Nomad: Migrating OS-bypass Networks in Virtual Machines

WEI HUANG, JIUXING LIU, MATTHEW KOOP, BULENT ABALI, AND DHABALESWAR K. PANDA

Technical Report
OSU-CISRC-10/06-TR74

Nomad: Migrating OS-bypass Networks in Virtual Machines

W. Huang† J. Liu‡ M. Koop† B. Abali‡ D. K. Panda†

† *Computer Science and Engineering*
The Ohio State University
Columbus, OH 43210
{*huanwei, koop, panda*}*@cse.ohio-state.edu*

‡ *IBM T. J. Watson Research Center*
19 Skyline Drive
Hawthorne, NY 10532
{*jl, abali*}*@us.ibm.com*

Abstract

Virtual machine (VM) technology is experiencing a resurgence due to various benefits including ease of management, security and resource consolidation. Live migration of virtual machines allows transparent movement of OS instances and hosted applications across physical host machines. It is one of the most useful features of VM technology because it provides a powerful tool for effective administration of modern cluster environments.

Migrating network resources is one of the key problems that needs to be addressed in the VM migration process. Existing studies of VM migration have focused on traditional I/O interfaces such as Ethernet, however, modern high-speed interconnects with intelligent NICs pose significantly more challenges as they have additional features including hardware level reliable services and direct I/O accesses.

In this paper we present Nomad, a design for migrating modern interconnects with the aforementioned features, focusing on cluster environments running VMs. We introduce a thin namespace virtualization layer to efficiently address location dependent resource handles and a handshake protocol which transparently maintains reliable service semantics during migration. We demonstrate our design by implementing a prototype based on the Xen virtual machine monitor and InfiniBand. Our performance analysis shows that Nomad can achieve efficient migration of network resources, even in environments with stringent communication performance requirements.

1 Introduction

Virtual machine (VM) technologies are experiencing a resurgence in recent years in both industry and academia [26]. A key component in a VM environment is the virtual machine monitor (VMM), which is also referred to as the hypervisor. The VMM provides virtualized hardware interfaces to hosted guest virtual machines (VMs). It allows many different guest VMs to run simultaneously in a single physical box, and provides a wide range of benefits including resource consolidation, performance isolation, and user-transparent migration and checkpointing/restart. Among them, user-transparent live migration is one of the most interesting features. It helps separate the hardware and software management and consolidate clustered hardware into a single coherent management domain [5].

Recently, network interconnects providing low latency (less than 5μs) and very high bandwidth (multiple Gbps) are emerging, such as InfiniBand [10], Myrinet [16], Quadrics [25], etc. Such interconnects also support features including OS-bypass I/O and Remote Direct Memory Access (RDMA). With OS-bypass, applications can directly initiate communication operations without the involvement of the operating system. RDMA allows processes on remote nodes to access certain memory buffers of a local process. Modern interconnects with excellent performance and flexibility provided by these features are being widely adopted in cluster environments, which typically host data center or high performance computing (HPC) applications.

Virtual machine technology, due to its wide range of benefits, is one of the possible remedies to solve performance, scalability, and system management problems caused by today's ultra-scale clusters [22, 12]. By utilizing the OS-bypass feature of the high speed interconnects, direct I/O access without involvement of the VMM can be realized for high performance I/O in virtual machine environment, as we have done in previously in VMM-bypass I/O [13]. As a result, virtual machine based cluster environments are a promising solution to achieve both high performance and high manageability.

However, compared with traditional network devices such as Ethernet, modern network interconnects with OS-bypass pose additional unresolved challenges with respect to VM migration:

- First, intelligent NICs of OS-bypass networks manage large amounts of resources, such as a memory protection table in InfiniBand NICs to allow safe user level access. This information can only be accessed by opaque handles, and the value of these handles are usually location-dependent and cannot be migrated easily with the virtual OS.

- Second, with user-level communication various operations can bypass the virtual OS and VMM, thus the VMM has less control of network activity from the processes hosted in the migrating OS. In contrast, with TCP all traffic will go through the kernel and it is relatively easy to manage the communication of the processes during migration.
- Further, modern interconnects may store connection state information on NICs, which cannot be directly managed by software. This information is used for features like reliable service implemented in hardware. Compared with traditional TCP where connection states are stored in memory, which will be automatically migrated with the virtual OS, it is much harder to manage the NIC-stored connection states with modern interconnects.

Current VM technologies have proposed several solutions for migrating virtual OSes with traditional networks. However, problems related with migration of modern interconnects, which are more widely adopted in cluster environments, are left unstudied. In this paper, we attempt to overcome these challenges by proposing *Nomad*, a framework to address the migration issues of modern OS-bypass interconnects. We target cluster computing environments, which are very tightly coupled systems with stringent communication performance requirements.

Our approach introduces a thin namespace virtualization layer to efficiently address location dependent resource handles. We also devise a coordination protocol to avoid packet drops or out-of-order communication during migration. To demonstrate our ideas, we implemented a prototype of Nomad in the Xen virtual machine environment for InfiniBand. The prototype is based on our earlier work of VMM-bypass I/O [13], which extends the OS-bypass features to bypass both the OS and the hypervisor for time-critical I/O operations. Through the high performance communication of VMM-bypass and the Nomad's efficient migration, we can realize the promise of cluster computing environments with both high performance and the benefits of modern VM technologies. Nomad can also be used for coordinated checkpointing of the virtual machines in a cluster environment. Our design is readily applicable to other virtual machine environments and other OS-bypass networks as well. The concepts presented can also be extended for process-level migration.

In summary, the main contributions of our work are:

- Discussing in-depth the challenges of transparently migrating modern OS-bypass interconnects in virtual machine environments, and proposing Nomad, a possible solution with namespace virtualization and coordination protocols.
- Implementing Nomad for a Xen-based cluster using InfiniBand. Based off of our earlier work of VMM-bypass I/O, our implementation maintains application transparency and requires no changes to native device drivers running in the Xen privileged domain, device firmware, or hardware.
- Evaluating our prototype on an InfiniBand cluster with various high performance computing (HPC) benchmarks. Our evaluation shows that together with Xen live migration, Nomad can be used efficiently even in environments with stringent requirements on communication performance.

The rest of the paper is organized as follows: In Section 2, we briefly introduce the background information, including the Xen VM environment, modern RDMA capable interconnects using InfiniBand as example, and VMM-bypass I/O. In Section 3, we discuss in detail the challenges of migrating OS-bypass interconnects. In Sections 4 and 5, we present the design and implementation of Nomad. Performance evaluation results are given in Section 6. In Section 7, we note several related issues and limitations of our current implementation and how they can be addressed in future. We discuss related work in Section 8 and conclude the paper in Section 9.

