Noise Estimation in HPC Cloud Networks

Bachelor Thesis

Author(s):
Rahn, Tobias

Publication date:
2021

Permanent link:
https://doi.org/10.3929/ethz-b-000513171

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Noise Estimation in HPC Cloud Networks

Bachelor Thesis

Tobias Rahn

September 8, 2021

Advisors: Prof. Dr. T. Hoefler, Dr. D. De Sensi, K. Taranov
Department of Computer Science, ETH Zürich

Abstract

As computing is moved more and more to the cloud due to scalability, affordability and availability, so are high performance computing (HPC) clusters. It is expensive to build a real-world cluster that needs the scale and performance to run the highest predicted load but is not used to its full extent most of the time. Additionally, by relying on cloud services, the highly specialised workforce needed to maintain the cluster is no longer needed. As a result, more work can be put into the development of applications. With clusters in the cloud, it is possible to build a specific cluster just for the time it is used and with exactly the characteristics that are needed. This significantly lowers the cost of running HPC applications and opens up the possibility to run such applications to a wider variety of researchers and the public. The decreased cost and ease of use of such systems further increase the availability of running HPC workloads. Some leading providers such as Amazon, Microsoft and Google all have their special tailored solutions that try to provide a replication of the performance that a local cluster delivers. Each provider has its solutions to ensure that the nodes that make up the cluster are connected by a consistent high throughput, low latency network and that the virtualisation and sharing of the hardware have no noticeable impact on the performance. But do these promises hold and can it be expected to see the advertised performance on these systems? In this thesis, we start to answer this question.
We show differences between the providers and that Azure performs the best and most similar to a non-virtualised cluster.

Contents

Chapter 1

Introduction

High performance computing (HPC) clusters, also known as supercomputers, are an integral part of computational science. It finds applications in a wide variety of scientific fields with the movement to use computers as a tool to assist humanity in improving everyday life. In Healthcare, HPC is used to analyse medical data faster and for more advanced tasks such as genome sequencing. City planners use HPC to make better design choices in order to increase the quality of life in large cities and decrease environmental pollution. Machine Learning and artificial intelligence are used to learn patterns and detect fraudulent credit card transactions, which is in our best interest. To effectively train large machine learning models a lot of computing power is needed. Moreover, HPC is used in research to improve medical treatments or to improve the efficiency of renewable energy sources and develop new materials for construction that drive innovation. This shows that a lot of people benefit if HPC is available at an affordable price and to a wide range of researchers.
The growth in cloud computing and the performance increase has lead to more powerful virtual machines in the cloud. Can cloud computing provide the possibility to make HPC more affordable and available by offering HPC-as-a-Service? If this is possible the need for researchers to have access to the expensive and highly optimised hardware of a supercomputer is no longer a means to an end and a large amount of computing power can be accessed from everywhere. To the best of our knowledge, there has not been an extensive study that analyses the network stack of different providers that offer HPC-as-a-Service and benchmark them against a non-virtualised cluster.
In our work, we will analyse the network stack of Amazon, Microsoft and Google. Moreover, we will assess the performance of different layers in the network stack. This includes latency and bandwidth benchmarks of the Message Passing Interface (MPI), the Transmission Control Protocol (TCP)

and the InfiniBand (IB) verbs interface. We do this to get a better understanding of the different approaches that the three cloud providers take to offer HPC-as-a-Service in the cloud. We want to analyse the performance gap between virtualised cloud clusters and traditional non-virtualised supercomputers and see where the differences lie.

Our contributions with this work are the following:

- An evaluation of the three different cloud providers with a special focus on the networking components such as the network stack and the network interface(s).
- Compare the performance of the HPC cluster on the three providers and compare them to a non-virtualised cluster. We do this to see if the HPC clusters in the cloud are a viable alternative to on-premise supercomputers.

The rest of this thesis is arranged as follows: Chapter 2 provides some background on topics covered in this thesis and looks at some earlier work done in the context of benchmarking virtualised clusters. In the next chapter, Chapter 3, we look at the different ways that Amazon, Microsoft and Google chose to provide HPC-as-a-Service. We will specifically analyse how they set up their network stack to provide low latency and high bandwidth networking. The experiments used to evaluate the respective solutions and their results are discussed in Chapter 4. Finally, Chapter 5 summarises our findings and outlines future work.

Chapter 2

Background

In this chapter, we provide some background information on high performance computing, cloud computing in general and some related work.

High performance computing (HPC) is the task of processing data with a high throughput rate and performing complex calculations. Usually, a supercomputer is used for these tasks. A supercomputer is a cluster that consists of hundreds or thousands of servers (called compute nodes) interconnected by a high-speed network. Together they work like an orchestra where each node contributes its share of work. Many different technologies benefit from this large amount of computing power to process the massive amount of data that is produced. For example, hugely complex weather simulations can be run at a higher resolution to give better estimates and spot natural catastrophes earlier to save lives.

Cloud Computing was defined by the National Institute of Standards and Technology (NIST) in 2011 as follows: "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*" [32]. Traditionally HPC has been used on-premise by large institutions that build their own system that is optimised to get the best performance out of it. On the other hand, cloud computing provides more flexibility and does not force an institution to buy and build its own system. Instead, the system can be rented for the time needed with the specific configuration. Furthermore, cloud computing increases the availability and redundancy of corporate computer infrastructure. This is achieved by hosting the service in multiple data centres that are geographically separated to provide resilience against any interruptions such as power outages or floods. The recent push to improve the cloud infrastructure in the

last decade led to HPC-as-a-Service being an alternative to local clusters. As mentioned earlier, this offers several advantages, such as the possibility to scale up the computing resources for specific projects and the ability to easily customize the computing environment. There are different types of services offered in the cloud. First, we look at Infrastructure-as-a-Service (IaaS). In this model, the cloud provider only provides the hardware such as the server, storage, networking hardware and possibly a hypervisor. This most basic form of cloud computing outsources the hardware, but still needs technicians to maintain the system on this hardware. In a further step, more control can be handed over to the cloud provider. In the Platform-as-a-Service (PaaS) model some major software components like the operating system are additionally outsourced. Finally, with the Software-as-a-Service (SaaS) model everything is managed by the cloud provider. In this case, the company does not need to install and maintain any software and applications. HPC-as-a-Service can not be put in any of the three models, as it can be offered as any of them.

Related Work

Several works analysed the networking capabilities of HPC cloud clusters. Many of those works analyse the MPI performance on Amazons Elastic Compute Cloud and Microsoft Azure. However, to the best of our knowledge, no study exists that extensively evaluates and compares Amazon's, Microsoft's and Google's solutions to provide networking that is suited for HPC with a specific focus on networking performance.
Guidi et al [22] compared the performance of two AWS HPC clusters with 25 Gbps and 75 Gbps bandwidth respectively to two partitions (Haswell and KNL) of the Cori supercomputer [4] with 82 Gbps bandwidth. They analysed the computational power of the central processing units (CPU), the memory subsystem and the internode communication performance and showed that cloud clusters made a significant step in the right direction. Our work will go beyond that and not only compare an AWS HPC cluster with 100 Gbps networking to a non-virtualised cluster but also an Azure and a GCE HPC cluster. Additionally, we will evaluate the latency and bandwidth for TCP and IB communication besides benchmarking the MPI performance.
Chakraborty et al [17] intensively studied Amazon's in house developed transport mode Scalable Reliable Datagram (SRD) which provides hardware reliable delivery. They propose designs to improve the MPI performance under the specific characteristic and features of the Elastic Fabric Adapter (EFA) that Amazon designed itself and showed that the new transport mode SRD has a positive effect on the performance of collective operations. In our work, we will not do any optimizations based on specific solutions that the providers use to offer HPC performance in the cloud. Instead, we will focus

on a comparison between the three providers.
Tuning the HPC clusters in the cloud can boost the performance massively as shown by Xu et al [42]. An evaluation of the MPI latency and throughput performance under different settings on AWS and Microsoft Azure clusters showed an improvement of up to 150% in point-to-point communication bandwidth. To achieve these improvements different algorithms and protocols were used to see which one performs the best for each message size. For AWS they additionally developed a new zero-copy design in *MVAPICH2-X* [8] as the original was not able to deliver good performance. As Azure allows root access to their high performance VM's, the *XPMEM* kernel module[1] can be installed which significantly improves the throughput. Instead of tuning system characteristics and evaluating how different MPI libraries fare, the focus of our work lies in spotting and analysing differences between the providers and understanding why those differences exist.
Uta et al [41] discuss how reproducible and predictable experiments in the cloud are. Due to the multi-tenant nature, and the coupled sharing of the underlying hardware, the application of different users can have an impact on each other's results. This can lead to a lack of reproducibility and predictability, despite the different mechanisms that cloud computing providers employ to ensure Quality of Service and fairness. Additionally, often a bucket token system is employed for different resources such as network bandwidth and CPU scheduling. This means that a certain VM has a budget for the bandwidth and if that is used up, it is significantly slowed down. This can lead to different results for the same experiment if the buckets are still partially filled from earlier experiments that did not deplete them. These measures make it more difficult to draw scientific conclusions but Uta et al [41] provide some guidelines that should help to make the results more predictable and reproducible.
Compared to all the works listed here we want to take an extensive look at the networking solutions that are provided and compare them against each other. Moreover, we want to evaluate how they perform in different benchmarks to analyse different layers of the network stack (MPI, TCP and IB).

[1]This enables a process to map the memory of another process into its own virtual address space.

Chapter 3

Cloud Computing interconnects

In this chapter, we compare and analyse the cloud platforms of the three cloud computing providers Amazon, Microsoft and Google. In this work we refer to them as follows: Amazon Elastic Compute Cloud (EC2)[1] [1], Google Compute Engine (GCE)[2] [27], and Microsoft Azure [7] concerning different features in terms of their solutions to provide HPC-as-a-Service. To start this chapter Table 3.1 provides a rough overview of the three providers.

	Amazon EC2	**MS Azure**	**GCE**
NIC	Nitro Card (SoC), EFA and ENA	Mellanox (FDR, EDR, HDR), smartNIC (FPGA)	gVNIC
OS bypass	Yes (EFA)	Yes (Mellanox)	Partial (Intel DMA Technology)
Max. Bandwidth per instance	Up to 400 Gbps	Up to 200 Gbps	Up to 100 Gbps
Network architecture / topology	High-radix Folded Clos Topology with ECMP	Non-blocking low-diameter fat tree	Clos Topology
Cluster Management	AWS Parallel Cluster AWS Batch	Cycle Cloud Azure Batch	Terraform

Table 3.1: A high level overview of the most important aspects of the cloud clusters

3.1 Network solutions

We will now look at the networking solutions of the three providers. In Section 3.1.1 we describe the network stack and in Section 3.1.2 we will look at the different network interfaces that are attached to the virtual machines.

[1]We will often use the term Amazon Web Services (AWS) interchangeably with EC2 as it is a core component of it.

[2]Sometimes we refer to it as Google Cloud Platform (GCP) as GCE is a component of it.

3.1.1 Network Stack

In this section we outline the characteristic of the network stack and see the different approaches taken. We will start with Amazon and the Elastic Fabric Adapter. Then we will continue with Microsoft and *Accelerated Networking* before looking at Google's *Andromeda* network stack.

3.1.1.1 Amazon

AWS instances (HPC and "normal" instances) are launched into a Virtual Private Cloud (VPC) which consists of one or more subnets. Each instance has a primary network interface, which is a logical virtual network card and has a primary private IP address. It can be chosen whether an instance has a public IP address or not. The logical component that represents the network card in the VPC is called elastic network interface (ENI) which is connected to a network card. Depending on the instance type, multiple network interfaces can be attached to improve the networking performance, notably a Bandwidth (BW) above 100 Gbps. In the following, we will look at the more specific solutions that Amazon provides to improve the networking performance. Enhanced networking can be either enabled through an Elastic Network Adapter (ENA) or the Intel 82599 Virtual Function (VF) Interface. On all but the T2 instances, enhanced networking can be enabled. The ENA allows speeds of up to 100 Gbps while it is only 10 Gbps with the Intel 82599 VF interface. With single root I/O virtualization (SR-IOV)[3] it provides higher networking capabilities (higher BW, lower network jitter and consistently lower inter-instance latencies). To further increase the networking performance, Amazon offers the Elastic Fabric Adapter (EFA). It is an ENA with added capabilities to bypass the operating system (OS) that provides lower and more consistent latency, a higher throughput than TCP transport and supports OpenMPI [9], Intel MPI [18] and Nvidia's Collective Communication Library (NCCL) [12]. The functionality to bypass the OS is based on the IB verbs interface. Figure 3.1 shows the network stack with and without the EFA. For instances that support the Elastic Fabric Adapter, up to one can be assigned per network card. Only the P4d instance (accelerated computing) supports more than one, namely four, network cards, all others only support one. In total, there are four choices of which one can choose a network interface. The basic ENI, which is just a simple virtual network card. Then we have the next step with SR-IOV provided by the ENA and the Intel 82599 VF interface. And Last we have the EFA which is essentially an improved version of the ENA that is tailored for tightly coupled workloads such as high performance computing. HPC applications use the aforementioned libraries to interface the Libfabric API [6], which then,

[3]SR-IOV is a device virtualisation method that provides lower CPU usage and higher I/O performance by letting one PCI-device to be seen as multiple physical devices.

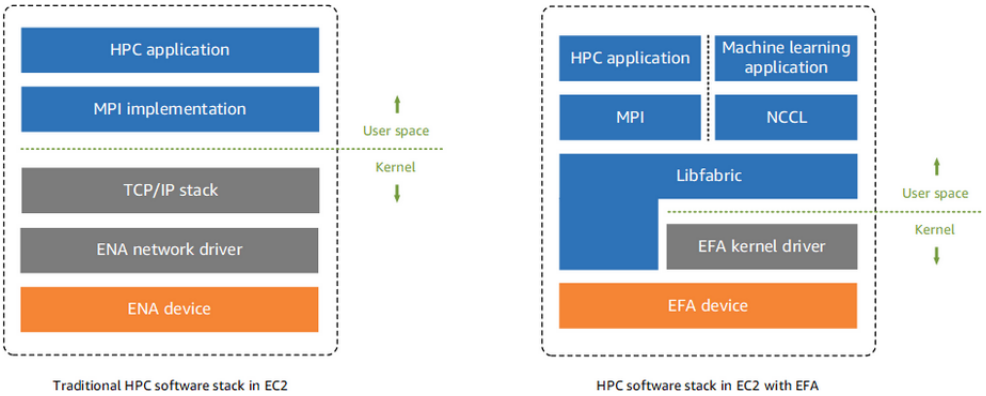


Figure 3.1: Traditional TCP network stack vs EFA Libfabric network stack, Source: [15]

in turn, bypasses the operating system kernel to directly communicate with the EFA. Libfabric defines interfaces between applications and the underlying fabric service. It targets high-bandwidth, low-latency Network Interface Cards (NIC) and scales to tens of thousands of nodes.

