways to present descriptive statistics of a data set. An artificial example of such plot is given in Figure 14. The lowest *whisker* of a box plot depicts the minimum value in the data set i.e. 0% of the samples have a value less than this. The highest whisker depicts the maximum value. The box in the middle has its lower and upper edges so that the second and the third quartile sets are inside the box. The line inside the box marks the median value in the data set.

5.3 Initial IB RDMA results

The first set of results describes the initial characteristics of the system. To begin with the native InfiniBand (IB) performance is tested with measurement utilities provided by the Mellanox OFED distribution. These utilities report the latency and bandwidth for the RDMA read and write operations.

The results for IB RDMA read and write latency tests are shown in Figure 15 for different messages sizes. The results show that for small message sizes the latency is almost a constant, but starts to grow exponentially when the message size is increased above 512 bytes. The latency for write operations is less than a microsecond for writes for small message size and less than two microseconds for read operations returning the data. The difference is saturated when message sizes are increased.

Similarly the results for IB RDMA bandwidth tests are shown in Figure 16 for different

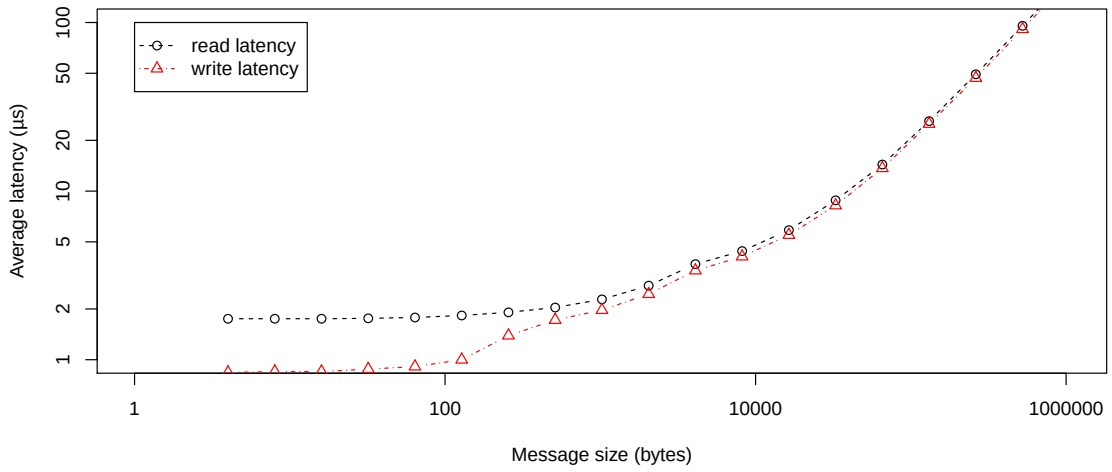


Figure 15: Native IB read/write latency for different message sizes (message size = 2^n bytes). Logarithmic axes.

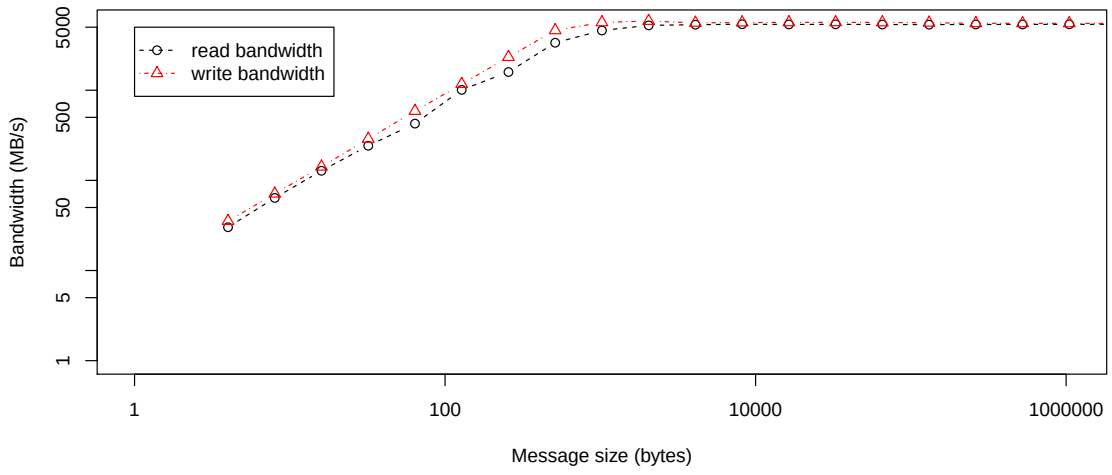


Figure 16: Native IB read/write bandwidth for different message sizes (message size = 2^n bytes). Logarithmic axes.

message sizes. From these results it is visible that the throughput is increased with bigger message size until the capacity of the link is exceeded around message size of 512 - 1024 bytes. When compared to Figure 15 this is also the point where the latency starts to grow, which verifies the claim of link being fully utilized.