2 Background

In this section we discuss background information for our work. In Section 2.1, we introduce the OS-bypass approach of modern high speed interconnects and give an overview of InfiniBand architecture, which is a typical OS-bypass interconnect. In Section 2.3 we describe direct I/O access in virtual machines and how OS-bypass interconnects are supported by VMM-bypass I/O. Since our prototype is based on Xen virtual machine environment, we introduce Xen in Section 2.2.

2.1 OS-bypass I/O

Device I/O accesses have traditionally been carried out inside the OS kernel. This approach, however, imposes several overheads into the critical path such as context switches between user processes and OS kernels and extra data copies which degrade I/O performance [2]. It can also result in *QoS crosstalk* [27] due to lack of proper accounting for I/O access cost carried out by the kernel on behalf of applications.

To address these problems, a concept called user-level communication was introduced by the research community. One of the notable features of user-level communication is *OS-bypass*. Using this model, devices allow frequent and time-critical operations such as I/O communication be performed directly by user processes without involvement of OS kernels. Devices usually allow OS-bypass for frequent and time-critical operations while other operations, such as setup and management operations, are often handled by OS kernels. Due to these advantages, OS-bypass has been adopted by commercial products, many of which have become popular in areas such as high performance computing where low latency is vital to application performance.

The key challenge to implement OS-bypass I/O is to enable safe access to a device shared by different applications. To achieve this, OS-bypass capable devices usually require more intelligence in the hardware than traditional I/O devices. Typically, an OS-bypass capable device is able to present virtual access points to different user applications. Hardware data structures for virtual access points can be encapsulated into different I/O pages. With the help of the OS kernel, the I/O pages can be mapped into the virtual address spaces of different user processes. Thus, different processes can access their own virtual access points safely, with the protection provided by the virtual memory mechanism.

2.1.1 InfiniBand Architecture

InfiniBand [10] is a high speed interconnect offering OS-bypass features. InfiniBand host channel adapters (HCAs) are the equivalent of network interface cards (NICs) in traditional networks. InfiniBand uses a queue-based model for communication. A *Queue Pair (QP)* consists of a send queue and a receive queue, which hold work descriptors to transmit data. Once a work descriptor is posted to the QP, it is carried out by the HCA. The completion of communication events is reported through *Completion Queues (CQs)* using *Completion Queue Entries (CQEs)*. InfiniBand offers reliable connection service (RC) as well as Remote Direct Memory Access (RDMA). After QPs are created, they need to be explicitly bound together to establish a reliable connection (RC). RDMA operations can be carried over RC.

To ensure safe hardware access at the user level, InfiniBand requires all buffers involved in communication be registered. Upon the completion of registration, a local key and a remote key are returned, which will be used later for local and remote (RDMA) accesses.

A user communication library takes care of time-critical operations. In the Mellanox [15] approach, which represents a typical implementation of the InfiniBand specification, initiating data transmission includes copying a work descriptor to the user-space queue pair (QP) buffer and ringing a doorbell. Please note that when discussing InfiniBand, we refer to the Mellanox approach in this paper. Doorbells are rung by writing to the registers that form the *User Access Region (UAR)*, which is a 4k I/O page mapped into the virtual address space of a process, forming virtual access points to the HCA. The completion queue entries (CQEs) are also located in user space (CQ buffer) and can be directly accessed from the process virtual address space. These OS-bypass features make it possible for InfiniBand to provide very low communication latency. Figure 1 illustrates the architecture of OpenFabric Gen2 stack, which is a popular software stack for InfiniBand.

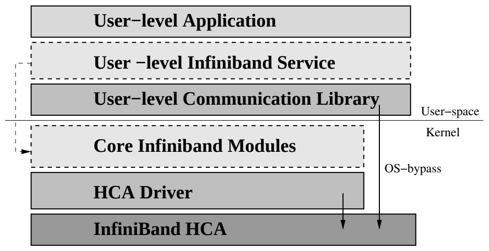

Figure 1: Architectural overview of OpenIB Gen2 stack

2.2 Xen Virtual Machine Monitor

Xen is a popular high performance virtual machine monitor originally developed at the University of Cambridge. It uses para-virtualization [36], in which host operating systems need to be explicitly ported to the Xen architecture. This architecture is similar to native hardware such as the x86 architecture, with only slight modifications to support efficient virtualization.

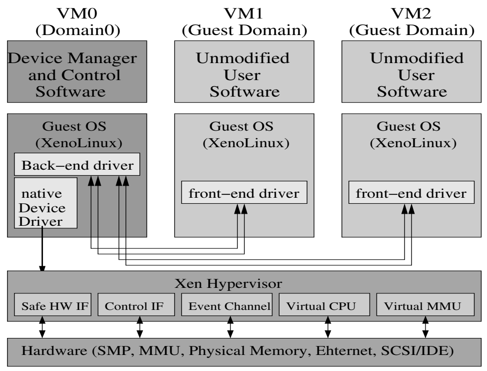

Figure 2: The structure of the Xen hypervisor, hosting three xenoLinux operating systems (courtesy [23])

Figure 2 illustrates the architecture of Xen. The Xen hypervisor is at the lowest level and has direct access to the hardware. Above the hypervisor are the Xen domains (VMs); many domains can be run simultaneously. A special *domain0*, which is created at boot time, hosts application-level management software and performs the tasks to create, terminate or migrate other domains.

To ensure manageability and safe access, device virtualization in Xen follows a split device driver model [8]. Each device driver is expected to run in an *isolated device domain (IDD)*, which hosts a *backend* driver to serve access requests from guest domains. Each guest OS uses a *frontend* driver to communicate with the backend. This split device driver model requires the development of frontend and backend drivers for each device class.

Xen supports hot migration, which transparently moves one running VM to another physical host without interruption of the services hosted on the virtual OS of the migrating VM. Xen uses a pre-copy approach, which iteratively copies the modified pages of memory from the source machine to the destination host. Because the VM is only paused at the final stage of migration, only a very short downtime is noticed by the user application. Migration of devices is handled by the para-virtualized device drivers. The front-end drivers receive *suspend* callbacks at the final migration stage when the VM is about to be paused and *resume* callbacks when the VM is resumed at the new host machine.

2.3 Direct I/O in Virtual Machines

Traditional I/O accesses in virtual machines involve either the hypervisor [35] or a device domain [8] to achieve device sharing and safe access. Such schemes, however, also introduce extra overhead to access I/O devices. This behavior is not desirable in many cases especially for cluster computing environment, where I/O performance is often critical to application performance. To overcome this shortcoming, several techniques have been proposed to directly access the I/O devices in VM environment without compromising the safety.