Amazon does not use a separate network infrastructure for the HPC instances and instead uses the existing high-radix[4] Folded Clos[5] topology out of commodity switches with equal-cost multipath (ECMP) routing to provide affordable supercomputing. TCP is ill-suited for latency-sensitive applications because the retransmission of lost packets can produce outliers with very high latency. Amazon states that Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) [6] is not scalable enough for their infrastructure. One of the specified reasons is the requirement to have priority flow control (PFC) which can lead to head-of-line blocking and additionally RoCE also suffers from ECMP collisions under congestion. The EFA supports the Unreliable Datagram (UD) mode [14] and the in house developed Scalable Reliable Datagram (SRD) mode [39]. The goal of SRD is to load balance the packets across multiple paths while providing fast recovery from packet drops and link failures. In SRD the sender controls the path selection by manipulating the packet encapsulation. SRD guarantees reliable and out-of-order packet delivery. This means that the packets must be reordered by a higher layer in the network stack. But this allows for multiple applications to reorder their specific packets separately without interfering with the streams of the other applications. Libfabric in the layer above reorders the packets. Congestion control has the aim to get a fair share of the available bandwidth with the minimum number of bytes in flight to prevent incast congestion and resulting packet drops. To that end, they use a dynamic per-connection rate limit and an inflight limit. Congestion is detected

[4]High radix = large number of skinny ports per router instead of a few fat ones
[5]Also called fat tree
[6]Also known as InfiniBand over Ethernet

if the RTT goes up on a majority of paths or the estimated rate becomes lower than the transmission rate. Network-wide congestion that affects all paths can be detected early and the individual sending rates can be adjusted. Congestion on individual paths can be handled with rerouting.

3.1.1.2 Microsoft

Microsoft uses the Azure Virtual Network (VNet) as the fundamental building block for the private network in the Azure cloud. It enables the VM instances to communicate with each other and the internet. To improve the network performance Microsoft utilizes what it calls *Accelerated Networking* (AccelNet) to bypass the virtual switch by offloading the policies applied by it (network security groups, access control lists, isolation, and other network virtualized services to network traffic) to hardware. This is achieved by enabling SR-IOV. Thus on arriving at the network interface, the traffic is directly forwarded to the VM and does not pass through the virtual switch. This results in lower latency, a higher packet per second (PPS) rate, reduced jitter and decreased CPU utilization as it has to process less network traffic if the policy enforcement is offloaded to hardware. Microsoft uses IB to

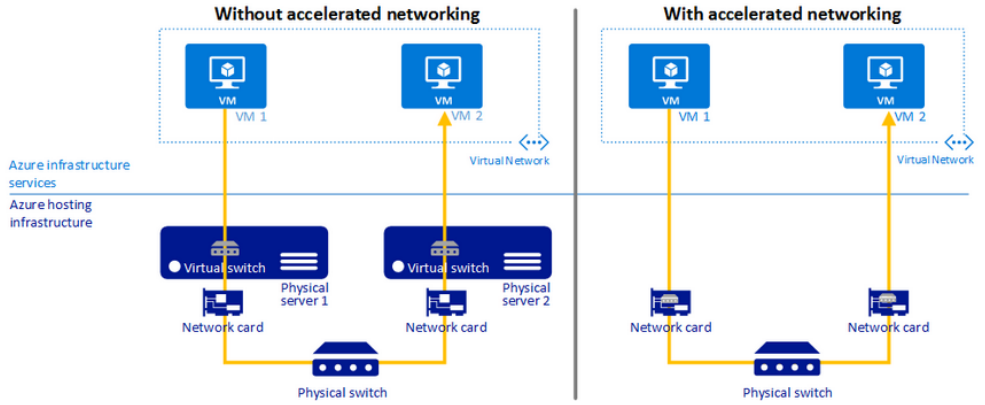


Figure 3.2: Communication with and without Accelerated Networking, Source: [33]

provide scalable MPI performance for HPC workloads. Depending on the specific instance Microsoft uses Mellanox Connect-IB (56 Gbps), Mellanox ConnectX-5 (100 Gbps) or Mellanox ConnectX-6 (200 Gbps) Network Interface Cards. The low latency, high bandwidth IB network provides RDMA capability. The structure of the interconnect is a non-blocking fat-tree with a low diameter design for optimized and consistent RDMA performance.

3.1.1.3 Google

Figure 3.3 shows Googles self-designed network virtualization stack *Andromeda* [20], a software-defined networking (SDN) [7] substrate for their network virtualisation platform and it sits on top of its own Jupiter network fabric. *Andromeda*'s data plane is based on a flexible hierarchy of packet

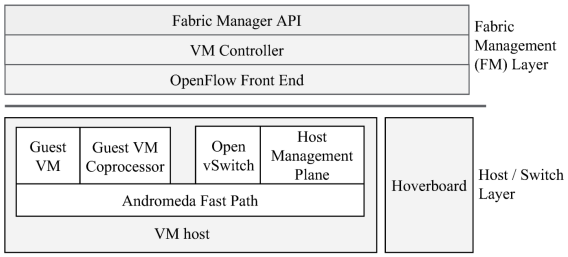


Figure 3.3: Andromeda stack, Source: [20]

processing paths. The on-host *Fast Path* is the top of this hierarchy above the on-host *Coprocessor path*. The third layer is the *Hoverboard path*, which consists of dedicated Gateways called Hoverboard Gateways. To dynamically select whether a flow is processed on one of the on-host paths or not, Google introduces the Hoverboard Programming Model. This model makes use of the advantages of the On-Demand Model[8] and the Gateway Model [9]. By default, the VM's send all flows via the dedicated Hoverboard gateways. If a flow exceeds a certain usage threshold it is dynamically detected by the control plane and offloaded to a direct host-to-host path that bypasses the gateway. The sending VM sends usage reports to the control plane which then detects these flows based on the reports. Figure 3.4 illustrates this direct host-to-host path. For such a direct host-to-host path the necessary routing information is stored in the on-host forwarding table and only flows for which there is no information in this forwarding table are then sent to the Hoverboard Gateway. With this method, the per-server memory utilisation is lowered as not all hosts need to store forwarding information for the whole network. In the following we discuss the on-host *Fast Path* and *Copro-*

[7] In an SDN the data plane (hardware) and the control plane)software) are separated, thus the hardwares sole purpose is to forward traffic and it has nothing to do with the routing. This results in a central control unit and the intelligence is not partitioned into the different switches and routers in the network.

[8] The first packet of a flow is sent to the controller and determines the route for the whole flow. This scales better than the Preprogrammed Model as not all flows need to be known in advance.

[9] Packets of a specific type are sent to a gateway device that is designed for high speed packet processing. The advantages of this model are that it provides predictable performance and that changes in the virtual network state only need to be communicated to the gateways instead of all VM's.

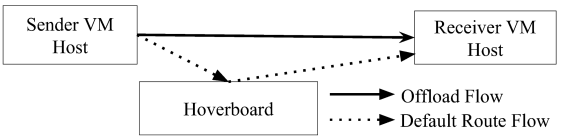


Figure 3.4: Hoverboard Packet Forwarding, Source: [20]

cessor Path in more detail. Figure 3.5 shows an overview of the on-host data plane with the two different paths. The *Fast Path* is used for performance-critical flows and is designed for high performance packet processing such as encapsulation and routing via the aforementioned on-host flow table. To this end, it has its own egress and ingress engines that are used for packet processing and other periodic work. Another measure to improve the performance is to avoid thread handoffs which is achieved by letting the *Fast Path* directly access the queues of the virtual and physical NIC. To keep the per-packet CPU cost low it is designed to minimize its number of features. If the allocated 300*ns* CPU budget is not enough the packets are offloaded to the per VM *Coprocessor Path* as can be seen in Figure 3.5. This design choice provides fairness and isolation between VM's. This path handles packets that need more processing and are not latency critical. For example packet encryption and abuse detection are handled via this path.

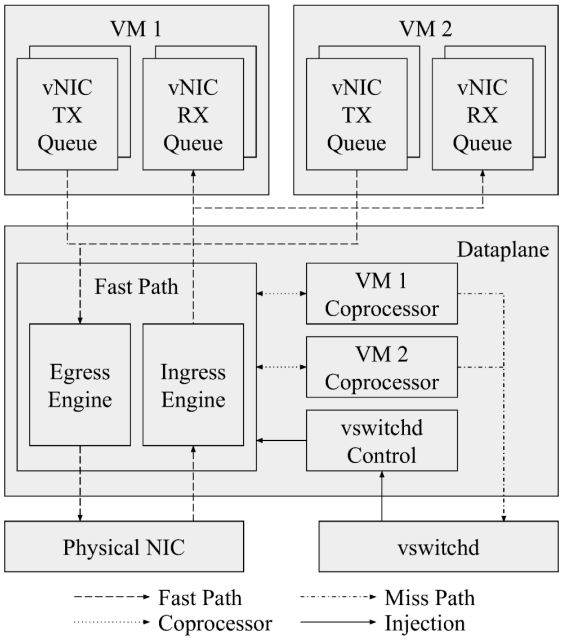


Figure 3.5: Andromeda on-host Dataplane overview, Source: [20]

Google gradually adds new features to its *Andromeda* network stack and it now supports packet offloading to the different paths as described before and Intel Quick DMA Engines to offload larger packet copies. We conclude this chapter with the key takeaway about the *Andromeda* network stack: Active, performance-critical flows are offloaded to a direct host-to-host path while idle/bursty flows are routed over Hoverboard Gateways.

3.1.2 Network Interface Card

Amazon and Microsoft have developed their own SmartNIC's to offload the network stack and management from computing resources that are rented to cloud customers.

3.1.2.1 Nitro Card for VPC

The AWS smartNIC is based on an in-house developed System on a Chip (SoC)[10]. Amazon builds its instances on top of the Nitro system. Among other things, this system consists of four Nitro Cards, of which one is the Nitro Card for VPC. The Nitro cards are a family of cards with the goal to improve system performance by offloading and accelerating I/O. The Nitro Card for VPC is a PCIe attached NIC that is the hardware interface between the EC2 servers and their network connections. The Elastic Network Adapter is the driver for these cards. It takes over the tasks of routing, implementing security groups (virtual firewall) and packet encapsulation/decapsulation. This has the benefit that no cores need to be reserved for AWS services to handle network tasks etc... and the user can use all cores to its disposal. The EFA driver uses network acceleration features of the Nitro Card for VPC's EFA hardware device to bypass the operating system when interacting with the user-space libraries to provide more consistent performance and lower CPU utilization.

3.1.2.2 Azure SmartNIC

The Azure smartNIC is attached to all instances. Instances that are InfiniBand ready, additionally have a second IB NIC which is one of the following: Mellanox Connect-IB, Mellanox Connect-X5 or Mellanox Connect-X6. The smartNIC is used to connect to the internet while the Mellanox NIC should be used for internode communication as the smartNIC only provides 50 Gbps bandwidth. To ensure that always the right network card was used during our evaluation, we specified the interface manually where it was necessary. Although the test used the Mellanox NIC we still want to discuss this smartNIC briefly as it is attached to our instances and a core component

[10] A System on a Chip integrates many if not all parts of a computer such as the CPU, memory, etc. . . on a single die.

of all other instances, that are not designed for HPC.
The Azure SmartNIC is an FPGA based custom-built smartNIC by Microsoft [21]. To improve the performance they moved to SR-IOV. This meant that they had to move the SDN stack to the NIC, as it gets bypassed together with the hypervisor which is bypassed to reduce the load on the CPU by allowing direct access to the NIC for packet processing. Because these SDN policies are fast-moving, they resorted to FPGA's due to their reprogramability in contrast to Application-Specific Integrated Circuits (ASIC). This flexibility is needed as SR-IOV is an all-or-nothing offload and if an SDN feature is not handled by the SmartNIC, the packets must be sent and processed via the SDN stack which almost negates the performance gain from the SR-IOV offload. The goal is to switch to ASIC's if they can be used for a longer time if the cloud needs are more stable [40]. While the control plane used for *Accelerated Networking* runs mostly in the hypervisor the data plane for *AccelNet* is offloaded to the FPGA SmartNIC.

3.1.2.3 Google Virtual NIC (gVNIC)

All instances on GCE use the virtIO-based ethernet driver [36] to connect to the network. This can be upgraded to the gVNIC to enable *Tier 1* networking and bandwidths between 50 and 100 Gbps. Unfortunately, Google has not published a paper where this NIC is described nor is there documentation on their website. Therefore it is not possible to compare it to the two NIC's of Amazon and Microsoft.

3.2 Virtual machine properties

In this section, we analyse different properties of the virtual machines. In Section 3.2.1 we look at the different options that the virtual machines can be placed physically close together for higher inter-node connection speeds or further apart for redundancy. Next, in Section 3.2.2, we will look at the different levels of hardware sharing and virtualisation that are offered. Last we will describe, in Section 3.2.3, the different instances that are advertised for HPC workloads.

3.2.1 Instance placement

All of the three providers we look at in this thesis have some form of grouping policy to place the Virtual Machines close together for better network performance.

3.2.1.1 Amazon

Amazon has three types of placement groups they offer:

Cluster puts the instances into the same availability zone.

Partition spreads instances across logical partitions, such that instance groups in different partitions do not share the underlying hardware.

Spread places small groups of instances on distinct underlying hardware.

For high performance computing *Cluster* is the preferred placement policy as the physical closeness of the instances results in low latency network performance. Per default only 20 instances can be launched per region, if more instances are needed in a single region, a request to Amazon can be submitted.

3.2.1.2 Microsoft

When creating instances on Azure the VM can be placed in the same region (a set of data centres within a latency-defined perimeter) and/or the same availability zone (one or more data centres, independent power, cooling and networking). Additionally, they offer Proximity Placement groups to bring the VM's physically closer together and reduce the latency further. It is advised by Microsoft to put the virtual machines into a single placement group for HPC. With a Scaleset this means that at most 100 virtual machines (300 with a quota increase) can be placed in a single Scaleset such that the VM's are not scattered for redundancy. If one uses availability sets (for availability and redundancy) the limit is 200 (maximum that can be put into one availability set).

3.2.1.3 Google

Google has the following placement policies on offer:

Spread spreads the instances out from each other which can be used for redundancy.

Compact keeps the instance physically close together.

The latter is preferred for HPC workloads as physically closer machines result in lower network latency. At most 22 machines can be placed close together and at most eight can be put into a *Spread* group.

3.2.2 Virtualization level

There are different options when choosing how to place the virtual machines. They can be explicitly put on different hardware and single tenancy to guarantee that other VM's have no impact on the performance.

3.2.2.1 Amazon

For most of its instance families, Amazon offers metal instances. There is no virtualization software (e.g: hypervisor) preinstalled and the application has direct access to the underlying processor and memory. Amazon also offers dedicated hosts, a dedicated physical server. In contrast to the metal instances, they have virtualization software preinstalled and different instances sizes of the same instance family can be mixed on it. On a dedicated host, multiple virtual machines of the same family can be run on and only VM's of a single user run on it. In contrast to the other two providers, AWS also offers dedicated instances. If an instance is launched, one can choose whether the instance is launched on shared-, dedicated hardware or a dedicated host. The dedicated instance gives the following guarantees: It runs on hardware that is not shared with other users, thus we have single-tenancy. No two instances that have the attribute dedicated are run on the same hardware, but other instances with the shared attribute of the same user can be placed on the same hardware to "fill it up".

