Politecnico di Torino

Microsoft Azure InfiniBand Configuration

Gabriel Maiolini Capez

Advisor: Masoud Hemmatpour

October 30, 2018

Abstract

A small cluster of InfiniBand Remote Direct Memory Access (RMDA) virtual machines was configured in Azure and its capabilities assessed to determine the feasibility of migrating InfiniBand applications being run in private clusters there. This report describes the activities realized in this context, presenting the work done and the limitations found.

Contents

1 Introduction

Large-scale testing of RDMA algorithms with dozens or hundreds of nodes over InfiniBand is desired to evaluate their characteristics before being deployed. As most private clusters are too small, they would have to be greatly expanded, requiring high capital expenditures, to do so. Instead, executing the tests using available cloud computing services that offer large InfiniBand RDMA networks can be done quickly and at a much lower cost. This report is concerned with the latter approach. For non-enterprise customers, there were three potential vendors of Infrastructure As A Service (IaaS) to investigate:

1. Amazon AWS EC2
2. Google Cloud Platform
3. Microsoft Azure

Before delving into the architectural differences and billing policy, a careful study of their offered services revealed only Microsoft Azure offers InfiniBand RDMA to customers for specific types of Virtual Machines (VM):

> For MPI [Message Parsing Interface] applications, dedicated RDMA backend network is enabled by FDR InfiniBand network, which delivers ultra-low-latency and high bandwidth. [1]

Therefore, Microsoft Azure was chosen as the cloud IaaS provider.
However, most RDMA applications are not MPI based and this type of access - directly interfacing with the InfiniBand Network Card (NIC) - to their RDMA network had to be tested.

1.1 Remote Direct Memory Access

Remote Direct Memory Access is a way to remotely and directly access an application's memory, without intervention by the processor and bypassing the kernel. Data can be transferred quickly between machines and applications without incurring costly context switches or kernel overhead. Therefore high speed, low latency, network communication can be achieved. For some applications, extremely low latency interprocess communication (IPC) is fundamental and can only be achieved through RDMA.

1.2 InfiniBand

InfiniBand (IB) is a very high speed switched-fabric interconnect that provides extremely low latencies and very high throughput for network communication. When using RDMA, applications can exploit IB networks to minimize IPC latency. As the most RDMA applications are network-bound and require extremely low latencies, IB is the only solution able to satisfy their requirements.

1.3 Microsoft Azure

1.3.1 Infrastructure as a Service (IaaS)

IaaS was chosen as the clouding computing service because it provides instant, on-demand infrastructural resources - such as computing servers, storage nodes, network interfaces, VLANs, firewalls -, allowing a greater degree of control over the deployed resources than a Software As a Service model (SaaS), in which applications are run with no control over the resources

allocated to do so. Another consideration is that Azure's IaaS gives full control over the Operating Systems, Applications and Network Architecture to the consumer, even if the underlying infrastructure cannot be directly managed. Lastly, the RDMA algorithms require access to Azure's InfiniBand RDMA network, something that is only offered in the IaaS Service Model.

1.3.2 Microsoft Azure

Microsoft Azure is a cloud computing service that allows customers to quickly provision and access on-demand computing resources to deploy and run their applications as needed without having to worry about the underlying infrastructure and its management. From the customer's point of view, its resources are practically inexhaustible and it is possible to quickly scale applications. Moreover, the service provider guarantees of Quality of Service parameters such as availability through the the Service Level Agreement (SLA), expert technical support provided on demand and the cloud's distributed nature, with its inherent flexibility and reliability, make such a service very alluring to a customer.
Azure's billing policy can be either on a pay as you go basis, in which one is billed by the minute a resource is allocated and can quickly scale computational capacity on demand, or through reserved virtual machine instances, where one pays for the infrastructure allocation over a long period of time (usually a year) and trades off flexibility to reduce operating costs due to the greater predictability of resource demands. As the activities developed in this report were short term and targeted a low cost, the pay as you go model was adopted.

2 Microsoft Azure Cluster Configuration

Cluster configuration is segmented into two parts for clarity: Hardware configuration and Software configuration. The former is concerned with the infrastructure, cluster architecture, computing resource allocation and configuration, deployment model, availability and other properties, whereas the latter refers to the software requisite configuration to access such resources and run applications on them.