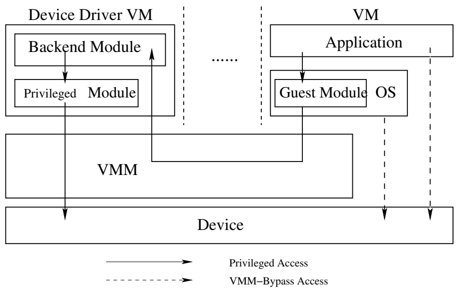

Figure 3: VMM-Bypass I/O

Our previous work proposed VMM-bypass I/O, which extends the idea of OS-bypass I/O to allow safe direct network I/O access in virtual machines. The architecture of VMM-bypass I/O is illustrated in Figure 3. We target network devices with OS-bypass capabilities. A device driver called *guest module* in the OS of the guest VM is responsible for handling all privileged accesses to the device. In order to allow I/O operations be carried out directly in the guest VM, the guest module must be able to create virtual access points on behalf of the guest OS and map them into the appropriate addresses (e.g., UAR for InfiniBand) of user processes. Since the guest module does not have direct access to the device hardware, a *backend module* in the device domain helps to provide such access to all the guest modules. In addition to serving as a proxy for device hardware access, the backend module also coordinates accesses among different VMs so that system integrity can be maintained. Once the virtual access points have been setup, applications in the guest VM can directly access the hardware, which brings close-to-native I/O performance.

Other direct I/O techniques include Xen PCI pass through [39], which passes a PCI device through to an unprivileged domain, who will then has the exclusive access to this specific PCI device. Thus the I/O performance can be greatly enhanced. PCI-Express I/O virtualization [21] also allows multiple operating systems running simultaneously within a single physical machine to natively share PCI-Express devices. To the best of our knowledge, there is no commercial product that supports this specification yet.

3 Challenges

Figure 4 illustrates an environment using virtual machines for cluster management. Each physical machine hosts several VMs which can run serial or parallel jobs. Migration of VMs plays several important roles in such an environment. First, the system administrator can move the VMs across nodes according to the work loads, which improves resource utilization. Second, for performance reasons, it is desirable to move the VMs hosting a parallel job to physical nodes adjacent in network topology whenever possible. Additionally, if physical nodes need to be shutdown for maintenance, hosted VMs can be migrated to other nodes. To achieve this, one of the basic requirements of VM migration in a cluster environment is to transparently keep alive the open network connections.

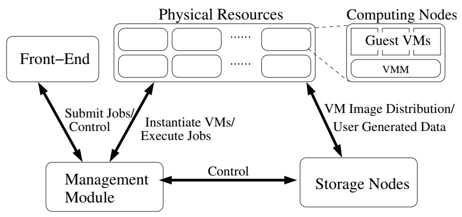

Figure 4: A VM-based cluster environment

Compared with traditional network devices such as

Ethernet, migration of modern OS-bypass interconnects are less studied. In this section we take a closer look at the challenges of migrating OS-bypass interconnects. We use InfiniBand and Ethernet as examples for OS-bypass interconnects and traditional network devices, respectively, in our description.

3.1 Location Dependent Resources

Many network resources associated with OS-bypass interconnects are location dependent, making them very difficult to migrate with the virtual operating system.

First, as mentioned in Section 2, to allow user level communication and remote memory access, the HCAs of modern interconnects will often manage some data structures in hardware. Software can only use opaque handle to access HCA resources. Taking InfiniBand as an example, software can only use QP numbers, CQ numbers, and local and remote keys to refer the connections, completion queues, and registered communication buffers, respectively. Many of those structures are maintained by the device and cannot be managed directly by software. Once a VM is migrated to another physical machine with a different HCA, the opaque handles are no longer valid. One approach is to attempt to reallocate the resources on the new host using the same handle values. This method requires changes to the device firmware which assigns the handles. Even with that additional complexity, we will have a problem if multiple VMs are sharing the HCA, because the handles may have already been assigned to other VMs.

Further, InfiniBand port addresses (local ID or LID) are associated with each port, and only one LID can be associated with each port. The mapping between the LIDs and physical ports are managed by external subnet management tools, making it difficult to change during migration. Also, since the LIDs can be used by other VMs sharing the same HCA, in many case it is not feasible to change them during migration. In contrast, both IP and MAC addresses used in Ethernet are device-transparent and can be associate with any Ethernet devices. Multiple MAC and IP addresses can also be associated with one device, which offers flexibility to migrate and share the network devices in VM environment.

3.2 User Level Communication

User level communication makes migration more difficult from at least two aspects:

First, besides kernel drivers, applications can also cache multiple opaque handles to reference the HCA resources. If those handles are changed after migration we cannot update those cached copies at the user level. Also, RDMA needs some handles (remote memory keys) be cached at remote peers, which makes the problem even more difficult. In contrast, applications for traditional networks generally use the sockets interface, where all complexities are hidden inside the kernel and can be changed transparently after migration.

Second, with direct access to the hardware device from the user level it is difficult to suspend the communication during migration. For traditional networks with socket programming, the kernel intercepts every I/O request, making it much easier to suspend and resume the communication during migration.

3.3 Hardware Managed Connection State Information

To achieve high performance and RDMA, OS-bypass interconnects typically store connection state information in hardware. This information is also used to provide hardware-level reliable service, which automatically performs packet ordering, re-transmission, etc. With hardware managed connection states, the operating system avoids the stack processing overhead and can devote more CPU resources to computation. This presents a problem for migration, however, since there is no easy way to migrate connection states between the network devices. Given this, the hardware cannot recover any dropped packets during migration. Meanwhile, any dropped or out-of-order packets may cause a fatal error to be returned an application since there is no software recovery mechanism.

In contrast, migration is relatively easier for a traditional TCP stack on Ethernet. The connections states are managed by the operating system. Thus it is usually sufficient to save the main memory during migration and all connection states will be migrated with the virtual OS. The in-flight packets during migration may be lost, but they can be recovered by the OS at TCP layer.

4 Design

In this section we present some of the design choices made for Nomad to address the challenges we explained in Section 3. We use namespace virtualization to virtualize the location dependent resources and a handshake protocol to coordinate among VMs to make the connection state deterministic during migration. We first focus on migrating VMs hosting applications using RC services only. Then we will also briefly mention how to deal with unreliable datagram (UD) services.

We use Xen and InfiniBand throughout our discussion. Xen and InfiniBand are each very typical in their domains, virtual machine monitors and OS-bypass interconnects, respectively. Thus, most of the issues discussed are common to other VM technologies and OS-bypass network devices.

4.1 Location Dependent Resources

As we have discussed in the last section, software can only use opaque handles to refer to HCA resources, which are location dependent. In some specific HCA implementations it may be possible to directly modify the resources, for example, "Memory Free" Mellanox InfiniBand cards store a majority of resources in main memory and access them through the PCI-Express bus. Such a design opens the chance to modify those memory areas directly to support migration. However, a design based on this scheme is very hardware dependent, so the first design choice we make is to strictly assume HCA resources are not modifiable directly. Thus, we need to free the location dependent resources before the migration starts and re-allocate them after the migration.