5.4 Initial IPoIB results

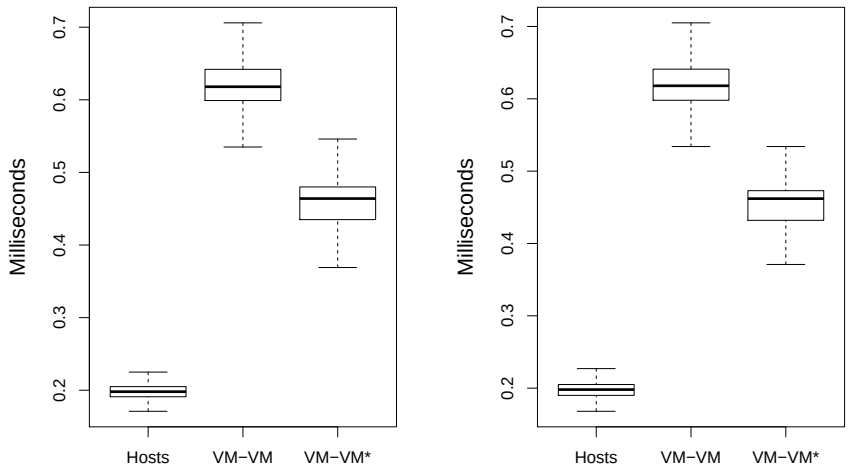
Next the IB networking performance is tested with an encapsulation protocol IP over InfiniBand (IPoIB) without any advanced virtualized I/O support available in the system.

The hosts column in the Figure 17a shows the IPoIB latency between two separated hypervisor hosts connected with InfiniBand fabric. This sets a practical baseline for network latency without any virtualization. The second column in the figure shows the IPoIB latency between two virtual machines running on the two hypervisors hosts. The last column also shows the statistics for latency between two virtual machines, but these virtual machines are now located on the same hypervisor host. When both virtual machines are located on the same host machine, the packets are not assumed to leave from the hosts and the communication is expected to perform more efficiently.

From the data shown in Figure 17a it is clearly visible that the latency is very different compared to what native InfiniBand with Full Data Rate promises which should be around few microseconds. The reason for this is that the connection is now using IPoIB, an encapsulation protocol which carries the generic IP data traffic in IB data frames. The encapsulation protocol adds more overhead for the payload of the packets and for the required traffic processing and state management. As a result the efficient communication seen in the results with IB RDMA is lost.

Initially the system installation had the default maximum transmission unit (MTU) of 1500 bytes on IPoIB links. While this value is common in local area networks using Ethernet this value is not optimal in our system running on top of InfiniBand. Smaller MTU size means that the ratio of packet headers to payload is greater meaning that less bits within a given time frame can carry useful data. Also more packets means that more processing is required from the end points of communication as well as from the middleboxes participating the transfer. Bigger packets means that the delays in the packet transmission and delivery are going to increase. Also bigger packets are more likely to be affected by bit errors introduced by an unreliable link. However in our system the IB links are delivering the packets in reliable manner and the delays on the network are small compared to delays caused by IPoIB protocol processing on end hosts.

The MTU size of 1500 bytes was next increased to 4034 bytes maximum possible MTU value for IPoIB link as described in Section 5.1. Increasing the MTU should have little effect for latency in networks where there are plenty of capacity available. This is true which can be seen in Figures 17a and 17b showing the latency for 1500 bytes MTU and 4034 bytes MTU.



(a) MTU 1500 (b) MTU 4034

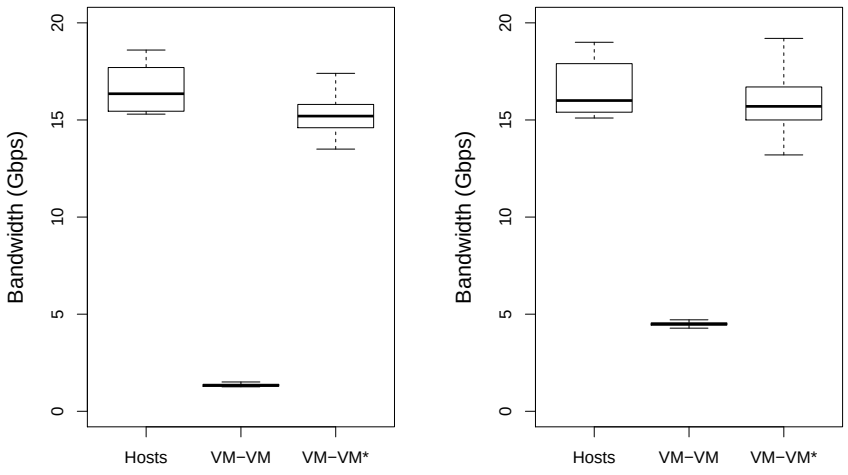
Figure 17: IPoIB latency measurements between the hypervisor hosts, between virtual machines on two separate hosts, and between two virtual machines on the same host (*).