2.1 Hardware

The period during which the activities of this report were elaborated, Microsoft Azure was transitioning from their Classic deployment model to the Resource Manager model [2]. As the latter is more modern and recommended for new deployments, it was adopted as the deployment model.

2.1.1 Resource Manager Model

The Resource Manager Model provides an application called Resource Manager through which the user can create a Resource Group where all the resources demanded by the cluster and the application are deployed.Thus, management is simplified by allowing the user to control and monitor all of his resources using a single control interface. Most importantly, it allows the user to create templates in JSON [3] - (Section 5) - to automate the deployment of his cluster by parameterizing a Resource Group as a function of its resources. Simply put, it permits the user to create a cluster architecture with varying amount of nodes, different node properties, initialization scripts, VM images, network architecture, among many other configurations.
This very concise and powerful model provides an additional graphical interface to create, tune or reconfigure existing cluster. Moreover, it allows the user to define access controls, view diagnostic logs, keep track of expenses and resource allocations, control individual resources,

etc. For simplicity, the graphical interface was used to manually deploy each resource of the cluster, validate them and ultimately, to derive a template from the final architecture.
The specifics of the architecture is discussed in Section 2.1.2.

2.1.2 Cluster Architecture

The Cluster Architecture is composed of several elements:

1. Virtual Machines (Kanzi-01, Kanzi-02 and Kanzi-03):
   Where the operating system and applications are run. These are the virtualized servers that form the core computing resource of the cluster architecture.
   Three machines were arbitrarily chosen because the application requires at least two (one server, at least one client) and their configuration is described in (Section 2.2.2)

2. An Availability Set (RDMASet):
   Contains all the Virtual Machines that must communicate with each other over Azure's IB RDMA Network.
   An Availability Set describes resources that must be run in different physical nodes for greater availability and reliability. Originally designed for redundancy of services that rely on multiple Virtual Machines (VM), for our purposes it means the VMs that support RDMA of a single Resource Group will be run in unique servers, physically connected by FDR Infiniband. [4].

3. A Virtual Network (kanzi-IB-vnet):
   In addition to IB, we want to create a local network in the cluster for the VMs to communicate with each other over IP, isolated from other clusters.

4. A Network Interface Card (kanzi-01846, kanzi-02531, kanzi-03299):
   Each VM requires a NIC to connect it to the local area network and to the Azure RDMA Network. A firewall and a Network Security Group define the network access control and security policy, being also present. Furthermore, a public IP address was attributed to each machine to make them remotely accessible over the Internet.

5. A Storage Account (kanzidiag):
   A location to store all of the resource group's information, generated by the management software. Physically, the Storage Account provides a network disk (over SMB/Samba) with which any resource can communicate. Additionally, each VM can use it to write their logs to and share data such as the Intel Parallel Studio XE Cluster Edition installation license keys needed by Intel MPI (Section 3.3).

2.1.3 InfiniBand Virtual Machines Size

After choosing the service model (Section 1.3.1), deployment model (Section 2.1.1) and elaborating the cluster architecture and the set of resources to be used (Section 2.1.2), their computational requirements had to be defined. For the Virtual Machines, it meant choosing an instance type or, in other words, the hardware configuration of the machine on which the Operating System (OS) will be run. As most distributed applications are I/O-bound (IPC over the network), an IB-based RDMA was essential.
Among the Azure instance types, IB is provided only by the High Performance Compute (HPC) VMs, detailed in Table 1.
These are focused on compute-bound applications and have a high number of CPU cores, copious amounts of RAM, use Intel Xeon CPU series better fit for HPC but above all, they have geographical proximity between servers that are physically connected by a high-throughput IB

RDMA network.[1]. This is the coveted property of nodes for the applications that we desire to evaluate as it translates into very low latencies.

Instance Type	A8/A9	H16/H16_mr	NC24_r v2/ND24_r v2
CPU Type	Intel Xeon E5-2670	Intel Xeon E5-2667 v3	Intel Xeon E5-2667 v3
#CPUs	8/16	16/16	24/24
Memory (GB)	56/112 (DDR3)	112/224 (DDR4)	448/448 (DDR4)
Storage (GB)	382/382	2000/2000	1344/1344
GPU	N/A	N/A	4x Tesla P100/4x Tesla P40
Cost (US$/hour)[1]	0.975/1.95	1.989/2.665	9.108/9.108

Table 1: Azure VMs with InfiniBand RDMA

Considering all of the instances of Table 1 are connected to the Azure RDMA Network and have sufficiently powerful hardware that can run IB applications, the main criterion for selecting an instance type was cost minimization and because the A8 type has the lowest hourly cost (below one dollar), it was chosen for two of the three nodes. Instead, the A9 type was chosen for the third machine to test cross-compatibility among them. Lastly, it was necessary to choose a geographical location to which the resources would be deployed and due to the Azure test account's limitations, the location chosen was Eastern-US.