The opaque handles are assigned by the firmware, meaning they will be changed after the resources are reallocated. The main difficulty is how to approach the cached opaque handles at the user application. For example, a typical parallel application using InfiniBand reliable connections may cache the following information:

- LIDs (port addresses), which are exchanged among the processes involved after the application starts to address the remote peers;
- Queue Pair (QP) numbers after the QPs are created; To establish a reliable connection, each QP has to know both the LID and the QP number of the remote side.
- Local and remote memory keys after registration. The same keys must be used to reference the communication buffer for either local communication operations or remote access (RDMA).

All above handles may be changed upon migration. In order to maintain application transparency, we must ensure the application can still use the re-allocated resources with the old handles.

4.1.1 Nomad Namespace Virtualization

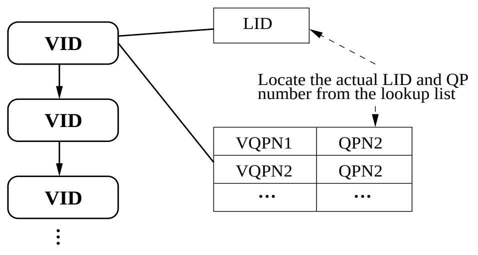

Figure 5: Destination lookup list

To achieve application transparency, we introduce a virtualization layer between the opaque handles that applications may use and the real handles associated with HCA resources. To virtualize the LID, we assign a VM identification (VID), which is unique within a cluster, to each VM once it is instantiated. The VID is returned to the application as LID. Similarly, once a QP is created, a virtual QP number (VQPN) is returned to application instead of actual QP number.

In order to determine the real LID and QP number when the applications try to setup the connection, Nomad maintains a *destination lookup list* similar to Figure 5 in the front-end driver. When an application tries to setup a connection to a remote peer represented by VID and VQPN, the front-end driver intercepts the connection request and replaces the VID and VQPN according the content in the translation table. If the driver is not able to locate an entry in the lookup list, which happens the first time the connection is established, it will send a lookup request to the front-end driver on the VM denoted by VID to "pull" the actual LID and QP numbers that should be used and put an entry in the lookup list.

Once a connection is setup between two processes on different VMs we consider those two VMs connected. When a VM is migrated, the same VQPN and VID then may correspond to a different QP number and LID. Nomad must make sure that any changes are reflected in the *destination lookup list* on each of the connected VMs. Each VM maintains a *registered list* at the front-end driver to keep track of the connected VMs. Once a VM receives a lookup request, it puts the remote VM into the *registered list*. After the migration, updates of new handles will be sent to all the connected VMs in the *registered list* to "push" the updates, which will be reflected in their lookup list. The connections can then be re-established automatically between the VMs without notifying the application, using the latest handles.

Once an application completes, the driver will determine all remote VMs to which it no longer has connections. It then sends an "unregister" request to those VMs to remove itself from their *registered list*. In this way we avoid unnecessary updates being sent among VMs.

4.1.2 Virtualizing Memory Keys

An additional challenge is handling of the memory keys, especially for remote memory keys, which are sent to peers for remote direct memory access (RDMA). If we use the lookup list and update schemes as mentioned above, each time a new remote memory key is used for communication, there will be no entry in the lookup list, requiring a query to the remote VM. This approach may impose a significant and unacceptable delay in the critical path of communication.

To avoid the extra cost of looking for updates, we make modifications based on the lookup list mentioned in the last section. We still keep a similar lookup list, as shown in Figure 6. Note that we need a translation table for the local keys because they are used by applications to reference the local memory buffers. We return the real memory keys as the "virtual" keys to the applications. Thus, if no entry is found corresponding to a key used by application, instead of querying for updates, Nomad will use the key directly for communication.

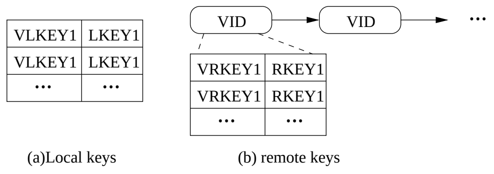

Figure 6: Nomad keeps: a) translation table for local keys; b) list of translation table for remote keys, updated upon peer's request

After migration we will get a new set of "physical" memory keys for the communication buffers. We then will update all connected peer VMs with the new keys, which will be kept in the lookup list.

The extra complexity caused by using real keys as virtual keys is that we may have a conflict of physical keys with virtual keys after migration, For example, on the origin host we first register a user buffer and get memory key 0x1000. After migration we re-register the buffer and get key 0x2000, creating an entry "0x1000→0x2000" in the lookup list. Now we register another buffer, this time we get key 0x1000 since it has not been used on the new host. Then if the application uses key 0x1000 to communicate, Nomad cannot distinguish which buffer it is referencing. To solve the conflict, we have to select a unused virtual key, say key 0x3000, and update all connected VMs with an entry "0x3000→0x1000". Connected VMs are the only locations where the keys could possibly be in use. We believe for memory keys, such conflicts will not occur often, reducing any possible overhead. For instance, InfiniBand firmware randomly select 32-bit keys, making the possibility of such conflicts very low.

We do not use this scheme for virtualizing LIDs or QP numbers. The main reason is that in case of conflict, we do not know which VM will use the LID/QP number to set up a connection. We will end up updating all active VMs in the cluster, which may be prohibitively expensive on large clusters. Also, connection setup is not in the communication critical path, thus there is no need for this extra complexity.

4.2 User-level Communication

With namespace virtualization the applications can use the same handle to access the HCA resources after migration. Even with this, we still need to suspend the network activity during the migration. Unlike the traditional TCP/IP stack where all communication goes through kernel, the user level communication leaves no central control point from where we can suspend the communication. Fortunately, almost no applications access the hardware directly. The user level communication is always carried out from a user communication library, which is maintained synchronously with the kernel driver. This allows us to intercept all send operations in this communication library. Taking InfiniBand as an example, we generate an event to the communication library to mark the QP suspended if we want to suspend the communication on that specific QP. If the application attempts to send a message to a suspended QP, we buffer the descriptors in the QP buffer, but do not ring the doorbell. When resuming the communication, we update every buffered descriptor with the latest memory keys and ring the doorbell, the descriptors then will be processed on the new HCA. This delays the communication without compromising the application transparency. Note that this scheme does not require extra resources to buffer the descriptors, because the QP buffers are already allocated.