Increasing the MTU size comes visible when the bandwidth usage between two end hosts is analyzed. In Figures 18a and 18b the network link throughput is shown for the three scenarios. While the measured maximum throughput values are higher in all of the scenarios only the scenario of communication with two virtual machines on separate hosts shows clear improvement. For the scenario where the virtual machines are located on the same host there is no clear improvement. The reason for this is that actually packets from a virtual machine to another are never actually leaving the host machine but passed through the software switch in the hypervisor. One reason for the improvement in the scenario with two virtual machines on two separate hosts is the decreased amount of processing required for packet headers due to bigger payload sizes.

While the increased MTU clearly provides higher throughput between virtual machines on the separate hosts compared to the scenario where they are located on the same hosts, the native IB link between the two hosts seems to perform equally good with the virtual machines on the same host scenario.

Based on these results it seems that the IPoIB link provides consistent latency between to hosts which is even smaller than the latency between two virtual machines on the same host interconnected by a software switch. However the throughput is almost equal over the IPoIB link between hosts and over the software switch between two virtual machines on the same host.

The outcome of these results is twofold. The performance in terms of IPoIB latency seems to be dominated by the fact whether or not virtualization is used. For the throughput what



(a) MTU 1500 (b) MTU 4034

Figure 18: IPoIB bandwidth measurements between the hypervisor hosts, between virtual machines on two separate hosts, and between two virtual machines on the same host.

matters more is whether or not the virtualized communication happens over the physical IB link.

As seen in results, using the maximum MTU does not hurt the latency in any scenarios tested. For highest possible network performance the system installation should check whether this assertion applies. The effect of MTU size variations are studied previously and these results are known to apply [21, 45]. For the systems where the link bandwidth is limited the use of big frames i.e. jumbo frames might introduce adverse effects like noticeable latency increase.

5.5 SR-IOV enabled network interfaces

The results in this section are taken from the evaluated system when SR-IOV capability is enabled. As described in Section 2.2 this enables the system to offload the I/O processing of the virtual machine into special hardware so that the VMM does not need to handle the I/O processing and transfer between the VMM memory space and the memory space of the virtual machine, which reduces the I/O virtualization overhead.

The InfiniBand performance with SR-IOV enabled host was tested prior to this evaluation and the results for achieved throughput were known to be rather poor for SR-IOV enabled hosts and virtual machines running on these hosts. Before setting the IOMMU pass-through option discussed in Section 5 the throughput performance between virtual machines using SR-IOV was few hundred megabytes per second and the jitter present in

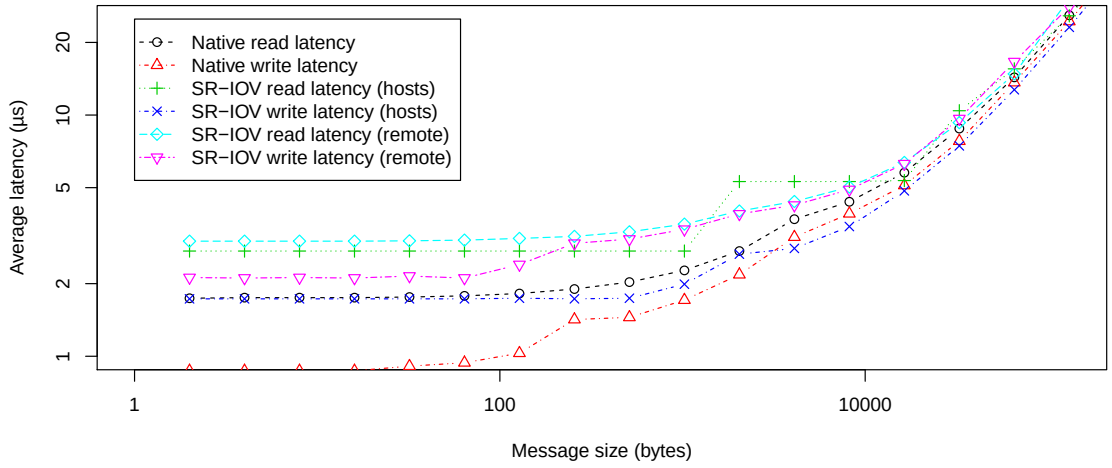


Figure 19: InfiniBand read/write latency for different message sizes between InfiniBand hosts without SR-IOV and between virtual machines using SR-IOV on the same host machines (hosts) and on separate hosts (remote). Logarithmic axes.

latency. The high CPU load of the system under heavy network I/O were clearly telling that the system was misconfigured.