2.1.4 InfiniBand Network Interface Card Vendor

Discovering who is the NICs vendor is useful to better understand the capabilities of the Azure RDMA network. After some research, the maker of Azure's InfiniBand Network Interface Cards was identified as Mellanox Technologies. [5] [6] [7]. Later, this information was validated by the presence of a Mellanox Connect-X 4.0 NIC in the configured VMs (Section 3).

2.2 Software

After all hardware requirements are defined and resources are allocated and configured inside the cluster, the VMs have to be set-up by choosing an operating system and configuring it to run the applications.

2.2.1 Operating System

Linux was chosen as the operating system due to its stability, flexibility, simplicity of configuration, low cost and my familiarity with it. However, there are many Linux distributions that can be used as a hypervisor guest with similar capabilities, but one had to be chosen.
Microsoft Azure offers a list of "endorsed distributions" - distributions that are better supported by Azure's hypervisor (Hyper-V), embedded with Azure-specific kernel modules -. and their versions. [8]. Ubuntu and CentOS are the most widely deployed endorsed distributions and have free licenses. The distinguishing feature of CentOS was its availability in the Azure Marketplace as a CentOS-based image dedicated to HPC and configured with RDMA drivers and a kernel to which automatic updates are applied for free by Azure to maintain compatibility with their network. Also, there are especially recommended for workloads on Azure's A8 and A9 instances, which perfectly fit the Cluster's Architecture of Section 2.1.2. Technical support for these images is provided by RogueWave for Azure [9].

2.2.2 Virtual Machine Configuration

2.2.2.1 Deploying the Operating System

Azure offers two ways to configure the OS: either a prepared Virtual Machine image, configured off-cloud, is supplied through a template [3] or one must be picked from the Azure Marketplace. Given our choice of OS in Section 2.2.1 and the simplicity of deploying an image from the Azure Marketplace directly to the node, this route was selected.
It would have been just as simple to have configured the OS and created an image of it offline. However, using the already available image is less time consuming, requires less network resources and is much more portable, facilitating adoption and configuration.

2.2.2.2 Operating System and Application Configuration

Despite selecting a good base image, it is minimalist and some extra configuration and dependency installation is required. A bash script was written for this task (install_dependencies.sh) and is present in the Files folder of this report. Its steps are below:

1. InfiniBand Support is only accounted for in the kernel, but there are none of the required libraries - in example, the InfiniBand Verbs API - for interfacing with IB. Moreover, there are no development tools present, as the image comes only with the package manager and no compiler. These must all be installed.

2. Interfacing with the Mellanox NIC requires the Mellanox OpenFabrics Entreprise Distribution (OFED) proprietary InfiniBand stack be installed following three steps:

   (a) Downloading the proprietary OFED drivers from Mellanox OFED.

   (b) Untar the file.

   (c) Execute the installer, following its instructions.

   More detailed instructions are present in [10] [11] [12] .

3. systemd must enable the IB services and IB drivers to be loaded on boot

4. For MPI, the included Intel MPI Runtime provides the facilities to run compiled code. However, to be able to compile Intel MPI programs inside the nodes, it has to be replaced with Parallel Studio XE Cluster Edition in all nodes to prevent mismatching library versions.

```
sudo yum install epel-release sshpass nmap \
sudo yum --setopt=group_package_types=optional \
groupinstall "Infiniband Support"
sudo yum install infiniband-diags perftest qperf \
numactl-devel numactl rdma-core-devel papi papi-devel \
libmemcached-devel libnuma cmake3 libibverbs-utils \
opensm libmemcached infiniband-diags
sudo yum group install "Development Tools"
# Add services to systemd init
sudo service rdma enable
sudo service opensm enable
# Add modules to boot
dracut --add-drivers "mlx4_en mlx4_ib mlx5_ib" -f
# Reboot and chckconfig
sudo reboot
chkconfig rdma on
chkconfig opensm on
# Remove default Intel Runtime to install Parallel Studio
# Otherwise there could be incompatibilities between versions
sudo yum remove intel-mpi-doc intel-mpi-rt-core-174.x86_64 \
intel-mpi-rt-psxe-050.x86_64
```

Listing 1: Bash code for steps 1-4.