4.3 Connection State Information

Since there is no easy way to manage the connection state information stored on the HCA, we work around this problem by bringing the connection states to a deterministic state. When the VM starts migrating, we not only mark all QPs as suspended, but also wait for all the outstanding send operations to finish during the suspension of communication. In this way, there will be no in-flight packets originating from the migrating VM.

Beyond that, we must avoid packets being sent to the migrating VM. Nomad achieves this by sending a suspend request to all the connected VMs. Upon receiving the suspend request the VM will notify the user communication library to mark the corresponding QP as suspended and wait for all outstanding send operations on that QP to finish. Note that communication on QPs to other VMs will not be affected. The *registered list* can be used to identify all the connected VMs.

After all communication on both the migrating VM and the connected VMs are suspended and all the outstanding sends finish, the connection states are deterministic thus need not be migrated. We simply need to resume the communication after the migration is done.

4.4 Unreliable Datagram (UD) Services

Besides RC service, most modern interconnects also provide unreliable datagram service (UD). UD service is easier to manage since we do not need to suspend the remote communication, lost packets during migration will be recovered by the application itself. Only the UD address handles need to be updated after migration; for InfiniBand these are the LID and QP number.

Updating the UD address can be done in the similar way as described in Section 4.1. For example, InfiniBand requires a UD address handle to be created before any UD communication takes place. Nomad checks with the destination VM, denoted by VID, for updates when creating the address handle. It is then registered with that VM. When a VM is migrated, it will update all VMs in the *registered list* with the new QP numbers and LIDs so the address handles will be re-created.

5 Implementation

In this section we present a prototype implementation of Nomad. We built the prototype based on our earlier work of *XenIB*, which virtualizes InfiniBand in the Xen environment with VMM-bypass I/O. Our implementation extends XenIB with migration functionalities. In the following section we first show the overall architecture of the prototype. Then we discuss the protocols to migrate one VM and how we extend the protocols to migrate a group of VMs. Finally, we discuss some optimizations with respect to scalability on large clusters.

5.1 Architecture

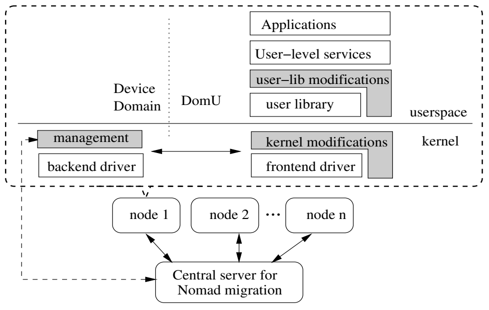

Figure 7: Architecture of Nomad

Figure 7 illustrates the overall architecture of Nomad, which consists of the following major components:

- Modified user communication library: The major modification includes code to suspend/resume communication on QPs, and a lookup list for memory keys as described in Section 4.1.2 for user level communication. All changes are transparent to the higher level InfiniBand services and applications.

- Modified InfiniBand driver in kernels of guest operating system: Major code changes include the suspend/resume callback interfaces interacting with XenBus interfaces[38]; the interaction with the user library notifying it to suspend/resume communication as necessary; the destination lookup list as described as Section 4.1; re-allocation of opaque handles after migration; and memory key lookup list for all kernel communication.

- Management network: This includes a central server and management module plug-ins at the privileged domain. All control messages (i.e. suspend or resume requests) are forwarded by a management module in the privileged domain. The central server keeps track of the physical host of each virtual machine so that control messages addressed by VID can be sent to the correct management module. Though the forwarding by a management module is not absolutely necessary, this design has its advantages. First, we can verify the correctness/validity of the control messages, so a malicious guest domain will not break the system security. Further, the privileged domain will not be migrated, so the management framework itself can be built on high speed interconnects like InfiniBand. If the management network involves the VMs that could be migrated, using InfiniBand may cause addition complexity.

5.2 Migrating a Single Virtual Machine

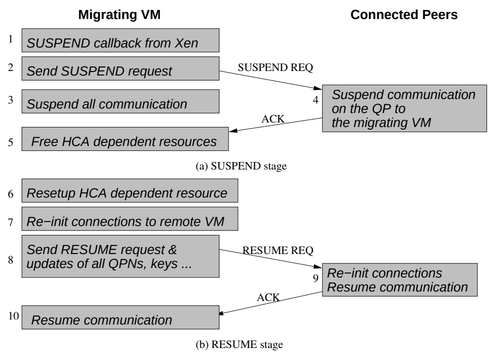
Migrating VM
Connected Peers
1 SUSPEND callback from Xen
2 Send SUSPEND request
SUSPEND REQ
3 Suspend all communication
4 Suspend communication on the QP to the migrating VM
ACK
5 Free HCA dependent resources
(a) SUSPEND stage
6 Resetup HCA dependent resource
7 Re−init connections to remote VM
8 Send RESUME request & updates of all QPNs, keys ...
RESUME REQ
9 Re−init connections Resume communication
ACK
10 Resume communication
(b) RESUME stage

Figure 8: Protocol for migrating one VM

Figure 8 illustrates the protocol of Nomad to migrate a VM. We use a two stage protocol, following the model of migrating para-virtualized devices in Xen. The front-end drivers go into a *suspend* stage after receiving a suspend callback from the hypervisor to get ready for migration. It goes into a *resume* stage after receiving the resume callback to restart communication on the new host.

The driver sends suspend requests to the connected VMs to suspend their communication. It will then suspend local communication in parallel. Once it receives the acknowledgments from all connected VMs, it frees the location dependent resources and finishes the suspend stage.

In the resume stage, the driver will first re-allocate all location dependent resources. It then sends update messages to all VMs in the *registered list*. Upon receiving the update, the connected VMs can re-establish the connection and resume the communication. After all connected VMs have acknowledged, the communication on the migrating VM will be resumed.

5.3 Migrating a Group of Virtual Machines

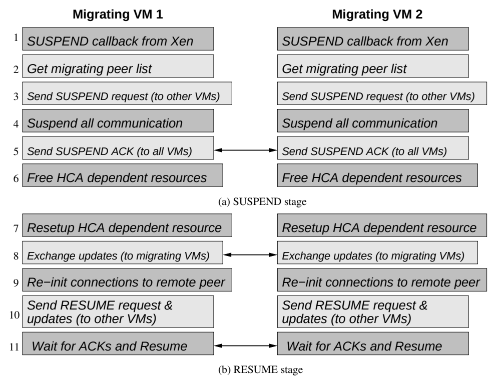
Migrating VM 1
Migrating VM 2
1 SUSPEND callback from Xen
2 Get migrating peer list
3 Send SUSPEND request (to other VMs)
4 Suspend all communication
5 Send SUSPEND ACK (to all VMs)
6 Free HCA dependent resources
(a) SUSPEND stage
7 Resetup HCA dependent resource
8 Exchange updates (to migrating VMs)
9 Re−init connections to remote peer
10 Send RESUME request & updates (to other VMs)
11 Wait for ACKs and Resume
(b) RESUME stage

Figure 9: Protocol for migrating two VMs

In some cases users need to migrate a group of virtual machines to new hosts. For instance, to bring down a set of physical nodes for maintenance. In this case, because of the existence of the central server as coordinator, we can simplify the control message exchange among the migrating VMs. We assume the central server will know the set of VMs that user wants to migrate simultaneously. Then during suspend stage, each migrating VM will query the server to get the list of connected VMs which are also migrating. It does not need to send suspension requests to those VMs, because they will suspend their communication regardless. Instead, all the migrating VMs will send the suspend acknowledgments directly to each other. For VMs not migrating the same protocol as in the last section is used.