After the discussion with the hardware vendor the fix for this was published and the system performance went up to reasonable performance characteristics shown in Figure 19 for latency and in Figure 20 for throughput. The communication with SR-IOV virtual functions for virtual machines is having a latency overhead around one microsecond and the throughput is consistently slightly less compared to communication between physical hosts. These results however show that the communication with SR-IOV virtual functions for virtual machines can scale up and the performance hit due to the virtualization is negligible and close to the level which is achievable without SR-IOV.

The Figure 21 shows results from IPoIB latency tests while the throughput performances are shown in Figure 22. The results show that enabling SR-IOV can improve the network performance of the virtual machines close to the bare metal characteristics. The interesting thing to note is the slightly inferior performance of communication when the virtual machines as located within the same hypervisor host compared to scenario when these virtual machines are deployed on two different hypervisors on separate hosts interconnected with a IB link.

To get an idea of network characteristics from the real application point of view the performance tests from ØMQ library were run with different message sizes with and the results are shown in Figure 23. For each case the latency stays stable until a message size exceeds the threshold of half megabyte after which the latency starts to ramp up.

In the host-to-host scenario of the ØMQ tests the latency is always below the values from

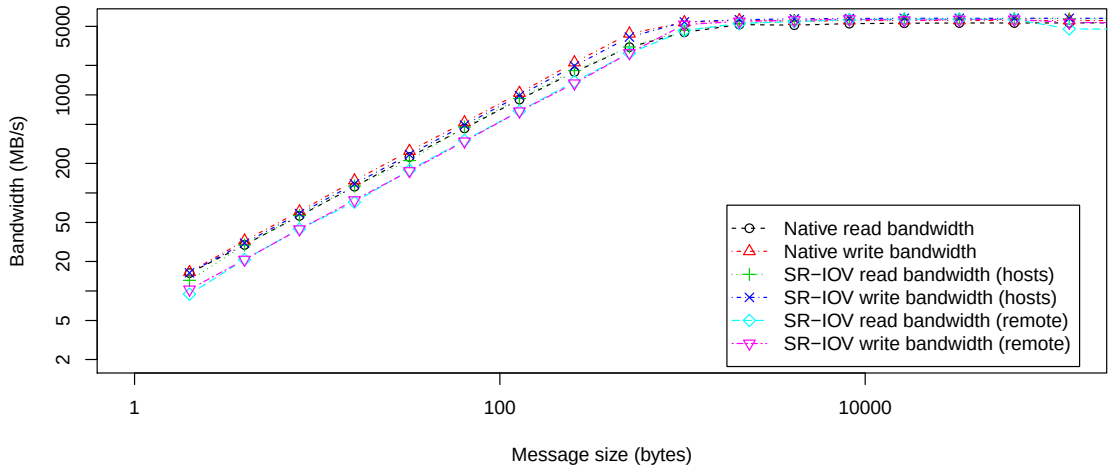


Figure 20: InfiniBand read/write bandwidth for different message sizes between InfiniBand hosts without SR-IOV and between virtual machines using SR-IOV on the same host machines (hosts) and on separate hosts (remote). Logarithmic axes.