5. To allow nodes to communicate freely among themselves and use MPI, they were configured with the same login information instead of different SSH keys as recommended by [4]. Each machine then generates its own SSH keys and shares it with others in the VLAN by running the user_authentication.sh script, which also offers automatic node discovery through Nmap, in a single node.

```
sudo ./user_authentication.sh $(user) $(password) $(internal VLAN prefix)
```

6. As will be discussed in (Section 3.2), Intel MPI must be utilized to access the RDMA network and to ascertain its capabilities, a few example programs were written. However, to compile and run them one must acquire a license of the Intel MPI Compiler and API and install them:
   (a) Acquire a Intel Parallel Studio XE Cluster Edition license to be able to compile the Intel MPI software.
   (b) Download package from Parallel Studio XE Cluster Edition's website and uncompress it with tar -xzvf.
   (c) Choose one node to run the Intel Installer from and follow the instructions, selecting "Cluster Install" and provide a list of hosts through a text file to which the Parallel Studio will be installed (or follow alternative steps). Note that Step 5 must have been done in order to allow remote SSH installation of applications among nodes.
   (d) Complete the installation, ignoring optional dependencies that will not be fulfilled by a VM such as X11.

(e) Adjust and set the appropriate environment variables inside the .bashrc configuration file (Section 5) to reflect the new installation. Lines 1-4 are loading the appropriate environment variables for using Intel MPI. Lines 5-7 define the MPI fabric (shared memory DAPL), its provider (OpenFabrics V2 InfiniBand) and disabling on-demand (dynamic) MPI connections.

```
export PATH=(INSTALL_DIR)/intel/compilers_and_libraries_2018/ \
linux/mpi/intel64/bin:$PATH
source $(INSTALL_DIR)/intel/bin/compilervars.sh intel64
source  $(INSTALL_DIR)/intel/bin/ifortvars.sh intel64
source  $(INSTALL_DIR)/intel/impi/2018.1.163/intel64/bin/mpivars.sh \
intel64
export I_MPI_FABRICS=shm:dapl
export I_MPI_DAPL_PROVIDER=ofa-v2-ib0
export I_MPI_DYNAMIC_CONNECTION=0
```

(f) Compile the code using mpiicc.

(g) Run the programs using mpirun. Observe that a copy of the program to be run must be available inside each node.

(h) (Alternative) As all applications to be run must be shared across all interacting nods in the cluster, one can use the Storage Account (Section 2.1.2) through SMB to set up a permanent, shared cloud storage between VMs in /etc/fstab. Hence, Parallel Studio XE can be installed (Step 6c) in a network shared folder that is accessed by all nodes instead of installing a copy of it in each node. The advantage is less storage usage at a small initial overhead cost of retrieving Parallel Studio applications from this shared folder to load in memory. Because the nodes are connected by Gigabit Ethernet in addition to IB, this cost is negligible.
Instead, one could also do the install locally in a head node and create a NFS share on it to share the Parallel Studio and application files. This was the approach I used for briefness.

```
sudo yum install nfs-utils
#If you want to use NFS, create the dirs
# On server
mkdir ~/cloud
sudo "~/cloud *(rw, sync, no_root_squash, no_subtree_check)" \
> /etc/exports
sudo exportfs -a
sudo service nfs-server enable
sudo service nfs-server start
# On client, edit fstab to reflect file
#master:$(PATH_TO_SHARED_DIR)/cloud $(PATH_TO_LOCAL_DIR)/cloud nfs
```

Ultimately, with the Intel MPI Compiler configured on the cluster created above, configuration is completed.

2.2.3 Imaging

After one node has been configured, it can be replicated using the Resource Manager; when adding a new machine, it is possible to create it from an image of an already existing VM. Hence, no specific image capturing procedure is needed for creation as the Resource Manager abstracts

it. However, a VM can be imaged at any time by selecting its name from the Resource Manager and ordering an image capture. This capture will be stored under the Storage Account's Network drive.