During resume stage, however, extra steps are needed to exchange the updated resource handles among the migrating VMs before the connections between the QPs can be re-established with the correct resource handles (step 8). Flowchart for simultaneously migrating two virtual machines is shown in Figure 9.

5.4 Scalability

Jobs running in a cluster environment, especially HPC jobs, can involve as many as hundreds to thousands of nodes. Application such as MPI may use a static connection model, which means that each process will set up a reliable connection to every other processes. This may cause scalability issues for Nomad during migration. Sending suspension requests to so many connected VMs and waiting for replies may cause significant delay.

Fortunately, research on communication characteristics of parallel programs [31] indicates that not all pairs of processes in a parallel job communicate among each other with equal frequency. Figure 10 shows the average number of communicating peers per process in some scientific applications written in MPI. In many cases the majority of process pairs do not communicate between each other. Based on this observation, we introduce a concept called *active connections*.

Application	Number of Processes	Average Number of Distinct Destinations
sPPM	64	5.5
	1024	< 6
SMG2000	64	41.88
	1024	< 1023
Sphot	64	0.98
	1024	1
Sweep3D	64	3.5
	1024	< 4
Samrai 4	64	4.94
	1024	< 10
CG	64	6.36
	1024	< 11

Figure 10: Average number of communicating peers per process in several large-scale applications (Courtesy of [11])

Once a connection (QP) is created, by default it is in "non-active" mode. And no communication is allowed on "non-active" QPs, just like what we do on suspended QPs. During migration, we do not handshake with VMs connected with "non-active" QPs, which may significantly reduces the number of control messages sent.

When a process posts a send to a "non-active" QP, Nomad will first contact the remote side to see if any side of the connection has been migrated. If so, the VMs will exchange any possible updates on the QP numbers, LID or memory keys, and re-establish the connection if needed. After that, the QP is switched to "active" state and is ready for communication.

6 Performance Evaluation

In this section, we evaluate the performance of our prototype implementation of Nomad. We first evaluate the impact of VM migration on InfiniBand verbs layer micro-benchmarks, then we move to application-level HPC benchmarks. We focus on HPC benchmarks since they are typically more sensitive to the network communication performance and allow us to evaluate the performance of Nomad better. Since there are barely any HPC benchmarks directly written with InfiniBand verbs, we use benchmarks on top of MPI [28] (Message Passing Interface). We use MVAPICH [14, 18], a popular MPI implementation over InfiniBand, for this evaluation.

6.1 Experimental Setup

The experiments are carried out on an InfiniBand cluster. Each system in the cluster is equipped with dual Intel Xeon 2.66 GHz CPUs, 2 GB memory and a Mellanox MT23108 PCI-X InfiniBand HCA. The systems are connected with an InfiniScale InfiniBand switch. Besides InfiniBand, the cluster is also connected with Giga-bit

Ethernet as the control network. Xen-3.0 with the 2.6.16 kernel is used on all computing nodes. Domain 0 (the device domain) is configured to use 512 MB and each guest domain runs with a single virtual CPU and 256 MB memory. Each physical node hosts one VM so that there is one spare CPU to provide computing resources needed in the pre-copy stage of the Xen live migration. This allows us to reduce the impact of the migration cost of Xen on our experiments.

6.2 Micro-benchmark Evaluation

In this subsection, we evaluate the impact of migration on micro-benchmarks. We use the standard *Perftest* benchmarks provided with the OpenFabrics Gen2 stack. They consist of a set of InfiniBand verb layer benchmarks to evaluate the basic communication performance between two processes. We ran the tests on two VMs, with each of them hosting one process. We measure the performance reported by the benchmarks while migrating the VMs, one at a time.

The RDMA latency tests were carried out in a ping-pong fashion. A process RDMA writes to the peer and the peer acknowledges with a RDMA write after it detects message arrival. This process repeats one thousand times and the worst and median half round-trip time are reported. We slightly modify the benchmark to keep measuring the latency in loops. Figure 11 shows the RDMA latency reported in each iteration. The worst latency is always higher than the typical latency due to process skews at the first few ping-pongs. We also observe that during iterations that we migrate the VMs, the worst latency increases to around 90 ms from under few hundred micro-seconds. This approximates the migration cost when migrating simple programs.

In the bandwidth tests, a sender sends a number of messages to a receiver and then waits for an acknowledgment. The bandwidth is obtained by dividing the number of bytes transferred from the sender by the elapsed time of the test. Since the migration cost is amortized among all the messages, we do see a degradation on the average bandwidth reported during the migration period, as shown in Figure 12.

We see the same trend for latency and bandwidth using send/receive verbs instead of RDMA. We also observe no noticeable overhead is caused by Nomad on either latency or bandwidth if no migration is taking place.

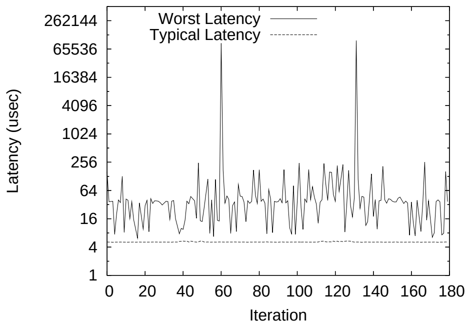

Figure 11: Impact of migration on RDMA latency

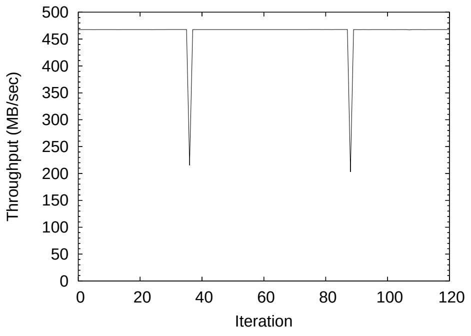

Figure 12: Impact of migration on RDMA bandwidth

6.3 HPC Benchmarks

In this subsection we examine the impact of migration on HPC benchmarks. We use the NAS Parallel Benchmarks (NPB) [17] for evaluation, which are a set of computing kernels widely used by various classes of scientific applications. Also, the benchmarks have different communication patterns which allows us to better evaluate the overhead of Nomad.