3.2.2.2 Microsoft

Azure's bare-metal infrastructure provides non-shared server hardware that is maintained by Microsoft. It operates under the bring-your-own-license model, such that already existing licenses can be used. Similar to Amazon they also offer dedicated hosts to run one or more VM's on a dedicated server. But these are not available for the HPC optimized instances as they run by default in single tenancy mode with one VM per physical server. Furthermore, Microsoft offers to attach a Cray supercomputer of the XC- or the CS series to a virtual network to further improve the performance.

3.2.2.3 Google

Google offers single-tenant nodes, which give exclusive access to a node that can run multiple VM's like the dedicated hosts of the other two providers. A bare-metal solution similar to the other two providers that can be easily created over the web interface is not offered by Google. Instead, it must be requested from the support which gives more room for customization when requesting a bare-metal solution. In contrast, one can choose whether Google should provide the hypervisor or not.

We did not use dedicated hosts on AWS and GCE as it was not possible within our quota limits.

3.2.3 Instance Types

In this section, we will have a look at the different instances that support high speed internode communication and are advertised for HPC workloads by the providers. Additionally, we share our experience on increasing the quota[11] for the vCPU's as it is not enough to create HPC-ready VM's by default.

3.2.3.1 Amazon

Of the three providers, Amazon offers the most families of instances and also the most variety within the families. The different families have different goals. They offer general purpose, compute optimized, memory-optimized, storage optimized and accelerated computing instances that run on AMD EPYC, Intel Xeon or AWS' own *Graviton* ARM processors. They all have differing maximum network bandwidths going up to 100 Gbps for most instances and 400 Gbps, with four network cards that use the ENA or the EFA) for one instance type (*p4d.24xlarge*) in the accelerated computing family. For HPC instances the following families should be taken into consideration as they provide 100 Gbps networking and support AWS' EFA: *M5n*, *M5zn*, *C6gn*, *C5n* and *R5n*.
Amazon increases the limit automatically on request if the previous limit was extensively used for some time. This means that one can gradually increase the limit and build larger clusters after some time. As the time of this thesis is limited, we could only push the quota high enough to create three instances.

3.2.3.2 Microsoft

Azure has divided their instances into similar categories and has an additional category for high performance computing instances which include the *H-series*, *HB[v2|v3]-series* and the *HC-series* instances. All the instances run on AMD EPYC or Intel Xeon processors and the HPC instances have between 56 Gbps and 200 Gbps internode networking through IB NIC's.
Up to a certain limit, Microsoft increases the vCPU quota automatically if it is requested, but after this limit, one needs to provide reasoning why that additional vCPU's are needed. But the two instances we used for our evaluation were within this limit.

3.2.3.3 Google

Similar to the other two providers Google has divided its machines into different categories. As for Azure, they all run on either AMD EPYC or Intel Xeon CPU's. Google has only one compute-optimized machine type,

[11] All services in the cloud are limited and the quota specifies the per-category limit.

the *C2*. *Tier 1* networking was supported on the *C2* instances up until a few months before writing this document. It is now only supported on the general-purpose machine family with the *n2-standard-80* and *n2d-standard-224* instance yielding a 100 Gbps. The *C2* instances now only support 32 Gbps of internode communication bandwidth.
Google was by far the hardest to get a quota increase. They wanted a detailed explanation to see for which reasons we would need more vCPU's. After a phone call and using the lower tier instances for a while, they were willing to increase our quota enough that we can build a cluster consisting of three nodes.

To summarize we have up to 100 Gbps per instance on Google, up to 200 Gbps on certain instances for Microsoft and Amazon provides up to 400 Gbps for one specific instance type. In the end, we had enough quota on all providers to create three node clusters with different amounts of work. On Amazon and Google, we made the experience that it is worth using the existing quota and then ask for an increase.

3.3 Cluster Management

Last, we will compare how to deploy and manage clusters on the three platforms. EC2 relies on the open-source cluster management tool *AWS Parallel Cluster* [16], a Command Line Interface (CLI), to deploy and manage HPC clusters. It supports various schedulers such as Son of Grid Engine (SGE), Slurm Workload Manager (Slurm), Torque Resource Manager (Torque) and AWS's own AWS Batch. Amazon also offers AWS Batch as a service to its customers, where they take care of the cluster management part by automatically scaling and provisioning compute resources depending on the demand. The user can set a maximum and minimum number of nodes and submit their jobs to the job queue.
Azure Cycle Cloud [35] is the tool to manage HPC cluster on Microsoft Azure. Microsoft provides a CLI and a web interface to manage the clusters and many schedulers such as Slurm, LSF, HTCondor, SGE, PBS Pro, Symphony or Microsoft's own HPC Pack are supported. The maximum number of nodes, the instance type of the nodes and the scheduler can be configured. Microsoft offers a second service called *Azure Batch* [34] to run HPC applications. The user can run its job without creating a cluster or choosing a scheduler. *Azure Batch* creates and manages a pool of nodes, installs the necessary software and schedules the jobs for the user, who controls it via scripts, the Azure portal or the Batch Application Programming Interface (API).
Google has guidelines on how to optimise an instance to make it HPC ready, but they also offer a pre-configured *CentOS 7* based HPC VM instance. This

instance can either be launched from the console (web interface) or with the *gcloud* CLI. They do not offer a service as the other two to create a cluster. Instead, they provide a Terraform [23] script that can be adapted to create a cluster. Amazon and Google additionally provide the possibility to use the service Cloudy Cluster [26] to create a cluster.

Chapter 4

Evaluation

In this chapter, we will describe the setup that we used on all platforms, as well as the different metrics we evaluated and with which methods. The data that was generated during this evaluation and the scripts used therefore can be found in the git repository [37].

4.1 Setup

First, we will look at the different setups we used and how we set up the clusters on the different providers. We tried to use similar configurations on all platforms to get comparable results. As the cloud computing providers do not specify everything and do not offer all the same configurations, it is however impossible to get the same one. As we evaluate the network performance, the configurations must have the same network speed, such that they can be compared and to detect strengths and weaknesses. Because Google only provides instances with a maximum bandwidth of 100 Gbps, we also chose instances with this bandwidth on the other two providers. Moreover, because we run network-intensive benchmarks that do not perform any computation, small differences in the CPU will be neglectable in the results. In Table 4.1 we have an overview of the used platforms and their configuration.

	AWS c5n.18xlarge	MS AzureStandard HC44rs	GCP N2-standard-80	Slingshot interconnect based cluster
Cores per Node	36	44	40	128
vCPU's / Threads	36	44	40	256
Frequency	3 GHz	2.7 GHz	2.8 GHz	2.25 GHz
Processor	2 × Intel Xeon Platinum 8124M	2 × Intel Platinum 8168	2 × Intel Xeon Gold 6268CL	2 × AMD EPYC 7742
Memory	192 GB	352 GB	320 GB	512 GB
NIC	Nitro Card, EFA	Mellanox Connect-X5, smartNIC	gVNIC	Mellanox Connect-X5
Bandwidth	100 Gbps	100 Gbps	100 Gbps	100 Gbps

Table 4.1: Details of the used machines: Provider and VM type, number of threads (called vCPU in the cloud), processor base clock frequency, Processor Model, memory, Network Interface, advertised bandwidth (Gigabits/s)

4.1.1 Amazon Elastic Compute Cloud

On AWS we used the CLI AWS *ParallelCluster* to create the cluster and specify the parameters. As the goal of the thesis is to evaluate the network speed and compare it to real-world HPC clusters, we chose an EFA- and ENA-enabled instance that is advertised for HPC workloads: *c5n.18xlarge*, the details for this specific virtual machine type can be found in Table 4.1. The *p4d.24xlarge* instances that provide 400 Gbps were not evaluated as our quota did not allow us to create two of these instances and Google only supports a maximum of 100 Gbps internode communication bandwidth. We used the Slurm workload manager to run MPI jobs. To ensure that the instances are physically close together we put the instances into the same placement group. The operating system on the master and compute nodes is *CentOS 7* because Google only provides a preconfigured *CentOS 7* HPC image. As AWS did not suggest a specific MPI library, we chose Intel MPI as we used that on GCP and HPC-X [11] was not available. As the option to run *dedicated instances* does not exist on GCP, we did not use it on AWS neither. Nonetheless, we assume that the instances were run in single tenancy and did not share their hardware, because of the instance parameters [1].

4.1.2 Microsoft Azure

On Microsoft, we use the browser-based Graphical User Interface version of their tool *Azure CycleCloud*. We host this tool on a general-purpose VM. Clusters that are managed with *Azure CycleCloud* automatically have their compute nodes put into proximity placement groups to ensure physical closeness. For the compute nodes we again used VM's that are advertised for HPC workloads and support 100 Gbps networking with *Accelerated Networking*: *Standard HC44rs*. We chose to not evaluate the newer HPC

[1] The used instances take up all physical cores that the two available processors provide.

instances equipped with the Mellanox Connect-X6 that would provide 200 Gbps internode communication bandwidth because Google does not support bandwidths over 100 Gbps. On Azure, the VM's that are advertised for HPC workloads run in single- tenancy by default. We again used the Slurm workload manager to run MPI jobs and *CentOS 7* on the master and compute node. We used HPC-X as Microsoft suggested to use it if the task doesn't specify one and one has the freedom to chose a library.

4.1.3 Google Compute Engine

The cluster on Google consisted of two compute nodes without a workload manager nor a master node. To ensure that the two instances are physically close together we used a compact placement policy. The first tests were run on *c2-standard-60* instances, as they are advertised for compute-intensive workloads. However, after a few days, Google removed the *Tier 1* networking support for the compute-optimized instance family and the tests shown in this thesis were then executed on *n2-standard-80* instances. We used *CentOS 7* as the operating system on the compute nodes, as this is the only operating system that comes preconfigured with Google's HPC optimizations. We used Intel MPI as suggested by Google's guidelines for HPC. Google's tutorial [31] on how to set up a VM for HPC suggest to use Intel MPI 2018 which can be installed by providing a flag (–metadata=google_install_mpi="–intel_mpi") on creation of the VM. this version only comes with *mpirun* and has no compiler, like *mpicc*, included. Thus we used a slightly newer version of Intel MPI [19] to compile the respective benchmarks.
Alternatively one could build a traditional cluster with a workload manager and a master node using Terraform as described by Google in their tutorial. But it is not documented how to turn on the *Tier 1* networking for the compute nodes using Terraform.

4.1.4 Slingshot Interconnect Based Cluster

The physical cluster we used as a baseline to compare our virtual clusters which is one of the Swiss National Super Computing Centre [3]. It is based on the Slingshot interconnection network [38] and uses Mellanox ConnectX-5 (100 Gbps) NIC's.

4.1.5 Benchmarks

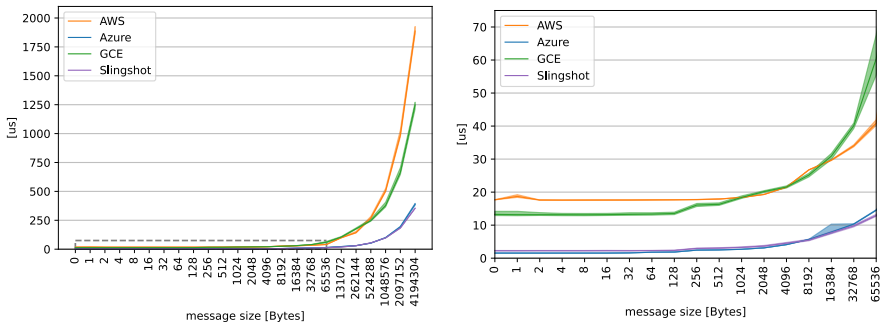
We used various benchmarks during our evaluation to test different characteristics of the systems. To assess the TCP throughput under different circumstances we used the latest version of iperf3 [5]. We used iperf3 instead of iperf2 as it is the more advanced tool. To assess MPI latency and throughput we used the OSU benchmark suite [10]. The perftest micro-benchmark collection [13] was used to evaluate the IB performance. Because Google

doesn't use InfiniBand, the perftest benchmarks can not be executed on the GCP cluster. On AWS they exist in a limited form for the SRD protocol, *ib_write_bw* and *ib_write_lat* are not supported and the maximum message size is 8 KiB. We also assessed the OS noise by using the OS noise test of *Netgauge* [24]. To analyse the behaviour of the Hoverboard Programming Model with the dedicated Hoverboard Gateways on Google, we wrote our own benchmark that is described in Section 4.4.1.
For most of the tests, we created band plots to have some sense of the variability. We did that by running the same test ten times and then plot the mean with a band around it that consists of the minimum and maximum over the ten iterations. If there is a difference in how we collected the data and plotted it, it will be described in the corresponding section.

4.2 Point-to-Point latency

First, we will compare the latency on the different systems. The OSU benchmarks were used to evaluate the latency for MPI applications. We used the standard *osu_latency* benchmark and additionally the *osu_multi_lat* benchmark which runs multiple parallel streams. We did that as one needs multiple streams to saturate the full bandwidth on 100 Gbps instances on GCE as stated by Google [28] and we consequently wanted to see how the latency behaves with multiple streams. To measure IB latency we used the *ib_send_lat* and *ib_write_lat* on the providers for which they were available. We will look at the benchmarks seperatly and start off with MPI latency.

4.2.1 Single Stream MPI Latency



(a) All message sizes

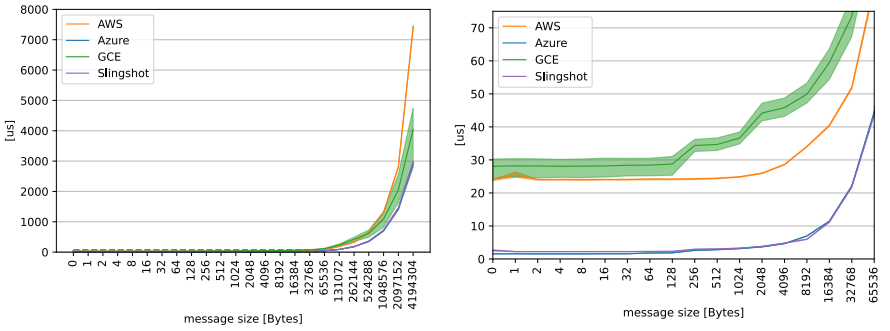
(b) Zoomed in on the dashed rectangle in Figure 4.1a

Figure 4.1: MPI point-to-point latency using two nodes with one process per node.

We observe a similar MPI latency for AWS and GCE in Figure 4.1a. Azure is the only virtual cluster that has similar if not the same performance as the

Slingshot interconnect based cluster. In the zoomed-in plot 4.1b we see that the cluster on Azure has identical latency to the physical cluster while the cluster on AWS has a slightly higher latency than the cluster on GCE.