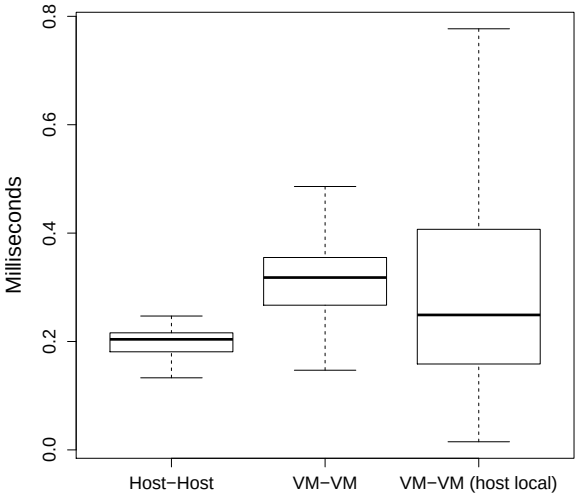


Figure 21: SR-IOV enabled IPoIB latency measurements

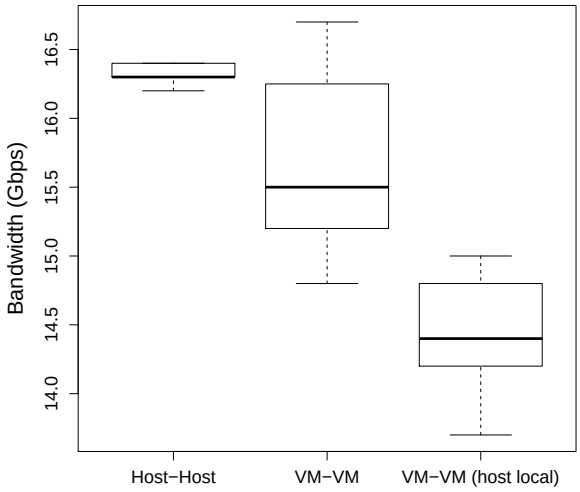


Figure 22: SR-IOV enabled IPoIB bandwidth measurements

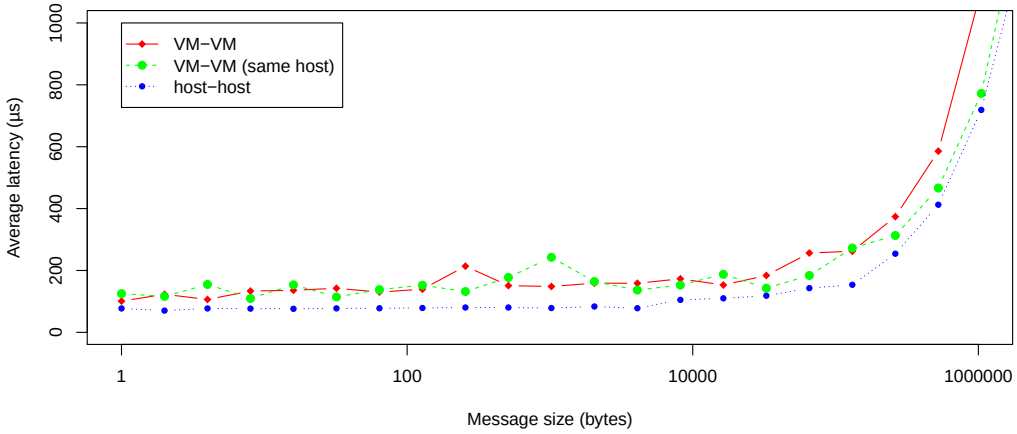


Figure 23: SR-IOV enabled ØMQ latency measurements with different message sizes

scenarios using virtualization. There are no clear distinction between the two scenarios using virtualization until the latency starts to increase more rapidly after which the scenario where the virtual machines are running on the same hypervisor provides constantly better service in terms of latency.

Comparing the ØMQ latency to latency reported by ping in SR-IOV enabled environment (figures 23 and 21) the results show no major difference with reasonable message sizes. This shows that the real-world application performance should be able to achieve stable latency for its communication. Comparing the ØMQ latency to RDMA read and write latency (figures 23 and 19) the difference in clear. RDMA communication is roughly hundred times faster than generic communication over IPoIB used by ØMQ tests.

5.6 Summary

The results shown in this chapter confirm the known assumption of IPoIB performing with higher latency and lower bandwidth than what is achievable with the native InfiniBand RDMA access. This is due to the processing overhead of IPoIB which is suitable for generic communication between applications build to run with standard TCP/IP stack. On the other hand the communication with RDMA semantics has lower latency and higher throughput, but the RDMA can only be utilized with RDMA aware applications. In the evaluated system the IPoIB performs roughly one hundred times slower link latency and the maximum bandwidth is one third of the RDMA performance.

The performance gain from high performance networking infrastructure like InfiniBand can be utilized by general client applications with an added overhead. In this particular scenario at least two sources of overhead exists. The first one is due to IPoIB protocol, which adds an overhead due to encapsulated protocol requiring more message headers to be transferred over the wire and more processing and accounting on end hosts. The second one is due to virtualization. The hypervisor is required to control the virtual machine communication flows and route messages according to this.

With virtualization without advanced virtualized I/O support the interrupt handling under heavy network I/O will cause a CPU load in the system which might even limit I/O processing and hence also the maximum network traffic bandwidth. Hardware assisted technology such as SR-IOV can be used to map the physical device into multiple virtual devices which can be associated with virtual machines on the host. These virtual machines can then directly use these virtual devices with standard PCI access to interact with the device, which will remove the need for extra interrupt handling and hence improve the network I/O performance.

The results from the evaluation scenarios using IPoIB and SR-IOV show only a minor latency overhead with virtualization. The throughput is interestingly a slower for virtual machines running on the same hypervisor host than in the scenario without SR-IOV. However the throughput loss can be seen negligible and could be assumed to be improved with more matured firmware. For the remote VM scenario the throughput is greatly improved compared to scenarios without SR-IOV.

The results from the SR-IOV scenarios with RDMA access show stable and consistent performance characteristics with and without SR-IOV. The SR-IOV results shows a negligible overhead for latency and throughput. This means that the variation in IPoIB scenarios are likely to caused by the tunneling protocol itself rather than due to SR-IOV implementation.

6 Conclusions

This thesis contains a network performance evaluation of IaaS cloud installation on top of IB networking. The evaluation scenarios study the effect of virtualization for network performance. One of the evaluation scenarios contains an analysis of the network performance where SR-IOV is used with IB devices. The results from this scenario shows a drastic improvement in both the network latency and bandwidth.