2.3 Reproducibility

Using the Resource Manager graphical interface to create the Resource Group and Cluster Architecture is very intuitive and requires few steps:

1. Select "Resource Group" from the left Menu of the Azure Portal.
2. Press Add.
3. Configure the Resource Group's name and location (in this case, Eastern US).
4. After the Resource Group has been created, click in its "Overview" tab and press Add.
5. Search for CentOS-based 7.3 HPC in the market and select i by pressing "Deploy".
6. Choose "Resource Manager" as a deployment model and click on "Create".
7. Insert the VM's name, login information, select the previously created resource group and location.
8. Choose the adequate instance type from the "Instance Type" Menu.
9. Create (if it is the first VM in the cluster) or select an exiting Virtual Network, Network Security Group, Public IP, whether Accelerated Networking is desired, etc.
10. Click on deploy and wait for the resources to be allocated. Repeat steps 4 through 9 until the cluster deployment has been finished.

However, to automate resource allocation and ease the reproduction of this report's cluster architecture, a template (available in the Files folder) was devised. To deploy it:

1. Click "New" on the left Menu of the Azure Portal.
2. Type "Template Deployment".
3. Press "Edit Template".
4. Press "Load File" and select the template provided.
5. Type in the required parameters.
6. Deploy.

After deploying the architecture, copying and running the provided files (Section 5) will result in a cluster identical to the one described here.

3 Application Testing

3.1 Configuration Validation

Before attempting to compile and execute an RDMA benchmark, rdma_bench [13], the IB interface status was checked with some default InfiniBand utilities, lsmod to check loaded modules and by running a ping-pong MPI application in order to check for connectivity.

With all services active and the devices ready, an Intel MPI ping-pong was run over InfiniBand:

Below is the partial output of lsmod indicating IB drivers have been loaded. The full list of loaded modules is available in a provided file called listed_modules.

```
*SNIP*
ib_isert                55484  0
iscsi_target_mod       298112  1 ib_isert
ib_iser                 47796  0
libiscsi                57233  1 ib_iser
scsi_transport_iscsi    99909  2 ib_iser,libiscsi
ib_srpt                 47966  0
target_core_mod        371914  3 iscsi_target_mod,ib_srpt,ib_isert
ib_srp                  48464  0
scsi_transport_srp      21089  1 ib_srp
ib_ipoib                91950  0
rdma_ucm                26837  0
ib_ucm                  22585  0
ib_uverbs               51854  2 ib_ucm,rdma_ucm
ib_umad                 22129  0
rdma_cm                 53755  4 rpcrdma,ib_iser,rdma_ucm,ib_isert
ib_cm                   47149  5 rdma_cm,ib_srp,ib_ucm,ib_srpt,ib_ipoib
iw_cm                   46022  1 rdma_cm
dm_mirror               22135  0
dm_region_hash          20862  1 dm_mirror
dm_log                  18411  2 dm_region_hash,dm_mirror
dm_mod                 114430  2 dm_log,dm_mirror
intel_powerclamp        14419  0
iosf_mbi                13523  0
ib_core                210381  14 rdma_cm,ib_cm,iw_cm,rpcrdma,ib_srp,ib_ucm, \
ib_iser,ib_srpt,ib_umad,hv_network_direct,ib_uverbs, \
rdma_ucm,ib_ipoib,ib_isert
*SNIP*
```

Listing 2: Loaded kernel modules

3.2 Program Execution

3.2.1 Results

With the system successfully configured and tested, applications were compiled with mpiicc and run. Unfortunately, they hung and the system had to be reboot from the management interface. These were programs that even after being adapted were perfectly working in a small private InfiniBand cluster, indicating a problem with the configuration.

```
ls -l /dev/infiniband/uverbs0
crw-rw-rw-. 1 root root 231, 192 Jan 22 14:45 \
/dev/infiniband/uverbs0
```

The kernel device was loaded, all kernel modules were up and according to ibv_devices, the device was active. A strace indicated the problem was ibv_open_device: the program hung whenever trying to listen on the device detected through ibv_get_device_list. Running the default IB ping-pong[2] also hung the machine. Other default utilities (such as ibhosts or ibping -S) immediately failed on the mad_rpc_open_port syscall. This indicated a device problem. Checking the data with ibv_devinfo -d mlx4_0 shows:

```
hca_id: mlx4_0
        transport:                      iWARP (1)
        fw_ver:                         0.0.000
        node_guid:                      0015:5d33:ff4a:0000
        sys_image_guid:                 0000:0000:0000:0000
        vendor_id:                      0x15b3
        vendor_part_id:                 4099
        hw_ver:                         0x0
        phys_port_cnt:                  1
                port:   1
                        state:                  PORT_ACTIVE (4)
                        max_mtu:                4096 (5)
                        active_mtu:             4096 (5)
                        sm_lid:                 0
                        port_lid:               0
                        port_lmc:               0x00
                link_layer:             Ethernet
```