4.2.2 Multi Stream MPI Latency



(a) All message sizes

(b) Zoomed in on the dashed rectangle in Figure 4.2a

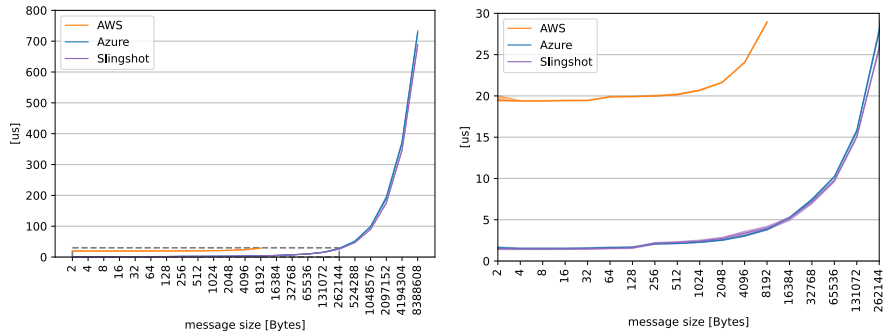
Figure 4.2: MPI point-to-point latency using two nodes with 16 processes each and 16 parallel streams.

Note that we have a different ordinate scale in Figure 4.2a than in figure 4.1a to be able to better understand the differences. We ran the OSU multi-stream benchmark with 16 streams as that is the number of streams that Google suggests to measure the performance of their 100 Gbps instances when using iperf. Figure 4.2b shows that for smaller message sizes, the MPI latency is the same for a single and multiple streams. But for larger messages (128 KiB upwards) the MPI latency is higher with multiple streams as can be seen in Figure 4.2a. This behaviour is probably due to buffering on the host machines as more data is injected into the network. This effect lets the latency rise further and further for every new packet that arrives if they keep getting larger. But we can again see that the Azure and the Slingshot interconnect based clusters have identical performance, while GCE and AWS are again similar and have a noticeably higher latency than the other two. For the largest few message sizes AWS has by far the highest latency for the single and multi-stream case.

4.2.3 Single Stream IB Latency

The IB latency is marginally better for the Slingshot based cluster and Azure while it is higher for the AWS cluster as can be seen in Appendix B.1[2]. Additionally, these plots show that the performance for *ib_send_lat* and *ib_write_lat*

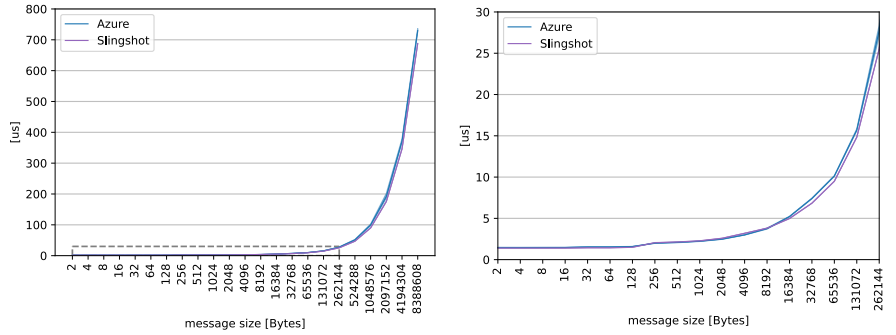
[2]These additional plots can be found in the appendix to not disturb the flow of reading with too many graphs.



(a) All message sizes

(b) Zoomed in on the dashed rectangle in Figure 4.3a

Figure 4.3: IB point-to-point latency using two nodes using *ib_send*



(a) All message sizes

(b) Zoomed in on the dashed rectangle in Figure 4.4a

Figure 4.4: IB point-to-point latency using two nodes using *ib_write*

is the same. Figure 4.3 shows that AWS again has a higher latency than both Azure and the Slingshot based physical cluster. But again we see that Azure is very close in terms of performance to a real cluster for both *ib_send_lat* and *ib_write_lat* which can be seen in Figure 4.3 and Figure 4.4 respectively.

4.3 Point-to-Point throughput

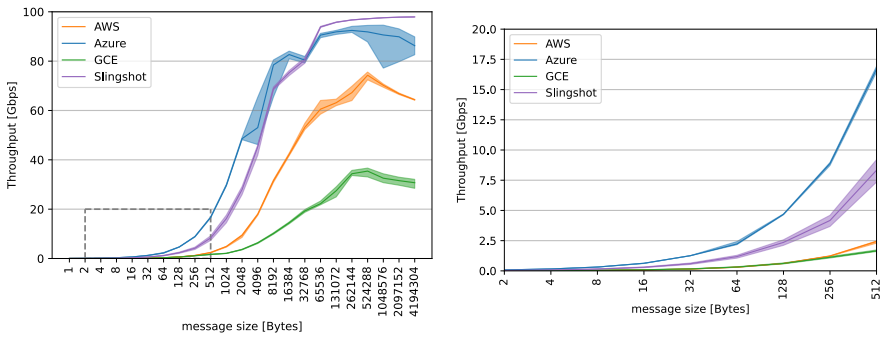
In this section, we analyse the throughput of the providers. They all advertise 100 Gpbs networking on the instances we chose. We want to evaluate whether this advertised bandwidth is reached, whether it can be achieved consistently over a longer period of time and if it is unidirectional or bidirectional.

4.3.1 Single Stream Unidirectional Throughput

In this section, we will specifically look at the single-stream unidirectional throughput that the providers can achieve. We want to see the performance

they achieve for MPI with the *osu_bw* benchmark. Additionally we used the *ib_send_bw* and *ib_write_bw* to evaluate the performance that the hardware is capable of. With the iperf3 test, we tested how well TCP flows are handled by the different systems.

4.3.1.1 MPI Throughput



(a) All message sizes

(b) Zoomed in on the dashed rectangle in Figure 4.5a

Figure 4.5: MPI point-to-point throughput using two nodes with one process per node

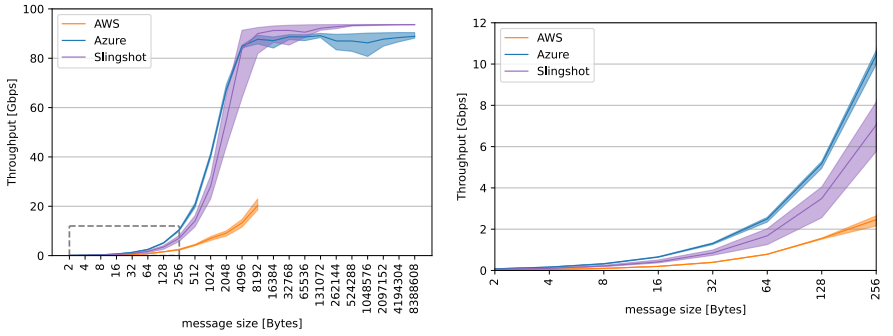
We will start with the *osu_bw* benchmark, to measure the MPI performance, which uses the non-blocking *MPI_Isend* and *MPI_Irecv* to send messages in batches of 64. In Figure 4.5a we observe that GCE has the worst performance out of the four platforms for single-stream unidirectional communication. We assume that this is due to a software stack that is not optimized as the hardware should be capable of delivering 100 Gbps throughput. As mentioned before, Google advises using multiple streams to test the throughput for instances with a fast network connection, which strengthens this assumption. AWS has a similar behaviour as GCE considering that it does not reach its full potential with just one stream which also leads us to believe that the software stack is not fully optimized yet. Azure reaches around 90 Gbps and is very closely matched to the physical Slingshot interconnect based cluster, but with a lot more variance. It is interesting to note that the virtualised cloud clusters have a higher throughput than the physical cluster for smaller message sizes which can be observed in Figure 4.5b, but for larger message sizes the physical cluster fares far better. This might be caused by differences in the software stack or in the network technology that is optimized to perform better smaller message sizes.

4.3.1.2 IB Throughput

Next we will look at the IB performance. Unfortunately IB is not supported on GCE and only available for message sizes up to 8 KiB on AWS as men-

tioned before. With the following two test we want to analyse how the performance of the network interface is without all the parameters and the variability that the different versions and implementations of MPI bring into play.

First, we will have a look at the performance under the *ib_send_bw* mi-



(a) All message sizes

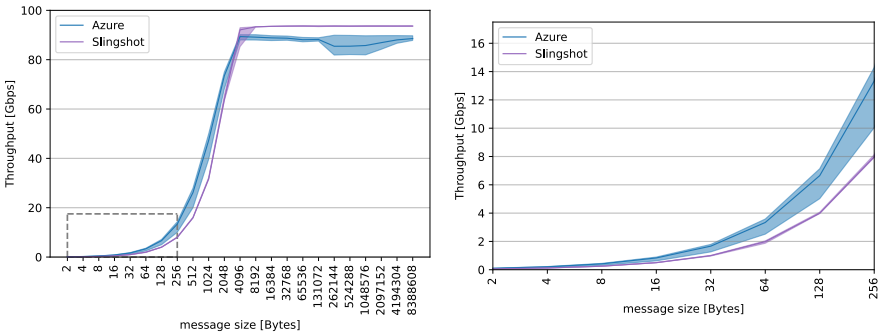
(b) Zoomed in on the dashed rectangle in Figure 4.6a

Figure 4.6: IB point-to-point throughput using two nodes using *ib_send*

crobenchmark. In Figure 4.6a we see the common theme that Azure performs similarly to the non-virtualised cluster and that the cluster on AWS lags behind a bit. We can also see that the Slingshot based cluster has more variance for medium sized messages compared to the cluster on Azure, but it then reaches a higher peak throughput and has next to no variance for the large messages. The Azure cluster reaches a similar peak throughput as for the MPI single-stream test while the Slingshot based cluster reached a higher peak throughput with the former[3]. In Figure 4.6b we can again see that the real cluster has more variance in the throughput than the virtualised ones.

Next, we will look at the *ib_write_bw* microbenchmark in Figure 4.7. Unsurprisingly, Azure and the Slingshot based cluster are again closely matched and Azure has the lower peak throughput as before. We see, especially in Figure 4.7b, that Azure now has a lot more variance than the real cluster. As this test uses RDMA it does not use the receiving side CPU as it directly writes into the receiving machine's memory and there is no need for a receiving function to be called. It thus makes sense that we have less variance for the real cluster. For Azure, we probably have more variance through the virtualisation as we most likely cannot write directly to the receiving side memory as it is also hidden behind some level of virtualisation.

[3]This can be seen very clearly in the plots in appendix B.2.1 where the different benchmarks are compared per provider



(a) All message sizes

(b) Zoomed in on the dashed rectangle in Figure 4.7a

Figure 4.7: IB point-to-point throughput using two nodes using *ib_write*

4.3.1.3 TCP Throughput

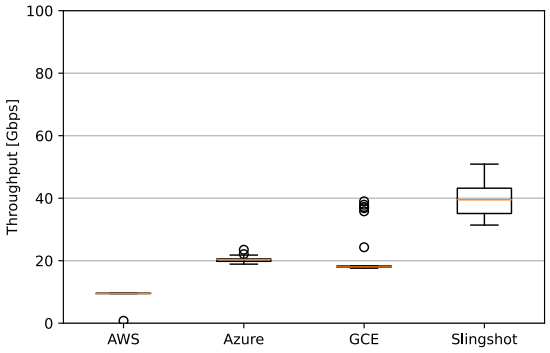


Figure 4.8: TCP point-to-point throughput using two nodes and 1 MiB messages. The box is defined by the first and third quartile while the orange line represents the median. The whiskers extend to $Q1 - 1.5 \cdot IQR$ and $Q3 + 1.5 \cdot IQR$ (IQR=Q1-Q3).

Last we will look at the TCP performance of the different platforms with the help of the iperf3 tool. To that end, we executed 25 runs of the benchmark. We can see that with a single stream none of the four systems can reach 100 Gbps. This makes sense as the transfer rate is dependent on the window size of the communicating parties and the RTT[4]. AWS has a very consistent throughput but has a relatively low transmission rate. We used the default values for the window size because manually tuning iperf3 for the different systems is outside the scope of this thesis. Thus the lower TCP throughput could be due to the parameters of iperf3. We can be pretty certain that the RTT should not be the problem here as we saw in section 4.2 that the latency

[4]Maximum transmission rate = TCP windows size/RTT in seconds, this is because we can send at most "TCP window size" many bytes without being acknowledged and it takes RTT/2 seconds for the packets to arrive and RTT/2 seconds for the acknowledgement packet to arrive back at the sender which results in RTT in total [30].

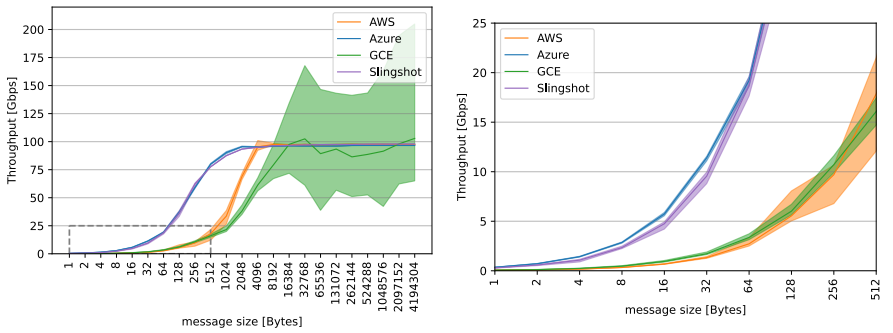
of AWS is the highest but similar to the one of GCE which has a similar performance in the TCP benchmark than AWS. Azure and GCE perform similarly, but GCE has more outliers. This shows that the VM's of Google should be capable of a throughput of 40 Gbps, but they can not hold it for a longer period of time with a single stream as most samples are concentrated at around 20 Gbps. The Slingshot interconnect based cluster reaches the highest median throughput, but has quite a high variance compared to the three virtualised clusters.

4.3.2 Multi Stream Unidirectional Throughput

As Google suggest using multiple streams in parallel to evaluate the performance on their instances, we followed their advice to see if it makes a difference and how they then compare to the other providers. We used 16 streams in parallel for all benchmarks because Google used this amount of parallel streams in their example of the iperf benchmark.