The evaluation results show that the major reasons for performance losses are the encapsulation protocols for generic communication like IPoIB and the virtualization itself without proper I/O support. To make the communication more efficient the generic communication protocols can sometimes be changed with technologies like RDMA. The loss of generality however going to limit the possible use cases. Sometimes protocols like SDP can be used to bypass parts of the protocol which are used to implement features already provided by the underlying protocol. Such tinkering can however only be used in certain use cases and is not going to completely remove the overhead introduced by the encapsulation.

To combat the performance problems due to virtualization there currently exists several hardware assisted solutions for virtualized I/O. In this thesis SR-IOV was used to assign virtual functions of IB HCA directly with virtual machines. The results from the SR-IOV scenario show that SR-IOV introduces only a negligible overhead for the communication latency and maximum throughput for the communication with virtual machines. In all practical cases the performance with SR-IOV outperforms the cases where virtualization is used without SR-IOV.

In this thesis only single flow scenarios were tested between two endpoints. To further study the network performance characteristics between the virtual machines multiple flows scenarios should also be studied. Previous studies show that InfiniBand should scale reasonable well with multiple flows.

The direct IB access with SR-IOV offers not only more performance for networking but also allows native IB interface to be used from virtual machines. Still widens the set of communication protocols available for virtual machines and increases the number of use cases for HPC IaaS cloud. One can also argue that without SR-IOV enabled virtual machines the definition of HPC IaaS cloud is a bit vague as the network performance is no way near the native IB performance.

The results presented in this thesis indicate that SR-IOV should be used in virtualized cloud environment whenever possible in order to obtain the best performance. However some shortcomings currently exists which are why the usage of SR-IOV might not always be feasible. Most crucial problem at the moment being the lack of support for SR-IOV enabled network access on the public cloud middleware applications.

The SR-IOV technology for IB is still very young. Several computing solutions utilization virtualization, such as the OpenStack cloud middleware, are integrating these technologies into their environments, but currently there are no open source and publicly available

solutions to combine the SR-IOV enhanced networking with the cloud middleware while commercial products using the SR-IOV have just become available for other network technologies like the SR-IOV enabled access to 10 Gigabit Ethernet on the Amazon's EC2 platform.

References

[1] Folding @ Home. http://folding.stanford.edu/home/, May 2014.

[2] Heroku Cloud Application Platform. https://www.heroku.com/, April 2014.

[3] Iperf. http://iperf.sourceforge.net/, May 2014.

[4] iputils. http://www.skbuff.net/iputils/, May 2014.

[5] OpenStack Open Source Cloud Computing Software. http://www.openstack.org/, May 2014.

[6] Salesforce.com. http://www.salesforce.com, April 2014.

[7] Top500, TOP 500 Supercomputer Sites. http://www.top500.org/lists/2013/06/, 2014.

[8] ØMQ - Code Connected. http://zeromq.org/, May 2014.

[9] Dennis Abts and John Kim. High performance datacenter networks: Architectures, algorithms, and opportunities. *Synthesis Lectures on Computer Architecture*, 6(1):1–115, 2011.

[10] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[11] Amazon Web Services, Inc. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/, December 2013.

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

[13] Pavan Balaji, Sundeep Narravula, Karthikeyan Vaidyanathan, Savitha Krishnamoorthy, Jiesheng Wu, and Dhabaleswar K Panda. Sockets direct protocol over infiniband in clusters: is it beneficial? In *Performance Analysis of Systems and Software, 2004 IEEE International Symposium on-ISPASS*, pages 28–35. IEEE, 2004.

[14] Brandon Buttler. Just how much bigger AWS is compared its next competitor may surprise you. http://www.networkworld.com/community/blog/just-how-much-bigger-aws-compared-its-next-competitor-may-surprise-you, 2014.

[15] Shefali Chinni and Radhakrishna Hiremane. Virtual machine device queues. *Intel Corp. White Paper*, 2007.

[16] J. Chu and V. Kashyap. Transmission of IP over InfiniBand (IPoIB). RFC 4391 (Proposed Standard), April 2006.

[17] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[18] Yaozu Dong, Zhao Yu, and Greg Rose. SR-IOV Networking in Xen: Architecture, Design and Implementation. In *Proceedings of the First Conference on I/O Virtualization*, 2008.

[19] Sally Floyd, Tom Henderson, and Andrei Gurtov. The newreno modification to tcp's fast recovery algorithm. Technical report, RFC 2582, April, 1999.

[20] J Fontán, T Vázquez, L Gonzalez, Ruben S Montero, and IM Llorente. Opennebula: The open source virtual machine manager for cluster computing. In *Open Source Grid and Cluster Software Conference, San Francisco, CA, USA*, 2008.

[21] Annie P Foong, Thomas R Huff, Herbert H Hum, Jaidev P Patwardhan, and Greg J Regnier. Tcp performance re-visited. In *Performance Analysis of Systems and Software, 2003. ISPASS. 2003 IEEE International Symposium on*, pages 70–79. IEEE, 2003.