Listing 3: ibv_devinfo -d mlx4_0 output

Which is a strange, incompatible, result. Microsoft Azure does not offer iWARP nor IP over InfiniBand [1] and the parameters above are inconsistent with their specification, in addition to null ports and firmware versions, suggesting a misconfigured device. Running lspci only showed the virtualized devices, confirming the InfiniBand NIC is not exposed to the host.
Reconfiguring and recompiling the kernel and all its modules (Section 5) proved fruitless, as did manually reinstalling all libraries and dependencies in the Kanzi-03 machine. Curiously, MPI applications were still being properly run over InfiniBand and therefore a more ulterior motive was suspected: Azure restricts RDMA InfiniBand access to MPI traffic through their hypervisor.

[2] ibv_rc_pingpong -g 0 -d mlx4_0 -i 1

3.2.2 Discussion with Support

Azure support was contacted using their paid ticket system to obtain a definite answer to the suspicions raised above and their documentation rechecked. Although defined as "For MPI applications", their description was sufficiently broad to understand that Azure allows non-MPI loads. Their answer was:

> Generally, RDMA capabilities can boost the performance of Message passing Interface (MPI). Hence, you are recommended to use MPI for applications. You can also use Non-MPI. However, am sharing you the article. https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-hpc#rdma-capable-instances

Which would be a satisfactory answer, except that link explicitly mentions Intel MPI as a requirement for access to the Azure RDMA Network. Interrogating for further clarification and providing the sequence of commands that produced the error, they asked for time to engage the Linux team on our problem.

```
ibv_device *dev;
dev = ibv_get_device_list();
ibv_open_device(dev) > Error.
```

Listing 4: InfiniBand Verbs API Error on Cluster

For completeness, a Windows VM was set-up but there was still no access to the RDMA network for non-MPI applications as the Mellanox NIC remained hidden from the hypervisor's guest. Searching for cached copies of their old documentation, a statement confirming the RDMA traffic was reserved for MPI traffic was found, showing that it is not possible to interface directly with the Mellanox InfiniBand Network Interface Card as required by the existing applications. [14]

3.3 Intel MPI

Message Parsing Interface (MPI) is a parallel computing standard that defines an Application Interface (API) for interprocess communication using a message passing model, in which messages containing data are sent between processes. Intel MPI is one of such libraries. As only Intel MPI can access the Azure IB-based RDMA Network, the applications that depend on InfiniBand Verbs API would have to be ported to Intel MPI. However, this is only possible if they are not antagonistic to the message passing model of MPI. Annexed in this report (Section 5) are simple, commented C programs that demonstrate one-sided (memory access without explicit cooperation/participation of the target process) and two-sided communication (both sender and receiver processes must cooperate/match operations) with the following operations:

1. Read:
   One-sided memory read (read from a remote memory region into local memory) is provided by the MPI_Get function, whose prototype is below.

   ```
   int MPI_Get(void *origin_addr, int origin_count, MPI_Datatype origin_data
   int target_count, MPI_Datatype target_datatype, MPI_Win win)
   ```

   It receives the address of the buffer to where the received data will be written, the number of entries to read to the buffer, the type of each entry (i.e. MPI_Int), the process ID (rank), an offset inside the buffer to be read (target_disp), the number of bytes to read from the target buffer and an MPI window.
   An MPI Window is a memory region of the process that is declared as remotely accessible. It must be created, allocated or freed using:

```
int MPI_Win_create(void *base, MPI_Aint size, int disp_unit, \
MPI_Info info, MPI_Comm comm, MPI_Win *win)
int MPI_Win_allocate(MPI_Aint size, int disp_unit, \
MPI_Info info, MPI_Comm comm, void *baseptr, MPI_Win *win)
int MPI_Win_free(MPI_Win *win)
```

Observe that the access to the window can be of two types:

(a) Active Synchronization: Using the MPI_Fence concept, which synchronizes all processes to a time window (epoch) during which any process may access another's memory. There is a more restricted version called MPI_win_start MPI_win_stop where it is possible to specify the processes that communicate.

```
int MPI_Win_fence(int assert, MPI_Win win)
```

(b) Passive Synchronization: Instead of the fencing synchronous communication concept, we have asynchronous communication controlled by the callee of MPI_win_lock and MPI_win_unlock. These operations "lock" and "unlock" the remote memory region of the process defined by the window so only a subset of processes is allowed to modify it during an epoch.

```
int MPI_Win_lock(int lock_type, int rank, int assert, MPI_Win win) \
int MPI_Win_unlock(int rank, MPI_Win win)
```

2. Write:
   One-sided memory write (write from local memory to a remote memory region) is provided by the MPI_Put function that has the same parameters as the MPI_Read function.

```
int MPI_Put(const void *origin_addr, int origin_count, MPI_Datatype
origin_datatype, int target_rank, MPI_Aint target_disp,
int target_count, MPI_Datatype target_datatype, MPI_Win
win)
```

3. Receive:
   Two-way communication receive through MPI_Recv It is a blocking call on a message buffer; until a message has been sent and received, the process is blocked.

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int
MPI_Comm comm, MPI_Status *status)
```

4. Send:
   Two-way communication send through MPI_Send. It is a blocking call until the process has sent the message. It has a non-blocking version and a synchronous version that only unblocks after the receiver has acknowledged the data.

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,
MPI_Comm comm)
```

Observe that MPI must be initialized and finalized through the usage of the MPI_Init and MPI_Finalize operations.

```
int MPI_Init( int *argc, char ***argv )
int MPI_Finalize( void )
```

All these concepts are illustrated in the annexed code.
Intel had an example [15] of one-sided communication with Active RMA Synchronization that I fully commented, indicating how to use Passive RMA Synchronization instead if one wants to avoid fencing. For the two-sided communication, two simple scripts were written based on prior experience from [16] [17] [18]. At last, some excellent MPI resources [19] [20] [21] [22] [23] are provided as reference.

3.4 Accelerated Networking

While this report was being written a new configuration option, "Accelerated Networking", became available for the creation of new Virtual Machines. By eliminating the hypervisor's virtual switch and offering direct access to the Network Interface Card, it is possible to improve application performance [24]. For RDMA applications, granting access to the NIC would eliminate the InfiniBand Verbs API issues originating from Azure's decision to restrict RDMA exclusively to MPI applications (Section 3.2) and virtualize the NIC. However, InfiniBand-connected instance types have yet to receive this capability (Single Root Input/Output Virtualization). Whenever it becomes available to IB-connected VMs, it will be necessary to **recreate** the entire cluster as Accelerated Networking is only supported by new Resource Groups and VMs [24]. Nevertheless, it will be sufficient to simply run this procedure on the first node in the new cluster, created according to (Section 2.2.2), and replicate it to create all remaining nodes using the Resource Manager's capability of deploying new nodes from another one's image.

4 Conclusion

Cloud-computing services were investigated for InfiniBand-based RDMA capabilities so RDMA-based applications could be tested at large scale. It was established that only Microsoft Azure offered such a capability and therefore an Azure InfiniBand RDMA cluster was configured to determine if the applications could be run there. Unfortunately, they could not. Tests confirmed that the Azure RDMA Network restricts access exclusively to Intel MPI traffic and these applications, based on the InfiniBand Verbs API, are prevented from accessing the network. Thus, they would have to be ported to Intel MPI before being migrated. However, a new technology (Single Root Input/Output Virtualization) is being deployed across the Azure Network and albeit not yet available for InfiniBand, it would allow those applications to run without modifications in the Azure RDMA Network.

5 Appendix

All code that was written/commented, configuration files that were changed, deployment template, etc can be found under the Files folder of the folder containing this report. They are:

1. Deployment Template (ExportedTemplate-Kanzi-IB.zip)
2. .bashrc configuration file
3. install_dependencies.sh script
4. Kernel Configuration
5. Loaded Kernel Modules
6. Intel MPI Scripts (MPI_one_sided.c, split_work.c, mpi_hello_world.c)

Bibliography

[1] (Aug. 11, 2017). High performance compute virtual machine sizes, [Online]. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/sizes-hpc?toc=%2fazure%2fvirtual-machines%2fwindows%2ftoc.json.