4.3.2.1 MPI Multi-Stream Throughput

We will again start with the corresponding benchmark from the *OSU* benchmark suite. They provide the *osu_mbw_mr* micro benchmark that runs multiple streams in parallel. With this benchmark one can specify how many streams to run with a flag when initiating the command. We note that in



(a) All message sizes

(b) Zoomed in on the dashed rectangle in Figure 4.9a

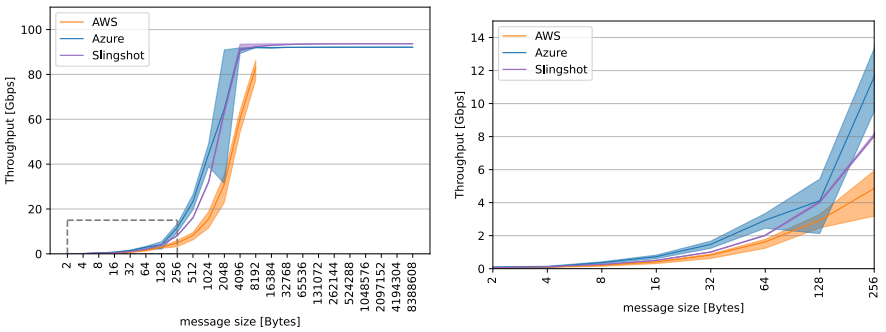
Figure 4.9: MPI point-to-point throughput using two nodes with 16 process per node and 16 parallel streams

Figure 4.9a AWS, Azure and the Slingshot based cluster peak at roughly 100 Gbps. While it is roughly the same for the Slingshot based cluster, it is a 10% increase for Azure and a 20% increase for AWS. Furthermore, we notice that Azure is again very closely matched with the physical cluster while AWS needs larger package sizes to reach its full potential. In Figure 4.9b we see that Azure again has the lower variance than the Slingshot interconnect based cluster and that AWS also has a noticeable variance for

the smaller message sizes. GCE reaches an unreasonable throughput in this benchmark. While the mean seems reasonable at around 100 Gbps, we have peaks at over 200 Gbps. We do not know what causes this behaviour. One reason could be that they employ multiple NIC's which each provide less than 100 Gbps, but it is not documented how many NIC's are attached per virtual machine. This could also explain why a single stream of data cannot fully saturate 100 Gbps.

4.3.2.2 IB Multi Stream Throughput

We will now look at the data we gathered with the perftest benchmarks. For these two tests there also exists a flag to specify how many queue-pairs should be used in parallel. First we will analyse the data from the *ib_send_bw* micro benchmark with 16 queue-pairs. In Figure 4.10a we see that Azure



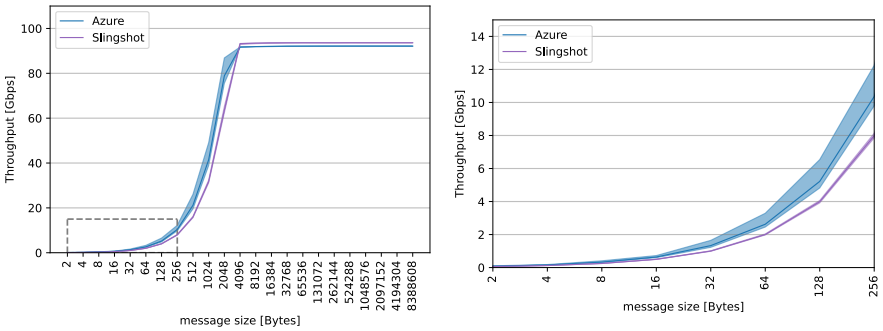
(a) All message sizes

(b) Zoomed in on the dashed rectangle in Figure 4.10a

Figure 4.10: IB point-to-point throughput using two nodes with 16 queue pairs using *ib_send*

and the Slingshot interconnect based cluster again reach a peak throughput of around 100 Gbps as advertised. Azure has a bit more variance for low to medium-sized packets. As this benchmark only supports a packet size of up to 8 KiB for AWS, we can only speculate that it would also reach a peak throughput of 100 Gbps. As it almost reaches it, has a similar curve to the other two albeit it begin shifted to the right and it having reached the same peak throughput as the other two in the MPI bandwidth test with multiple concurrent streams. In Figure 4.10b we can see that the non-virtualised cluster has the lowest variance also for small messages sizes in contrast to the other tests for only a single stream.

This second test from the perftest benchmark suite shows that the virtualised Azure cluster has a similar performance to the physical cluster. As we do not have the outliers in the band around the mean of the Azure cluster in Figure 4.11a we can assume that they were due to network noise in Figure 4.10a



(a) All message sizes

(b) Zoomed in on the dashed rectangle in Figure 4.11a

Figure 4.11: IB point-to-point throughput using two nodes with 16 queue pairs using *ib_write*

and that it is not a characteristic of the network stack that Azure uses. The zoomed-in Figure 4.11b shows that also for small message sizes the systems have a similar performance. In general, we can say that for the perftest benchmarks there is no noticeable difference in terms of throughput. This can be seen very clearly in the plots in B.2.4.

4.3.2.3 TCP Multi Stream Throughput

Last we will look again at the iperf tool to measure network throughput. As iperf3 is a single-threaded application we need to manually start multiple streams. To ensure that all 16 streams are running in our interval of measurement, we ran each stream for 30 seconds and only looked at the 10-second interval in the middle (10s to 20s) to take our measurements. With this method, we can avoid a situation where not all streams a running parallel and falsify our results by artificially increasing the measured value of the throughput. We ran 25 iterations of this benchmark to get multiple data points.

Azure and GCE almost reach 100 Gbps with very little variance. AWS shows a little more variance, but also has a noticeably higher throughput for TPC with multiple concurrent streams than with just a single stream as can be seen in Figure 4.12. In this benchmark, the Slingshot based cluster stands out in comparison to the virtualised clusters as it is the only one that does not show a larger improvement with multiple streams in parallel over a single stream. It might be that the former is not optimised for as many streams and thus does not show a significant improvement with that many streams. In general, we can conclude that the virtualised cluster on Azure is closest to the physical cluster while AWS and GCE are closely matched. In the case of the MPI benchmarks, we can reach more stability and slightly higher throughput rates with multiple concurrent streams, except for GCE.

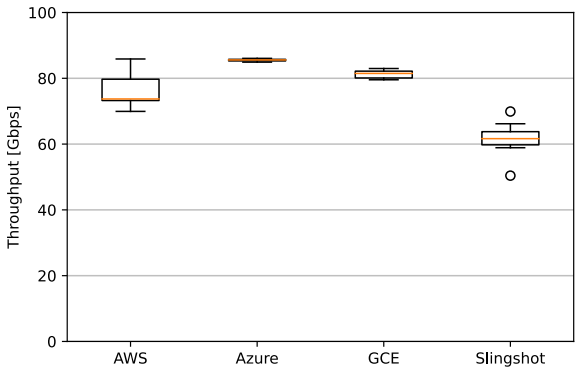


Figure 4.12: TCP point-to-point throughput with 16 parallel streams using two nodes and 1 MiB messages. The box is defined by the first and third quartile while the orange line represents the median. The whiskers extend to $Q1 - 1.5 \cdot IQR$ and $Q3 + 1.5 \cdot IQR$ (IQR=Q1-Q3).

4.3.3 Bidirectional Throughput

Usually, the advertised bandwidth is per direction. The following tests aim to see whether the virtual clusters can reach a bidirectional throughput of around 200 Gbps. We used the *osu_bibw* microbenchmark and *ib_send_bw*, *ib_write_bw* and iperf3 with the flag for bidirectional transfer.

4.3.3.1 Bidirectional MPI Throughput

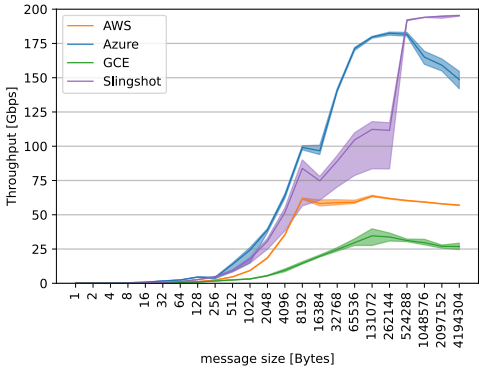


Figure 4.13: Bidirectional MPI point-to-point throughput using two nodes with one process per node

We will again start with the MPI benchmark. GCE performs similar to the unidirectional case and AWS is a bit slower. This is another hint that those two providers do not have their software stacks fully optimised or that their hardware cannot sustain 100 Gbps bidirectionally. Figure 4.13 shows that Azure reaches close to 180 Gbps before the throughput rate drops again, probably due to increased buffering times. The physical cluster is the only

one that reaches the full 200 Gbps and seems to be able to provide this throughput consistently.

4.3.3.2 Bidirectional IB Throughput

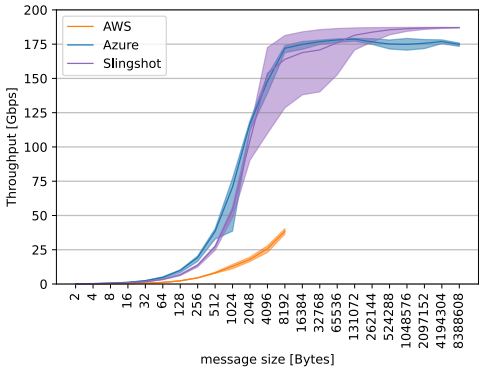


Figure 4.14: Bidirecitonal IB point-to-point throughput using two nodes using *ib_send* with the *–bidirectional* flag

In Figure 4.14 we see that with the elimination of the variabilities that the MPI implementation brings in, Azure is also able to provide its peak throughput rate for bidirectional traffic consistently. Unfortunately, we cannot say much about AWS as the curve is just too short to see something interesting. AWS again has the smallest variance out of the three as seen in previous tests and the Slingshot interconnect based cluster has the largest one for medium-sized messages. Similar to the single-stream unidirectional

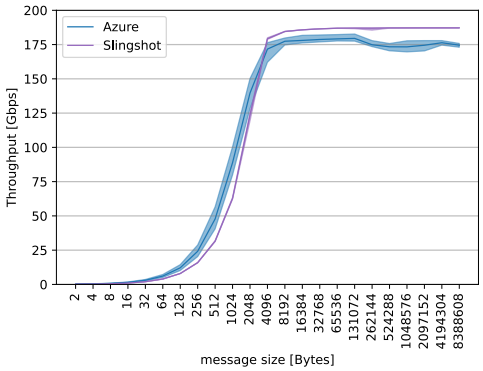
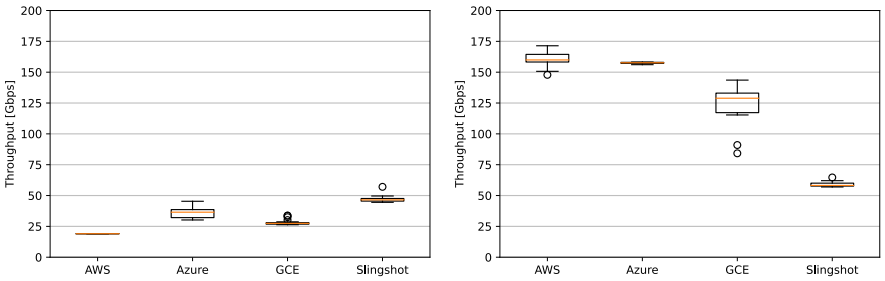


Figure 4.15: Bidirecitonal IB point-to-point throughput using two nodes using *ib_write* with the *–bidirectional* flag

case we can see in Figure 4.15 that the physical cluster again has almost no variability which is most likely due to writing directly to the memory of the second machine and circumventing the CPU in doing so. The virtual cluster shows a bit more variance which is probably due to some virtualization of

the main memory and thus the CPU cannot be completely bypassed. But again these two platforms have similar performance.

4.3.3.3 Bidirectional TCP Throughput



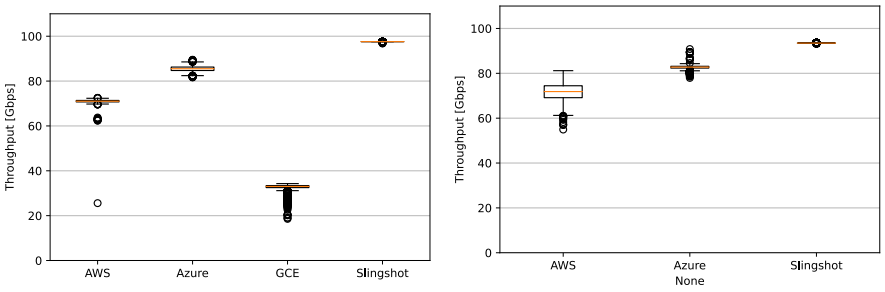
(a) Single stream **(b)** 16 parallel streams

Figure 4.16: Bidirectional TCP point-to-point throughput using two nodes and 1 MiB messages. The box is defined by the first and third quartile while the orange line represents the median. The whiskers extend to $Q1 - 1.5 \cdot IQR$ and $Q3 + 1.5 \cdot IQR$ (IQR=Q1-Q3).

We used the *–bidir* flag for iperf3 to evaluate the TCP performance and performed 25 iterations. In Figure 4.16a we can see that the single-stream performance is no higher than for the unidirectional test with only one stream. But in Figure 4.16b we can see some interesting things. First, we note the odd behaviour of the physical cluster. Its throughput is no higher than it was for the multiple stream unidirectional test. To eliminate the chance of it being an exception due to network congestion we reran the test a second time on another day and got the same overall picture. This could again be to due to the fact that the network stack is not optimized for this many streams. AWS and Azure have similar performance with Azure having a bit less variance. The cluster on GCE has a lower throughput, but also considerably higher than for the unidirectional test. This strengthens our assumption that GCE and AWS do not have fully optimized software stacks as their hardware is capable of providing high transmission rates or using multiple NIC's in the case of GCE.

4.3.4 Throughput Stress Test

In this last test we want to evaluate how the clusters handle a sustained load over for 15 minutes. To this end we only used the *osu_bw* and *ib_send_bw* benchmarks as it did not make sense to us to run the *ib_write_bw* benchmark as it showed such similar performance to the aforementioned *ib_send_bw* benchmark. In Figure 4.17a we see that Google has the lowest performance out of the four clusters under scrutiny and it also has the most outliers. The other two virtualised clusters seem to hold their throughput consistently



(a) MPI

(b) IB,for AWS only 8 KiB packets

Figure 4.17: Stress Test for roughly 15 minutes with 1 MiB packets. The box is defined by the first and third quartile while the orange line represents the median. The whiskers extend to $Q1 - 1.5 \cdot IQR$ and $Q3 + 1.5 \cdot IQR$ (IQR=Q1-Q3).

over the 15 minutes with only a few outliers. The physical cluster has by far the most consistent throughput over the 15 minutes. In the *ib_send_bw* the Slingshot interconnect based cluster again has the highest and most consistent throughput albeit begin a bit lower. The clusters on AWS and Azure show more variance and outliers as can be seen in Figure 4.17b.

4.4 Hoverboard Analysis

As we show in Section 3.1.1 Google relies on the Hoverboard Programming Model with the Hoverboard- and the direct path to bypass it. If a lot of data is sent between two virtual machines a direct route between these two VM's is established, such that the traffic does not have to go through the dedicated Hoverboard Gateway. The goal of that is to get a more stable connection for applications that depend on a stable low latency and high throughput connection.