[22] Free Software Foundation Inc. GNU Bash. `https://www.gnu.org/software/bash/`, 2014.

[23] Glenn Lockwood. High-Performance Virtualization: SR-IOV and Amazon's C3 Instances. `http://glennklockwood.blogspot.fi/2013/12/high-performance-virtualization-sr-iov.html`, 2014.

[24] Robert P Goldbert. Survey of Virtual Machine Re- search. *Computer*, 7(6):34–45, 1974.

[25] Paul Grun. Introduction to InfiniBand ™ for End Users. Technical report, 2010.

[26] HaiBing Guan, YaoZu Dong, Kun Tian, and Jian Li. SR-IOV Based Network Interrupt-Free Virtualization with Event Based Polling. *IEEE Journal on Selected Areas in Communications*, 31(12):2596–2609, December 2013.

[27] Abhishek Gupta and Dejan Milojicic. Evaluation of HPC Applications on Cloud. *2011 Sixth Open Cirrus Summit*, pages 22–26, October 2011.

[28] Takayuki Hatori and Hitoshi Oi. Implementation and analysis of large receive offload in a virtualized system. *Proceedings of the Virtualization Performance: Analysis, Characterization, and Tools (VPACT'08)*, 2008.

[29] Qiming He, Shujia Zhou, Ben Kobler, Dan Duffy, and Tom McGlynn. Case study for running HPC applications in public clouds. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, page 395, 2010.

[30] Ignacio M. Llorente. OpenNebula vs. OpenStack: User Needs vs. Vendor Driven. `http://opennebula.org/opennebula-vs-openstack-user-needs-vs-vendor-driven/`, 2014.

[31] Saeed Iqbal, Rinku Gupta, and Yung-Chin Fang. Job Scheduling in HPC Clusters. *Dell Power Solutions*, (February):133–136, 2005.

[32] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 159–168, November 2010.

[33] Lars E Jonsson and William R Magro. Comparative performance of infiniband architecture and gigabit ethernet interconnects on intel® itanium® 2 microarchitecture-based clusters. Technical report, Technical report, Intel Americas, Inc, 2003.

[34] V. Kashyap. IP over InfiniBand: Connected Mode. RFC 4755 (Proposed Standard), December 2006.

[35] Linux Kernel Organization. The Linux Kernel Archives. `https://www.kernel.org/`, 2014.

[36] Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Computing*, 22:789–828, 1996.

[37] Clifford Lynch. Big data: How do your data grow? *Nature*, 455(7209):28–29, 2008.

[38] Piyush Mehrotra, Jahed Djomehri, Steve Heistand, Robert Hood, Haoqiang Jin, Arthur Lazanoff, Subhash Saini, and Rupak Biswas. Performance evaluation of Amazon EC2 for NASA HPC applications. *Proceedings of the 3rd workshop on Scientific Cloud Computing Date - ScienceCloud '12*, page 41, 2012.

[39] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing, Recommendations of the National Institute of Standards and Technolog. *National Institute of Standards and Technology*, 2011.

[40] Mellanox Technologies. Mellanox OFED for Linux - User Manual. `http://www.mellanox.com/related-docs/prod_software/Mellanox_OFED_Linux_User_Manual_v2.0-3.0.0.pdf`, December 2013.

[41] Mellanox Technologies. Performance Tuning Guidelines for Mellanox Network Adapters (Revision 1.11). `http://www.mellanox.com/related-docs/prod_software/Performance_Tuning_Guide_for_Mellanox_Network_Adapters.pdf`, 2014.

[42] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[43] Michael Larabel. Intel Haswell Linux Virtualization: KVM vs. Xen vs. VirtualBox. `http://www.phoronix.com/scan.php?page=article&item=intel_haswell_virtualization&num=1`, July 2013.

[44] Cyriel Minkenberg, F Abel, and P Muller. Designing a Crossbar Scheduler for HPC Applications. *Micro, IEEE*, 26(3):58–71, 2006.

[45] David Murray, Terry Koziniec, Kevin Lee, and Michael Dixon. Large mtus and internet performance. In *High Performance Switching and Routing (HPSR), 2012 IEEE 13th International Conference on*, pages 82–87. IEEE, 2012.

[46] Jeffrey Napper and Paolo Bientinesi. Can cloud computing reach the top500? In *Proceedings of the Combined Workshops on UnConventional High Performance Computing Workshop Plus Memory Access Workshop*, UCHPC-MAW '09, pages 17–20, New York, NY, USA, 2009. ACM.

[47] Pär Olsson, Janne Wallenius, Christophe Domain, Kai Nordlund, and Lorenzo Malerba. Two-band modeling of α-prime phase formation in Fe-Cr. *Physical Review B*, 72(21):214119, 2005.