The benchmark sizes are selected so that they run long enough for us to migrating VMs. We run the benchmarks on four processes with each VM hosting one computing process. We then migrate VMs one at a time during the process to see the impact of migration. Figure 13 compares the performance between running NAS on native systems, with Nomad but no migration, migrating a VM once, and migrating a VM twice. As we can see from the graph, Nomad causes only slight overhead (less than 1%) if there is no migration, which conforms to our earlier evaluation on XenIB [34]. Each migration causes 0.5 to 2 seconds increase of total execution time, depending on the benchmarks.

6.4 Analysis of Migration Cost

In this subsection we take a closer look at the migration cost caused by Nomad. As we have discussed, the migration process can be divided into suspend and resume stages. We analyze the cost of both of these stages.

6.4.1 Suspend Stage of Nomad

The overhead of the suspend stage can be broken down into two parts, time to wait for local and remote peers suspend communication and the time to free local resources. Suspending local communication occurs in parallel with suspending remote communication. Suspension of remote communication typically takes a relatively

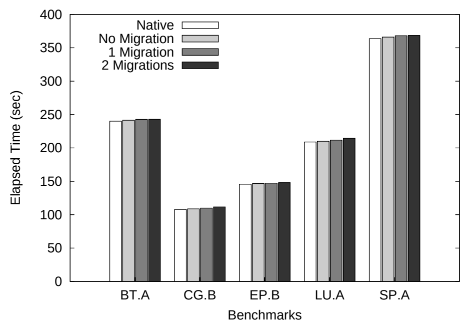

Figure 13: Impact of migration on NAS benchmarks (migrating one VM out of four)

larger amount of time since there is extra overhead to synchronize through the management network. To estimate the overhead of synchronization, we also measure the cost of suspending communication on the remote peers, that is, from the time it receives the request to suspend communication, to the time that the communication is suspended.

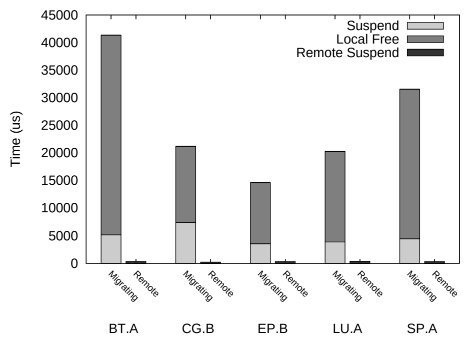

Figure 14: Suspend time running NAS benchmarks

We profile each of these stages. The results are shown in Figure 14. We observe that the remote side takes a few hundred milliseconds to suspend communication on the QP to the migrating VM. Considering the synchronization overhead, the migrating VM takes typically a few microseconds to wait for communication suspension. The major cost observed is the time to free local resources, which takes 10 to 36 ms based on the resources allocated. Because we fix the number of peers in the job, we see a strong correlation between the time to free the resources and the amount of memory registered. For instance, for NAS-BT, the VM has registered 7540 pages of memory by the time it is migrated, and it takes 36 ms to free the local resources. For NAS-EP, where only 2138 pages are registered by the time the VM is migrated, and it takes only around 11 ms to free all the resources. This suggests that a scheme which delays the freeing of resources will potentially reduce the migration cost further: the VM can be suspended without freeing HCA resources; and the privileged domain can track the resources used by the VM and free them after VM migration.

6.4.2 Resume Stage of Nomad

The cost at the resume stages mainly includes the time to re-allocate the HCA resources and the time to resume the communication. Similar to our analysis of the suspend stage, we also profile the time taken on the remote peers to resume the communication. The time is measured from the resume request arrival to send of the acknowledgment; this time includes updating the resource handle lookup list, re-establishing the connections, and reposting the unposted descriptors during the migration period. Time to resume local communication on the migrating VM has very low overhead because there are no unposted descriptors.

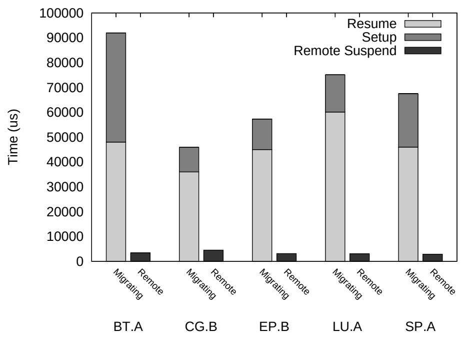

Figure 15: Resume time running NAS benchmarks

As shown in Figure 15, re-allocating the HCA resources is still a major cost of the resume period. We see the same correlation between the amount of registered memory and the time to re-allocate resources. The time varies from around 49ms for NAS-BT to 13ms for NAS-EP in our studies. This suggests a pre-allocation of the resources can help reducing the migration cost too. Pre-allocation of resources will also allow better migration safety, we will discuss more in Section 7.

Our evaluation shows slightly more time to resume than to suspend the communication on remote peers. This difference is largely due to the process to update the lookup list and to re-establish the connections. We also observe that the migrating VM resume time is much greater that of the non-migrating peers. For instance, NAS-BT takes around 3.2 ms on the non-migrating peers to resume communication, but the time measured on the migrating VM is around 48 ms. There are three possible reasons that may cause this longer delay: first, more traffic is on the control networks, mainly due to the update list of the resource handles; second, we have to inform the central server the new location of the migrating VM; third, the non-migrating peers take longer time to process the resume request, which increase the chance to have longer skews.

6.5 Migrating Multiple VMs

We also measure the overhead of migrating multiple VMs simultaneously while running the applications. We run the NAS benchmarks on 4 processes located on 4 different VMs. During the execution we migrate all 4 VMs simultaneously. Thus the protocols as described in Section 5.3 will be used.

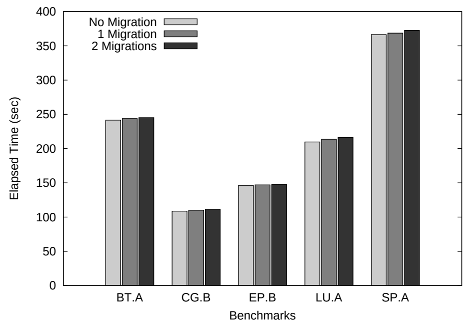

Figure 16: Impact of migration on NAS benchmarks (migrating all four VMs)

Figure 16 shows the comparison of the total execution time. We observe from the graph that the average per checkpoint cost is not increased much as compared to the case of migrating one VM. Despite this, we observe larger variation of the results we collected. We believe the skew between processes is the major cause for the variation. The skew can be mainly due to two reasons:

- Though we start migration of all 4 VMs at the same time, Xen may take a varied amount of time to precopy the memory pages, thus the time each process enters the Nomad suspend stage is different.
- Each VM may not take the same amount of time to suspend the local communication and to free the local resources. Similarly, the time to re-allocate the resources on the new host and resume communication can also be different.