4.4.1 Testing Method / Benchmark

To see how Googles Hoverboard Programming Model behaves and performs we needed to create our own test as this is not a standard behaviour and not something that is normally tested. In this section, we will explain how our test works.
To control the usage (how extensively a connection is used) in the test we use busy waiting[5] to "sleep" for certain amounts of time to control the amount of data that is sent and to simulate bursty and continuous flows. We use different levels of burstiness (different sleep times: 1s, 1ms, 1μs, 1ns, 0s) as we do not know what the threshold would be. To be able to plot the results and not only have single data points, we took 30 measurements per sleep

[5]Busy waiting is a technique of active sleeping. Instead of calling a function to sleep like *usleep* which entails a context switch, the process is kept in a loop until the amount of time to sleep for has passed.

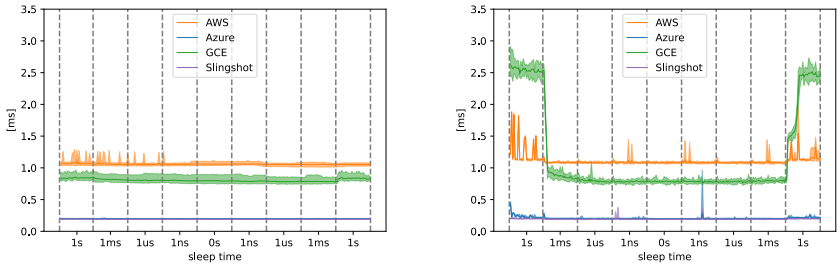
Algorithm 1 Hoverboard Test(X, *repetitions*)

```
sleeptimes ← [1s, 1ms, 1μs, 1ns, 0s, 1ns, 1μs, 1ms, 1s]
for s ∈ sleeptimes do
  for i ← 1 to 30 do
    t_start ← current_time
    for j ← 1 to repetitions do
      if rank is 0 then
        send X bytes
        receive X bytes
      else
        receive X bytes
        send X bytes
      end if
    end for
    t_stop ← current_time
    latency ← (t_stop − t_start)/repetitions
    sleep(s)
  end for
end for
```

time and decreased the sleep time progressively. We were also interested to see whether this direct connection between the two VM's would be installed permanently after it gets triggered. Thus after we arrived at the lowest sleep time (0s) we progressively increased the sleep time again such that we can see if the latency increases again or if it stays at the same lower level as if we would send data in a continuous stream. To simulate bursts of small continuous flows we added the option to have a different number of iterations (we call it repetition) between sleeping. We used the blocking *MPI_Send()* and *MPI_Recv()* methods of MPI to send and receive data. Algorithm 1 gives a high-level overview of the test and the full source code can be found in Appendix A.

4.4.2 Results

In this section we will use the aforementioned benchmark that is described in Section 4.4.1 to evaluate the behaviour and performance. In Figure 4.18a we executed 100 iterations over which we averaged between the sleeping and in Figure 4.18b we executed only a single iteration between sleeping. Additionally we ran the test 10 times and plotted the mean with a band around it that is given by the minimum and maximum. Per sleep time interval 30 iterations of the above procedure are executed.
To be sure that any detected behaviour has nothing to do with virtualisation, we also ran the test on the other two cloud solution provider as well as on the



(a) 100 repetitions

(b) 1 repetitions

Figure 4.18: Hoverboard Test with sequentially sent 1MiB packets

non-virtualised cluster. We can see that for AWS, Azure and the Slingshot interconnect based cluster the results of the two tests look similar, despite what we assume is some network noise and more distinct in Figure 4.18b as it is only a single run and not averaged over 100 runs. The interesting case is the latency plot for GCE. In 4.18a, where we send 100 1 MiB packets after each other without sleeping, we see that the latency is the same throughout the whole execution around 0.8 ms. But in 4.18b where we only send one packet between sleeping, we can see that the latency changes throughout the execution with different sleep times. First, we have a latency of roughly 1.8 ms and after that, we only have a latency of 0.8 ms, the same as in the case where we have a continuous flow of data. Additionally, there is more variance in the data if we sleep for a whole second. As we have not reverse-engineered their architecture and software we can only assume that this change in latency is due to the change in path that the packets take. In other words, we think that the flow exceeds the necessary usage threshold to trigger the ”installation” of a direct path between the two VM’s as described by Google [20]. And thus the delay and variance that stems from processing the packets in the dedicated Hoverboard Gateway to determine its path are gone. Instead, the two VM’s now each have an entry in their routing table of the other VM. It is interesting to note that as soon as the rate of traffic gets lower, the latency increases again. This means that those paths are indeed dynamic as they say in their paper [20] about their *Andromeda* architecture. We can conclude that on GCE, applications that have bursty traffic, will have a higher latency than applications that have a continuous flow of data. It is worth taking note that again the Azure cluster performs the most similar to a physical cluster with almost identical latency.

4.5 Operating System Noise

In this section, we analyse the operating system noise on the different systems as this also has an impact on the rate at which network traffic can be

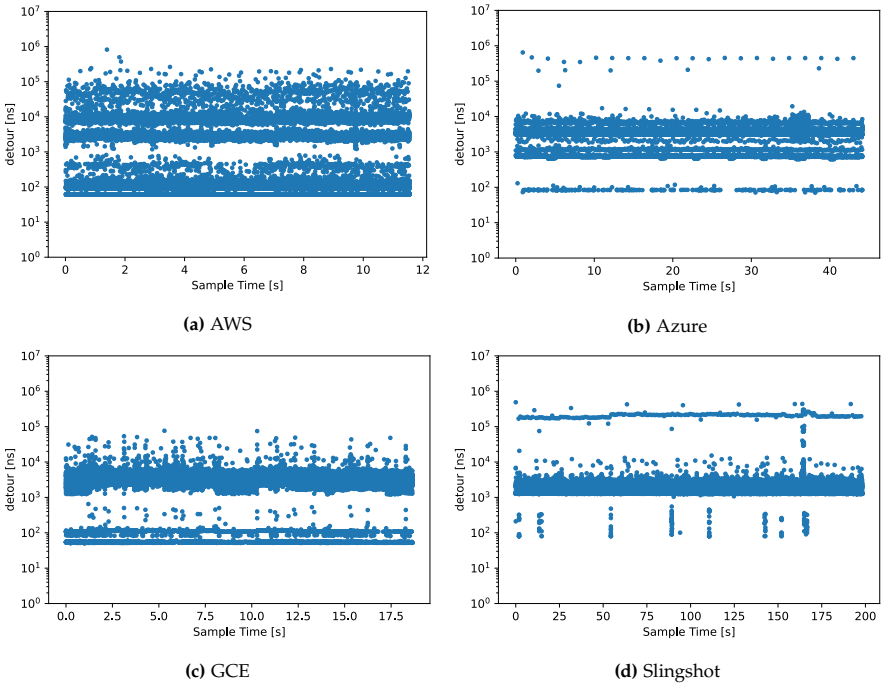
Platform	OS	t_{min}	Overhead	Interval
AWS c5n.18xlarge	CentoOS 7	7.50 ns	1.95%	11.55 s
MS AzureStandard HC44rs	CentoOS 7	7.42 ns	0.37%	44.14 s
GCP N2-standard-80	CentoOS 7	6.43 ns	0.73%	18.66 s
Slingshot interconnect based cluster	SUSE Linux Enterprise Server 15 SP1	9.80 ns	0.07%	198.26 s

Table 4.2: System / Node and its operating system, t_{min} represents the minimum loop time (measurement accuracy), serial noise overhead and the interval is the time it took to reach the 10^5 detours (iterations that took longer than the threshold)

sent. If the data/packets cannot be processed fast enough, we cannot receive all data and thus it needs to be resent or we cannot send everything in the first place. We used the *Netgauge* benchmark suite for this test, more specifically its noise test. This test computes a fixed amount of work and measures how long it took to perform this computation. If this time is above a certain threshold, this specific execution is considered a selfish detour and is stored. Only measurements that are higher than the threshold are stored to manage the huge number of measurements. The threshold is set relative to t_{min} which is the optimal, noiseless execution time on the specific system. This means that the benchmark can be run on a wide variety of systems without the need to manually adjust t_{min}. The threshold is chosen as $9 \cdot t_{min}$ to filter out cache misses that happen due to storing the selfish detours. t_{min} is evaluated in a separate step, outside the benchmark loop to improve the sampling frequency. The benchmark is run until 10^5 detours are registered. For more detailed information and explanations on the design, choices see [25]. In Table 4.2 the different t_{min}'s along the operating system that was used while running this benchmark are listed. Moreover, the interval shows the time it took for the benchmark to reach the stopping condition of 10^5 detours. The Overhead shows the ratio of the execution of the benchmark loop that took longer than the threshold and the total number of times it was run:

$$\text{Overhead} = \frac{\text{Selfish Detours}}{\text{Total \#executions}}. \tag{4.1}$$

This determines how much noise the systems encountered. The significantly higher t_{min} of the Slingshot cluster is most likely due to the lower frequency of the AMD EPYC CPU that it runs and has nothing to do with noise on the system. Figure 4.19 shows the scatter plots of the recorded detours on the four systems we evaluated. The AWS cluster hast the most noise out of the four systems as can be seen in Table 4.2. Figure 4.19a shows that most detours appear to lie around $0.1\mu s$ and between $1ns$ and $10ns$, but there is no periodic pattern and a lot of random noise. In Figure 4.19b we see that the most detours lie between $1ns$ and $10ns$, but with a lot less random noise than for AWS. There is again no periodic detours, but more regular noise at $0.1\mu s$ and some random detours that take $1ms$. GCE has noticeably more random noise, but again most of the detours are within between $1ns$



(a) AWS

(b) Azure

(c) GCE

(d) Slingshot

Figure 4.19: Scatter plot of 10^5 detours on the different systems

and $10ns$. In contrast to Azure, Figure 4.19c shows that we have two more detour lengths around $0.1\mu s$ that occur regularly. The slingshot based cluster is the only one that shows some periodic noise, but it does not occur in a regular pattern. Figure 4.19d additionally shows that the fourth cluster also has the most noise between $1ns$ and $10ns$, but also that detours at around $10\mu s$ occur more regularly.
We conclude that most of the noise for all four systems lies between $1ns$ and $10ns$, but with different amounts of random noise. Thus we conclude that none of the systems has low regular noise. Nonetheless, the non-virtualised cluster has by far the least amount of noise and none of the virtualised clusters, not even Azure, come close to it. In terms of random noise, GCE fares not much worse than Azure and AWS performs the worst in terms of random noise occurring on the system. But again, Azure performs the best and has the lowest total noise out of the virtualised clusters followed by GCE and AWS.

4.6 Performance per Dollar

As we have seen in the previous chapters the network performance between the three providers for HPC cloud solutions differs significantly. In this last comparison, we want to evaluate whether the price is correlated to the

	Hourly Price	Region	Provider	Cores	memory	Bandwidth
c5n.18xlarge	3.888 USD	US East (Ohio)	Amazon	36	192 GiB	100 Gbps
c5n.9xlarge	1.944 USD	US East (Ohio)	Amazon	18	96 GiB	50 Gbps
c6gn.9xlarge	2.7648 USD	US East (Ohio)	Amazon	32	128 GiB	100 Gbps
HC44rs	3.168 USD	US East	Microsoft	44	352 GiB	100 Gbps
n2-standard-80	3.88472 USD	US Central (Iowa)	Google	40	320 GiB	100 Gbps

Table 4.3: Hourly price of different virtual machines. Additionally, important parameters that have an influence on the price are included.

performance and whether it is justified. This analysis is partly based on our experience encountered during our evaluation phase. On AWS there is no additional cost for the EFA and ENA and it is included in the hourly price of the virtual machine. This can be seen in Table 4.3, the price doubles, if the *c5n.18xlarge* instances is used instead of the *c5n.9xlarge* but so do the number of cores and the amount of memory. But notice that not both VM's have a 100 Gbps bandwidth. The same holds for Azure where the *Accelerated Networking* does not increase the hourly price of the virtual machine. On GCE instead, the billing works differently. If one chooses to use the *Tier 1* networking to get higher bandwidth, the hourly price of the VM is increased [29]. On all three providers, the hourly cost per VM is highly dependent on the region they are run in, this difference can be more than a dollar per VM per hour.
Our clusters existed during a period of approximately three months and our bills show some interesting things. On Azure one pays a significant amount for the persistent storage that is used to store the cluster. The storage needed to store the cluster is negligible on AWS, but on the other hand, the time that a network address translation gateway is running can make up a significant amount of the total bill. Especially, if the cluster is only used infrequently. On GCE the costs to run the virtual machines make up the largest part of the bill and there are no significant additional costs.
From the cost of the virtual machines, Azure delivers to most performance. The difference in performance compared to the other providers justifies the additional cost that stems from the used persistent memory. Considering the costs in Table 4.3 AWS and Google deliver similar performance for a similar price. But Amazon offers instances with its in-house designed *AWS Graviton 2* ARM-based [2] processors at a cheaper price in all regions. We did not use them despite their support for 100 Gbps networking because a similar CPU platform was important to us. Additionally, the costs can be reduced if not 100 Gbps networking is needed because cheaper instances can be used which holds for all providers. Moreover, Google offers customers

the possibility to design their own virtual machines with the amount of memory and processing cores that are needed. Only the used resources are billed which can lead up to a 40% decrease in cost depending on the used instance family.

Chapter 5

Conclusion and Outlook

In our evaluation of the different cloud providers, we saw that Microsoft uses mostly components that are also used to build traditional physical high performance computing clusters. Amazon and Google each built a custom solution for their cloud services which is also used for high performance instances. Both of them try to provide a highly specialised infrastructure that is suited for tightly coupled workloads that need a stable connection with low latency and high throughput. All three providers have instances on offer that are built for compute-intensive workloads, but the ones of Google do not support 100 Gbps bandwidth, instead GCE offers general-purpose instances that provide 100 Gbps networking.
These different design choices are reflected in the results we analysed in the evaluation phase. Azure has almost identical performance to the real cluster in all aspects that we tested. The area where it lacks the most is OS noise, but it is still significantly better than AWS, which has the most OS noise, and GCE. While AWS and GCE lack some performance in general, we suspect that their software stack is not fully optimized as we need multiple concurrent streams to get the most out of the corresponding instances. This can be seen best in the comparison between single and multi-stream unidirectional and bidirectional communication. We were able to show a performance difference between bursty and continuous flows on GCE instances which supposedly stems from their architecture. The Hoverboard Programming Model directs all flows over a dedicated gateway, but for communication-intensive applications, this additional latency gets circumvented by ”installing” a direct path between the two instances.
As mentioned before Azure performs the closest to a non-virtualised cluster. At a competitive price, Microsoft offers also the most similar hardware to what is used in non-virtualised clusters. Amazon and Google also provided reasonable performance at the same price, but both do not fully support the IB modes of transport. Thus even if no IB verbs are used, Azure is the

preferred provider for an HPC cluster in the cloud as it is similarly priced to the other two providers but performs significantly better.