[48] Open Virtualization Alliance (OVA). Kernel-based Virtual Machine website: Status - KVM. `http://www.linux-kvm.org/page/Status`, 2013.

[49] David Ott. Understanding VT-d: Intel Virtualization Technology for Directed I/O. `http://software.intel.com/en-us/blogs/2009/06/25/understanding-vt-d-intel-virtualization-technology-for-directed-io`, 2009.

[50] PCI-SIG. Address Translation Services 1.1 specification. `http://www.pcisig.com/specifications/iov/ats`.

[51] Gregory F Pfister. An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O*, 42:617–632, 2001.

[52] PCISIGSR Primer. An Introduction to SR-IOV Technology. *Intel LAN Access Division, Revision*, 2, 2008.

[53] Python Software Foundation. Python Programming Language – Official Website. `http://python.org/`, May 2014.

[54] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.

[55] Red Hat Inc. KVM – Kernel based virtual machine. `http://www.redhat.com/rhecm/rest-rhecm/jcr/repository/collaboration/jcr:system/jcr:versionStorage/5e7884ed7f00000102c317385572f1b1/1/jcr:frozenNode/rh:pdfFile.pdf`, 2009.

[56] J J Rehr, J P Gardner, M Prange, L Svec, and F Vila. Scientific Computing in the Cloud. *Computing in Science & Engineering*, 3(12):34–43, 2010.

[57] Injong Rhee, Lisong Xu, and Sangtae Ha. Cubic for fast long-distance networks. 2008.

[58] Mendel Rosenblum and Tal Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *Computer*, (May):39–47, 2005.

[59] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[60] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[61] Sayantan Sur, Matthew J. Koop, and Dhabaleswar K. Panda. High-performance and scalable mpi over infiniband with reduced memory usage: An in-depth performance analysis. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[62] Teemu Takaluoma, Tero Säkkinen, Tom Bajorek, Risto S Laitinen, Bernt Krebs, Heidi Conrad, and Olaf Conrad. Computational study of hybrid chalcogenoborate anions. *Journal of molecular structure: Theochem*, 821(1):1–8, 2007.

[63] Tuomas Tallinen, J.A. Åström, P Kekäläinen, and Jussi Timonen. Mechanical and thermal stability of adhesive membranes with nonzero bending rigidity. *Physical review letters*, 105(2):26103, 2010.

[64] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. A compound tcp approach for high-speed and long distance networks. 2006.

[65] Radu Tudoran, Alexandru Costan, Gabriel Antoniu, and Luc Bougé. A performance evaluation of Azure and Nimbus clouds for scientific applications. *Proceedings of the 2nd International Workshop on Cloud Computing Platforms - CloudCP '12*, pages 1–6, 2012.

[66] N. Varis and J. Manner. Performance of a software switch. In *High Performance Switching and Routing (HPSR), 2011 IEEE 12th International Conference on*, pages 256–263, July 2011.

[67] Jerome Vienne, Jitong Chen, Md Wasi-ur Rahman, Nusrat S Islam, Hari Subramoni, and Dhabaleswar K Panda. Performance analysis and evaluation of infiniband fdr and 40gige roce on hpc and cloud computing systems. In *High-Performance Interconnects (HOTI), 2012 IEEE 20th Annual Symposium on*, pages 48–55. IEEE, 2012.

[68] VMWare, Inc. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. `http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf`, 2007.

[69] Edward Walker. benchmarking Amazon EC2 for high-performance scientific computing. *Usenix Login*, 33(5):18–23, 2008.

[70] John Paul Walters, Vipin Chaudhary, Minsuk Cha, Salvatore Guercio Jr., and Steve Gallo. A Comparison of Virtualization Technologies for HPC. *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*, pages 861–868, 2008.

[71] Guohui Wang and T. S. Eugene Ng. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. *2010 Proceedings IEEE INFOCOM*, pages 1–9, March 2010.

[72] Andrew Whitaker, Marianne Shaw, and SD Gribble. Denali: Lightweight virtual machines for distributed and networked applications. (Figure 1), 2002.

[73] Albert X. Widmer and Peter A. Franaszek. A dc-balanced, partitioned-block, 8b/10b transmission code. *IBM Journal of research and development*, 27(5):440–451, 1983.

[74] Jun Xie, Yujie Su, Zhaowen Lin, Yan Ma, and Junxue Liang. Bare Metal Provisioning to OpenStack Using xCAT. *Journal of Computers*, 8(7), 2013.

[75] Y Zhai, Mingliang Liu, Jidong Zhai, X Ma, and Wenguang Chen. Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications. In *State of the Practice Reports*, pages 1–10, New York, NY, USA, 2011. ACM.

[76] Ou Zhonghong, Hao Zhuang, Jukka K Nurminen, Antti Ylä-Jääski, and Pan Hui. Exploiting hardware heterogeneity within the same instance type of Amazon EC2. *4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, pages 4–8, 2012.