[2] T. Fitzmaken, R. Squillace, and K. Crider. (Nov. 15, 2017). Azure resource manager vs. classic deployment: Understand deployment models and the state of your resources, [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-deployment-model.

[3] T. FitzMacken and U. Strid. (). Create and deploy your first azure resource manager template, [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-create-first-template.

[4] D. Lepow, C. Wilson, Teppeilshi, and R. Squillace. (Mar. 14, 2017). Set up a linux rdma cluster to run mpi applications, [Online]. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/classic/rdma-cluster.

[5] G. Shainer. (May 18, 2017). Deep learning in the cloud accelerated by mellanox, [Online]. Available: http://www.mellanox.com/blog/2017/05/deep-learning-in-the-cloud-microsoft-azure/.

[6] J. Sirosh. (Nov. 29, 2016). From "A PC on every desktop" to "Deep Learning in every software", [Online]. Available: https://azure.microsoft.com/en-us/blog/from-a-pc-on-every-desktop-to-deep-learning-in-every-software/.

[7] B. Cotton. (Feb. 8, 2017). Lammps scaling on azure infiniband, [Online]. Available: https://cyclecomputing.com/lammps-scaling-azure-infiniband/.

[8] S. Zarkos, K. Panday, I. Foulds, and R. Squillace. (). Linux on distributions endorsed by azure, [Online]. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/endorsed-distros.

[9] (). Support for centos on azure, [Online]. Available: https://www.roguewave.com/products-services/open-source-support/centos-on-azure.

[10] M. Technologies. (). Mellanox in, [Online]. Available: MellanoxOFEDforLinuxInstallationGuide.

[11] ophirmaor. (Nov. 29, 2016). Howto install mlnx ofed driver, [Online]. Available: https://community.mellanox.com/docs/DOC-2688.

[12] D. Barak. (Feb. 27, 2015). Working with rdma using mellanox ofed, [Online]. Available: http://www.rdmamojo.com/2014/11/22/working-rdma-using-mellanox-ofed/.

[13] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance rdma systems", in *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '16, Denver, CO, USA: USENIX Association, 2016, pp. 437–450, ISBN: 978-1-931971-30-0. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026959.3027000.

[14] (Feb. 28, 2016). About the a8, a9, a10, and a11 compute-intensive instances, [Online]. Available: https://github.com/daltskin/azure-content/blob/master/articles/virtual-machines/virtual-machines-a8-a9-a10-a11-specs.md.

[15] L. Q. Nguyen. (). Mpi one-sided communication, [Online]. Available: https://software.intel.com/en-us/blogs/2014/08/06/one-sided-communication.

[16] (). Workshop on parallel programming and optimization for intel architecture, [Online]. Available: https://github.com/intel-unesp-mcp/Hands-on-Workshop.

[17] (). Infieri lab workshop ii, [Online]. Available: https://github.com/intel-unesp-mcp/infieri-2017-advanced.

[18] (). Infieri lab workshop i, [Online]. Available: https://github.com/intel-unesp-mcp/infieri-2017-basic.

[19] J. Jeffers, J. Reinders, and A. Sodani, *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition 2Nd Edition*, 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016, ISBN: 0128091940, 9780128091944.

[20] B. Barney and L. L. N. Laboratory. (). Message passing interface (mpi), [Online]. Available: https://computing.llnl.gov/tutorials/mpi/.

[21] V. Eijkhout. (). Parallel programming in mpi and openmp, [Online]. Available: http://pages.tacc.utexas.edu/~eijkhout/pcse/html/.

[22] (). A comprehensive mpi tutorial resource, [Online]. Available: http://mpitutorial.com/.

[23] T. Hoefler, J. Dinan, R. Thakur, B. Barrett, P. Balaji, W. Gropp, and K. Underwood, "Remote memory access programming in mpi-3", *ACM Trans. Parallel Comput.*, vol. 2, no. 2, 9:1–9:26, Jun. 2015, ISSN: 2329-4949. DOI: 10.1145/2780584. [Online]. Available: http://doi.acm.org/10.1145/2780584.

[24] (Jan. 2, 2018). Create a linux virtual machine with accelerated networking, [Online]. Available: https://docs.microsoft.com/en-us/azure/virtual-network/create-vm-accelerated-networking-cli.