On Azure the HPC optimized instances that we used run as dedicated instances by default. Due to the aforementioned limitations of the quota we were not able to run the instances on GCE and AWS in single-tenancy. Interesting future work would be to run benchmarks on instances that run in single tenancy on all providers to see if the performance gap can be reduced or even closed that exists between Azure and the other two providers. This could show if the performance difference or at least some of it in comparison to Azure and the non-virtualised cluster is due to sharing of hardware. Additionally, clusters with instances that provide a higher network bandwidth could be built on AWS and Azure to see how they fare. Can they provide the same relative performance as they did with the 100 Gbps instances? Moreover, collective operations that MPI provides such as *MPI_Scatter* or *MPI_Gather* can be analysed with larger clusters which we could not build do due to the quota limitations described in Section 3.2.3.

Appendix A

Hoverboard Test code

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <stdint.h>
#include <time.h>

#define add_time(now, expected, s, ns)           \
    {                                            \
        expected.tv_sec = now.tv_sec + add_s;    \
        expected.tv_nsec = now.tv_nsec + add_ns; \
    }
#define cmp_time_leq(time1, time2) (time1.tv_sec < time2.tv_sec ?
     1 : (time1.tv_sec == time2.tv_sec && time1.tv_nsec < time2.
    tv_nsec ? 1 : 0))

int runner(long int message_size, int repetitions, FILE *datafile
    );

struct timespec {
    time_t tv_sec;        /* seconds */
    long   tv_nsec;       /* nanoseconds */
};

int main(int argc, char *argv[])
{
    if (argc != 4)
    {
        printf("not enough args provided [msg_size, time, #
            repetitions, output file name\n");
        exit(0);
    }

    // parse arguments
    long int msg_size = atoi(argv[1]);
    int repetitions = atoi(argv[2]);
    char *datafile_name = argv[3];
```

```
    FILE *datafile = fopen(datafile_name, "w");
    int returncode = runner(msg_size, repetitions, datafile);

    fclose(datafile);
    return 0;
}

/**
 * runs the Hoverboard Test (MPI send/recv loop)
 *
 * @param message_size [bytes]
 * @param repetitions for how many times we send the message
    before we sleep
 * @param datafile file to write output to (opened as write file)
 */
int runner(long int message_size, int repetitions, FILE *datafile
    )
{
    int my_rank, num_procs;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Status stat;
    // check if we have two processes
    if (num_procs != 2)
    {
        if (my_rank == 0)
        {
            printf("We need two processes\n");
        }
        MPI_Finalize();
        exit(0);
    }

    // allocate buffer
    uint8_t *s_buf = (uint8_t *)malloc(message_size * sizeof(*
        s_buf));
    uint8_t *r_buf = (uint8_t *)malloc(message_size * sizeof(*
        r_buf));

    // set values in buffer to zero
    for (int i = 0; i < message_size * sizeof(*s_buf); i++)
    {
        s_buf[i] = (uint8_t)0;
        r_buf[i] = (uint8_t)0;
    }

    int tag_send0 = 42;
    int tag_send1 = 43;

    //write header
    if (my_rank == 0)
    {
```

```
        fprintf(datafile, "Hoverboard test - Increasing sleep
            time (%ld bytes)\n", message_size);
        fprintf(datafile, "time, BW [MB/s], message size,
            iterations, sleep time\n");
        fflush(datafile);
    }

    double start, end, measured_time, total_start, total_end, tmp
        ;
    long add_s, add_ns;
    struct timespec now, expected;

    double sleep_times[9] = {1, 1e-3, 1e-6, 1e-9, 0, 1e-9, 1e-6,
        1e-3, 1};

    total_start = MPI_Wtime();

    // For each sleep time we execute 30 iterations of the inner
        loop. In the inner loop we execute #repetition many
        iterations of senc/recv
    for (int l = 0; l < sizeof(sleep_times) / sizeof(sleep_times
        [0]); l++)
    {
        double sleep_time = sleep_times[l];
        double sleep_time_ns = sleep_time * 1e9;
        printf("rank: %d, sleep time: %.9f\n", my_rank,
            sleep_time);
        fflush(stdout);
        for (int i = 0; i < 30; i++)
        {
            start = MPI_Wtime();
            for (int j = 0; j < repetitions; j++)
            {
                if (my_rank == 0)
                {
                    MPI_Send(s_buf, message_size * sizeof(*s_buf)
                        , MPI_UINT8_T, 1, tag_send0,
                        MPI_COMM_WORLD);
                    MPI_Recv(r_buf, message_size * sizeof(*r_buf)
                        , MPI_UINT8_T, 1, tag_send1,
                        MPI_COMM_WORLD, &stat);
                }
                else
                { // rank == 1
                    MPI_Recv(r_buf, message_size * sizeof(*r_buf)
                        , MPI_UINT8_T, 0, tag_send0,
                        MPI_COMM_WORLD, &stat);
                    MPI_Send(s_buf, message_size * sizeof(*s_buf)
                        , MPI_UINT8_T, 0, tag_send1,
                        MPI_COMM_WORLD);
                }
            }
            end = MPI_Wtime();
            // write the results to the output file
```

```
            if (my_rank == 0)
            {
                measured_time = end - start;
                tmp = repetitions * 2 * message_size / 1e6;
                fprintf(datafile, "%f, %f, %ld, %d, %.9f\n",
                    measured_time, tmp / measured_time,
                    message_size, repetitions, sleep_time);
                fflush(datafile);

                // busy waiting
                if (sleep_time_ns >= 1e9)
                {
                    add_s = sleep_time_ns / 1e9;
                    add_ns = (long)sleep_time_ns % (long)1e9;
                }
                else
                {
                    add_s = 0;
                    add_ns = sleep_time_ns;
                }

                clock_gettime(CLOCK_REALTIME, &now);
                add_time(now, expected, add_s, add_ns);
                while (cmp_time_leq(now, expected))
                {
                    clock_gettime(CLOCK_REALTIME, &now);
                }
            }
        }
    }

    total_end = MPI_Wtime();

    // print the total time it took to complete the test into the
        output file
    if (my_rank == 0)
    {
        printf("\nTotal time needed to run the test: %.2f seconds
            \n", total_end - total_start);
    }

    free(s_buf);
    free(r_buf);
    MPI_Finalize();
    return 0;
}
```

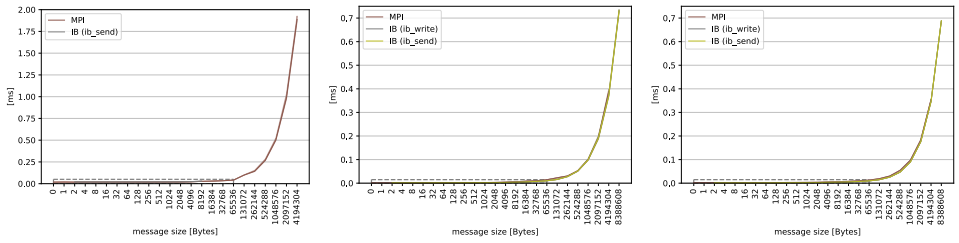
Appendix B

Additional Plots

In this appendix are additional plots that were omitted in the thesis. They they show the different layers of the network stack in a single plot per provider. This is done with the aim to see the differences between MPI, TCP and IB performance. As note all benchmarks run for the same message sizes in their default ”run all message sizes” the lines do not start and end at the same message sizes for all benchmarks. Note that we often have different ordinate axis scaling to depict the differences between the different network stack layers better.

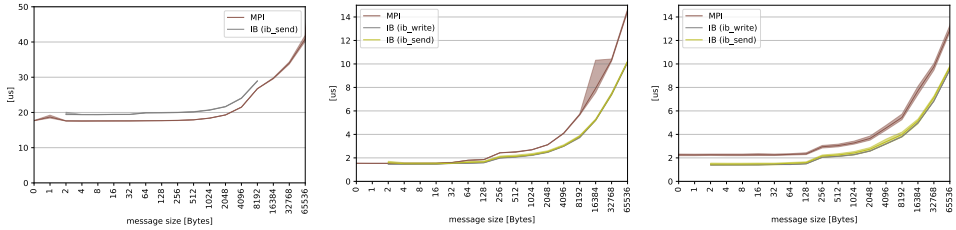
B.1 Per Provider Latency Plots

In this section we have additional latency plots. We omit the multi-stream benchmark and Google as only a single benchmark remains without the mutli-stream MPI evaluation.



(a) AWS **(b)** Azure **(c)** Slingshot

Figure B.1: Per provider latency plots.



(a) AWS **(b)** Azure **(c)** Slingshot

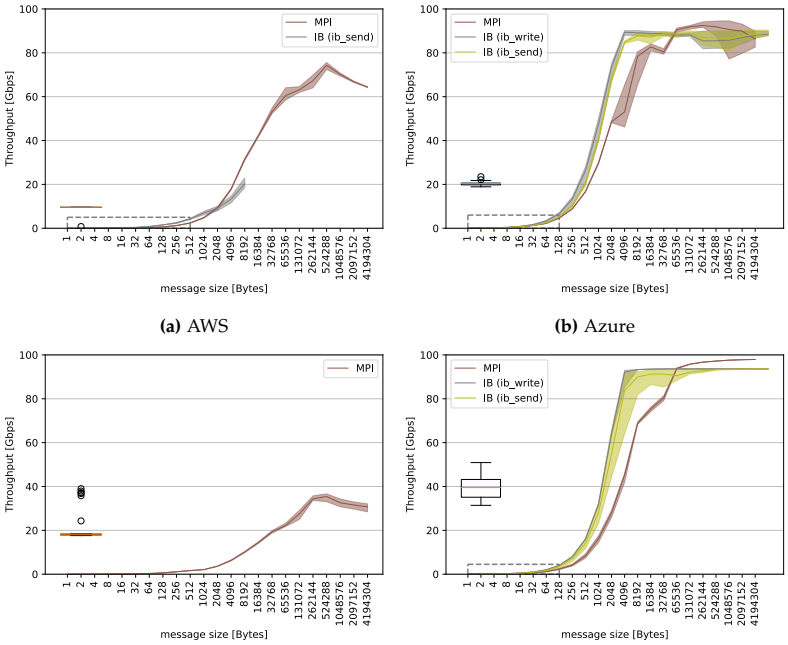
Figure B.2: Per provider latency plots, zoomed in on the dashed rectangle of Figure B.1

B.2 Per Provider Throughput Plots

In this section of the appendix B we have the per provider plots for the different throughput microbenchmarks. The boxplot represents the throughput of the TCP benchmarks and we do not provide in zoomed in plots for GCE as there is nothing to compare the single MPI benchmark to. The box in the boxplots is defined by the first and third quartile while the orange line represents the median and the whiskers extend to $Q1 - 1.5 \cdot IQR$ and $Q3 + 1.5 \cdot IQR$ ($IQR = Q1 - Q3$).

B.2.1 Single Stream Unidirectional Throughput

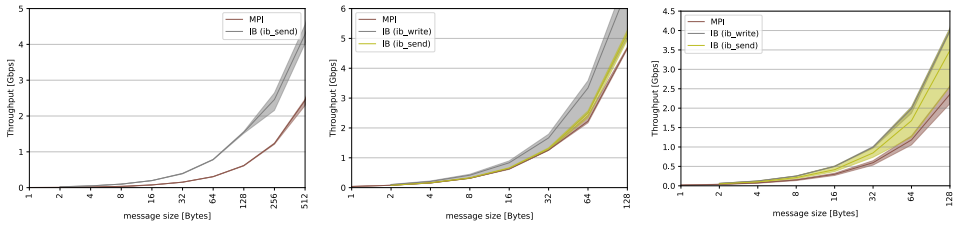
In this section the per provider plots for single stream unidirectional communication can be found.



(a) AWS **(b)** Azure

(c) GCE **(d)** Slingshot

Figure B.3: Per provider single-stream throughput plots

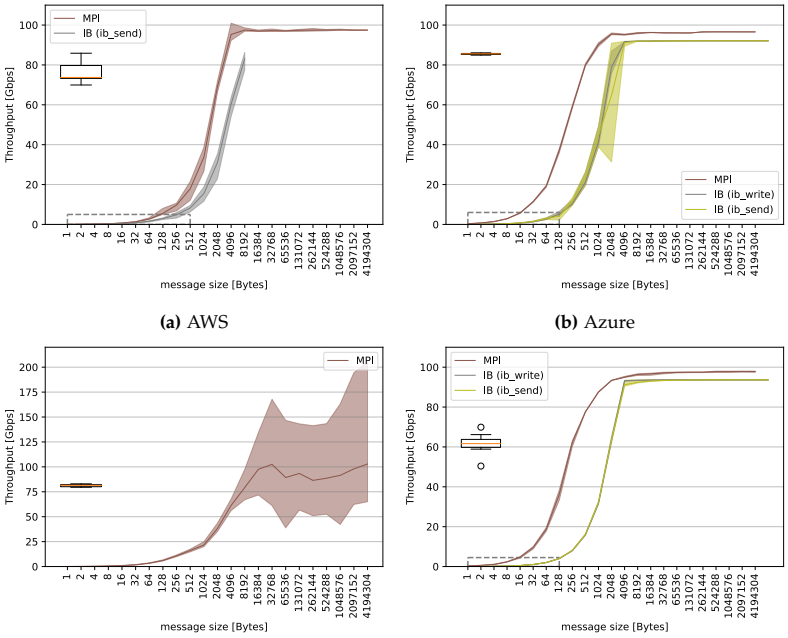


(a) AWS **(b)** Azure **(c)** Slingshot

Figure B.4: Per provider single-stream throughput plots, zoomed in on the dashed rectangle of Figure B.3.

B.2.2 Multiple Stream Unidirectional Throughput

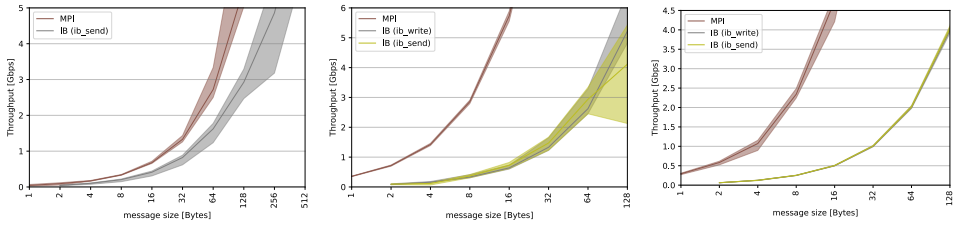
Here we have the plots to compare the throughput of MPI, TCP and IB with 16 parallel streams for the different providers. Note that we have a different ordinate scale for GCE.



(a) AWS **(b)** Azure

(c) GCE **(d)** Slingshot

Figure B.5: Per provider multi-stream throughput plots for multiple streams.

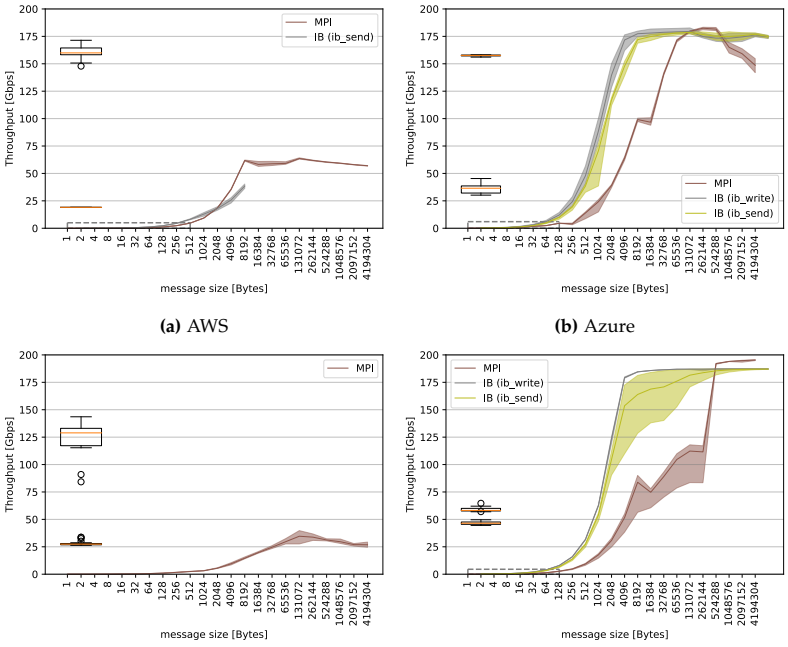


(a) AWS **(b)** Azure **(c)** Slingshot

Figure B.6: Per provider multi-stream throughput plots for multiple streams, zoomed in on the dashed rectangle of Figure B.5

B.2.3 Bidirectional Throughput

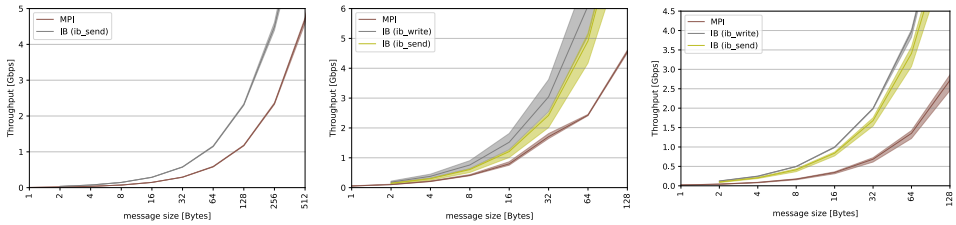
This section of the appendix compares the bidirectional throughput per provider. Note that we have two TCP boxplots, the upper one is the multi-stream run while the lower is the benchmark with a single stream.



(a) AWS **(b)** Azure

(c) GCE **(d)** Slingshot

Figure B.7: Per provider bidirectional throughput plots.

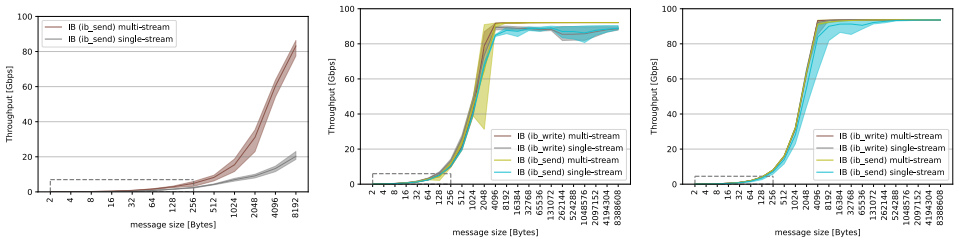


(a) AWS **(b)** Azure **(c)** Slingshot

Figure B.8: Per provider bidirectional throughput plots, zoomed in on the dashed rectangle of Figure B.7.

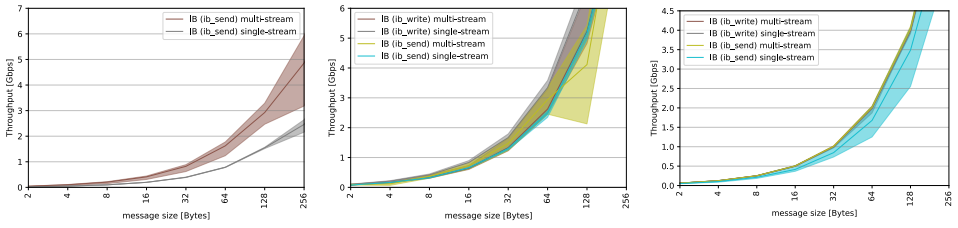
B.2.4 IB Single- vs Multi-Stream Throughput

In this section we have the plots to compare the single and 16 stream unidirectional throughput of the different IB *uverbs ib_send* and *ib_write*.



(a) AWS **(b)** Azure **(c)** Slingshot

Figure B.9: Per provider single- / multi-stream IB throughput plots



(a) AWS **(b)** Azure **(c)** Slingshot

Figure B.10: Per provider single- / multi-stream IB throughput plots, zoomed in on the dashed rectangle of Figure B.9.

Bibliography

[1] Amazone Elastic Compute Cloud. https://aws.amazon.com/de/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc.

[2] ARM. https://www.arm.com/.

[3] Centro Svizzero di Calcolo Scientifico (CSCS). https://www.cscs.ch.

[4] Cori Supercomputer. https://www.nersc.gov/systems/cori/.

[5] iperf3 Github. https://github.com/esnet/iperf.

[6] Libfabric. https://ofiwg.github.io/libfabric/.

[7] Microsoft Azure. https://azure.microsoft.com/de-de/.

[8] MVAPICH. https://mvapich.cse.ohio-state.edu/.

[9] OpenMPI. https://www.open-mpi.org/.

[10] OSU Benchmark. https://mvapich.cse.ohio-state.edu/benchmarks/.

[11] NVIDIA Corporation & affiliates. HPC-X. https://developer.nvidia.com/networking/hpc-x.

[12] NVIDIA Corporation & affiliates. Nvidia Collective Communication Library (NCCL). https://developer.nvidia.com/nccl.

[13] NVIDIA Corporation & affiliates. Perftest. https://community.mellanox.com/s/article/perftest-package and https://github.com/linux-rdma/perftest.

[14] NVIDIA Corporation & affiliates. Unreliable Datagram Transport Mode. https://docs.mellanox.com/display/RDMAAwareProgrammingv17/Transport+Modes.

[15] Inc. Amazon Web Services. Amazon software stack. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa.html.

[16] Inc. Amazon Web Services. AWS Parallel Cluster. https://aws.amazon.com/hpc/parallelcluster/.

[17] Sourav Chakraborty, Shulei Xu, Hari Subramoni, and Dhabaleswar Panda. Designing Scalable and High-Performance MPI Libraries on Amazon Elastic Fabric Adapter. In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 40–44, 2019.

[18] Intel Corporation. Intel MPI. https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/mpi-library.html.

[19] Intel Corporation. Intel oneAPI Toolkits Installation Guide for Linux* OS. https://software.intel.com/content/www/us/en/develop/documentation/installation-guide-for-intel-oneapi-toolkits-linux/top/installation/install-using-package-managers/yum-dnf-zypper.html.

[20] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 373–387, Renton, WA, April 2018. USENIX Association.

[21] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure Accelerated Networking: SmartNICs in the Public Cloud. In

15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pages 51–66, Renton, WA, April 2018. USENIX Association.

[22] Giulia Guidi, Marquita Ellis, Aydin Buluç, Katherine Yelick, and David Culler. 10 Years Later: Cloud Computing is Closing the Performance Gap. *Companion of the ACM/SPEC International Conference on Performance Engineering*, Apr 2021.

[23] HashiCorp. Terraform. `https://www.terraform.io/`.

[24] Torsten Hoefler, Torsten Mehlan, Andrew Lumsdaine, and Wolfgang Rehm. Netgauge: A Network Performance Measurement Framework. In *Proceedings of High Performance Computing and Communications, HPCC'07*, volume 4782, pages 659–671. Springer, Sep. 2007.

[25] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Characterizing the Influence of System Noise on Large-Scale Applications by Simulation. In *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2010.

[26] Google LLC. Cloudy Cluster. `http://cloudycluster.com/`.

[27] Google LLC. Google Compute Engine. `https://cloud.google.com/compute`.

[28] Google LLC. Multiple Streams on GCE. `https://cloud.google.com/compute/docs/networking/benchmarking-higher-bandwidth-vms#performing_the_benchmark`.

[29] Google LLC. Price of Tier 1 networking on GCE. `https://cloud.google.com/compute/all-pricing#high_bandwidth_configuration`.

[30] Google LLC. TCP window size. `https://cloud.google.com/architecture/tcp-optimization-for-network-performance-in-gcp-and-hybrid`.

[31] Google LLC. Use Intel MPI 2018 and MPI collective tunings. `https://cloud.google.com/compute/docs/instances/create-hpc-vm#tune`.

[32] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing, 2011-09-28 2011.

[33] Microsoft. Accelerated Networking. `https://docs.microsoft.com/en-us/azure/virtual-network/create-vm-accelerated-networking-cli`.

[34] Microsoft. Azure Batch. https://azure.microsoft.com/en-gb/services/batch/.

[35] Microsoft. Azure Cycle Cloud. https://azure.microsoft.com/en-gb/features/azure-cyclecloud/.

[36] Rusty Russell IBM Corp oration (Editor). Virtio PCI Card Specification v0.9.5 DRAFT, May 2012. https://ozlabs.org/~rusty/virtio-spec/virtio-0.9.5.pdf.

[37] T. Rahn, D. De Sensi, K. Taranov, and T. Hoefler. Thesis Repository. https://spclgitlab.ethz.ch/tobiasrahn/noise-estimation-in-hpc-cloud-networks.git.

[38] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler. An In-Depth Analysis of the Slingshot Interconnect, 2020.

[39] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro*, 40(6):67–73, 2020.

[40] Paul Teich. Vertical Integration Is Eating The Datacenter, Part Two. *www.nextplatform.com*, 2020. https://www.nextplatform.com/2020/02/03/vertical-integration-is-eating-the-datacenter-part-two/.

[41] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. Is Big Data Performance Reproducible in Modern Cloud Networks? In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 513–527, Santa Clara, CA, February 2020. USENIX Association.

[42] Shulei Xu, S. Mahdieh Ghazimirsaeed, Jahanzeb Maqbool Hashmi, Hari Subramoni, and Dhabaleswar K. Panda. MPI Meets Cloud: Case Study with Amazon EC2 and Microsoft Azure. In *2020 IEEE/ACM Fourth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware (IPDRM)*, pages 41–48, 2020.

BSc thesis: Noise Estimation in HPC Cloud Networks

Tobias Rahn
March 5, 2021

Advisor: Daniele De Sensi (ddesensi@ethz.ch)
Professor: Prof. Dr. T. Hoefler
Start time: 08.03.2021
End time: 08.09.2021

Introduction

In the last years, all the major cloud providers started designing and deploying solutions for running HPC workloads in the cloud. However, a detailed evaluation of such solutions is still missing. In particular, it is not clear if and by how much the workloads running on this ”HPC Clouds” would be affected by network congestion (also known as network noise). Network noise could lead to application unbalance, and this is a problem for tightly synchronized applications such as deep neural networks training, but also for HPC applications in general. The main idea is to analyze network performance and variability, by using both microbenchmarks and real HPC and Cloud applications, similar to what has recently been done for an HPC interconnection network. The impact of different software stacks (MPI, TCP, QUIC, etc...) also needs to be analyzed. The project will be particularly focused on the HPC solutions provided by the major cloud providers (Amazon EC2, Microsoft Azure, and Google GCE). There are major network software stack differences between providers. For example, Amazon bypasses the OS through libfabric, Google can offload part of the network processing to the NIC, and Microsoft relies on the Mellanox/InfiniBand stack, and these need to be compared, by analyzing PROs and CONs of each of them. Moreover, because part of the protocol processing might run in the OS, we also need to estimate/correlate the impact of OS noise on network performance, for example by using netgauge.

Project description

1. Review the main HPC solutions provided by the largest cloud providers (Amazon, Microsoft, and Google), understand and describe their main differences in terms of the network stack and if and what type of mechanisms they have in place to deal with congestion and noise.
2. Define a set of microbenchmarks and real applications to assess the performance of such clusters and their sensitivity to network noise. A starting point could be the microbenchmarks used for the analysis of the Slingshot interconnect. This step also

includes to estimate the amount of compute hours (and the cost) needed to run the microbenchmarks.
3. Set up a small HPC cluster on each of these platforms to run the aforementioned microbenchmarks and applications.
4. Analyze and visualize the collected data and compare the three different providers between them, and also against a non-virtualized HPC system. By either running the same set of experiments or by using already available data (e.g. data from the Slingshot paper).

Milestones / project plan

The goal is to be done in 24 weeks.
I will proceed in the order of the points given above. First I will look into the solutions of the three HPC cloud providers to analyze them in general and especially to get a deep understanding of their network stack. Then I will continue by defining microbenchmarks and looking for real applications to run on these clusters to analyze their performance. After that, I will setup a small cluster on all three services to run the aforementioned microbenchmarks and applications. The last step is to visualize the data and compare the solutions of the three different cloud providers Amazon, Google and Microsoft against each other and non-virtualized HPC systems.
I reserved the most time for task two, as analyzing the system and defining the microbenchmarks, that could outline weaknesses, is the core of this project. The work done in points 2. and 4. is / could be an iterative process of defining microbenchmarks, running and analyzing them. It thus may happen that I have to redefine and rerun the benchmarks because something has to be done differently, that I only realized when I analyzed the results. I want to reserve some time at the end to finish up the final report and to tie up loose ends.

week	**task**
1	administration and read first papers
2-3	review of HPC cloud solutions
4-5	in-depth analysis of the network stack
6-12	define microbenchmarks and estimate cost
13-14	set up small HPC cluster
15-17	run microbenchmarks and real applications
18-21	analyze and visualize data to compare the three providers
22-24	finish final report

Project administration

The supervisor Daniele De Sensi and the student Tobias Rahn agreed to meet weekly on Thursday at 5PM CET for 30 minutes to discuss the progress and potential questions.
The student is advised to write a weekly report at the end of each week and to send it to his advisors. The idea of the weekly report is to briefly summarize the work, progress and any findings made during the week, to plan the actions for the next week, and to bring up open questions and points.
Final Report The final report has to be presented at the end of the project and a digital copy needs to be handed in and remains property of the SPCL. This task description has to be attached to the final report.

Deliverables

- Scripts and programs used to run the benchmarks and Applications on the clusters
- collected data of the microbenchmark and application runs
- Scripts and programs used to analyze the data
- written report (documentation of the process, discussion of results and a conclusion)

Success criteria

The goal of this project is to:

- Provide a review of the three HPC cloud providers
- Define a set of microbenchmarks to analyze the performance of HPC cloud providers
- Analyze how those HPC Clouds are affected by network congestion / noise by using the above microbenchmarks and how they differ from traditional HPC systems
- Investigate the impact of OS noise on network performance in HPC clouds

We do this in order to better understand the impact of network congestion on the performance of HPC Clouds.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Noise Estimation in HPC Cloud Networks

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Rahn	Tobias

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Bonstetten, 8.9.2021	T. Rahn

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.