Figure 17 shows a breakdown of suspend time spent on each of the migrating VM. As we can see, VM1 takes a significantly shorter time to wait for remote communication suspension than other three VMs. It clearly indicates that VM1 enters the suspend stage later than other three: all other three VMs have already suspended their traffic and are waiting to synchronize with VM1.

7 Discussions

In this section we discuss several issues related to Nomad. Some of these are limitations of the current prototype implementation, such as the migration safety issues.

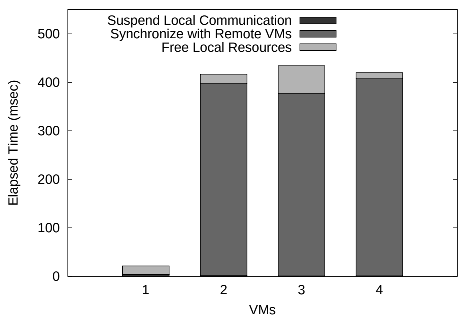

Figure 17: Suspend time running NAS-CG

We also propose several possible improvements for Nomad.

7.1 Safe Migration

A desirable feature of migration is safety. The Xen migration follows a pattern that should at no time leave a virtual OS more exposed to system failure than when it is running on the original single host [5]. For instance, if the remote host does not have enough resources to host the migrating VM, the migration process will be aborted at early stage, leaving the virtual OS running on the old host without being affected. Unfortunately, Nomad will allocate HCA resources on the new host after the VM is suspended on the old host. It risks the failure of a virtual OS if the HCA resources are not available on the new host.

A possible solution is to contact the privileged domain of the new host trying to pre-allocate the resources before suspending the virtual OS. A more elegant approach is to introduce a "pre-suspend" stage to the current Xen migration interface of the "suspend" and "resume" stages. During this stage all HCA resources can be pre-allocated. We plan to explore both directions.

7.2 Scalability of the migration protocols

Nomad requires coordination among peers to update all opaque handles and avoid delivery failure during the migration process. When migrating a VM involved in large-scale parallel jobs the coordination may cause high synchronization delays.

Besides the concept of "non-active" connections we used in the prototype implementation, several methods can be used to improve the scalability of the protocol. For instance, instead of sending the suspend/resume request to all peers one by one, we can distribute the request and gather the acknowledgments through a tree-based structure. Using this mechanism can significantly speedup the synchronization process. In addition, we can take advantage of hardware multicast, which is provided by many high speed interconnects like InfiniBand

and Quadrics, to expedite the distribution process. Using these methods the protocol overhead caused by the network transfer time can be significantly reduced.

7.3 Single-point of Failure

Using a central server in our management network creates a single point of failure. If the machine acting as this central server fails, none of the other VMs can be migrated successfully.

There have been many studies to use backup servers to improve the system robustness. A simple solution is to duplicate a central server as the backup server, which will become active if the main server fails. Alternatively for a large cluster, several servers can be active simultaneously, which will improve the performance if many queries are taking place. For both solutions we need to make sure the query answers provided by all servers are coherent.

8 Related Work

In this paper we discussed the migration of OS-bypass interconnects in virtual machine environment. OS-bypass is a feature found in user-level communication protocols such as active messages [33], U-Net [32], FM [20], VMMC [3], and Arsenic [24]. Later it has been adopted by the industry [7, 10] and has been incorporated into various commercial products [16, 25].

Our implementation is based on our earlier work of VMM-bypass I/O [13]. VMM-bypass I/O extends the idea of OS-bypass to VM environments. Using this method, I/O and communication operations can be initiated directly by userspace applications, bypassing the guest OS, the VMM, and the device driver VM. VMM-bypass also allows an OS in a guest VM to carry out many I/O operations directly, although virtualizing interrupts still needs the involvement of the VMM. It also avoids the additional cost of involving the VMM or a privileged VM to handle I/O operations, such as the approaches used in VMware Workstation [30], VMware ESX Server [35], and Xen [8]. Instead, VMM-bypass makes use of intelligence in modern high speed network interfaces, limiting it to a relatively small range of devices which are used mostly in high-end systems. The traditional approaches can be applied to much wider ranges of devices.

Current virtual machine technologies have provided several solutions for migration of traditional network devices like Ethernet. The solution used in Xen [5] is based on the observation that the network interfaces of the source and destination machines typically exist on a single switched LAN. The migrating virtual OS will carry its IP address. A unsolicited ARP reply will be generated to advertise the IP has been moved. However, the location dependent resources and the need for hardware level reliable service have make additional challenges for the migration of OS-bypass networks.

Some previous work on process-level migration also have addressed the issue of network migration. For example, Zap [19] adopts Virtual Network Address Translation (VNAT) [29] which intercepts all network packets and dynamically translates between the address seen by the pod and the physical address. Another method is to use a "home node" approach, as is used in Mobile IP [1]. In this method a home node will re-route packets sent to the default or old address to the current address. These schemes, however, due to OS-bypass communication and performance reasons, can not be utilized in cluster environments where communication performance is extremely important.

Both the VMM-bypass I/O and the Nomad migration require a para-virtualization approach. As a technique to improve VM performance by introducing small changes in guest OSes, para-virtualization has been used in many VM environments [4, 9, 37, 6]. Essentially, para-virtualization presents a different abstraction to the guest OSes than native hardware, which lends itself to easier and faster virtualization.

9 Conclusions and Future Work

In this paper we present Nomad, a design for migrating modern interconnects with OS-bypass features, focusing on cluster environments running VMs. We discussed in detail the challenges of migrating modern interconnects due to hardware level reliable services and direct I/O accesses. We proposed a possible solution based on namespace virtualization and handshake protocols. To demonstrate our ideas, we present a prototype implementation of Nomad based on the Xen virtual machine monitor and VMM-bypass I/O with InfiniBand. We elaborated on the detailed design issues and possible improvements with respect to scalability on large scale clusters. Our performance analysis shows that Nomad can achieve efficient migration of network resources.

We are working on improving the safety of Nomad migration by pre-allocating resources before the VM suspends. We plan to further reduce the migration overhead of Nomad and improve the scalability on large scale clusters. We have proposed several possible approaches in Section 6 and Section 7 and plan to study the pros and cons of those schemes. We plan to use high speed interconnects to accelerate the Nomad control and the Xen migration traffic because currently they go through Ethernet. We also plan to explore solutions to achieve interoperability of Nomad and unmodified hosts running native operating systems, where the handshake will be impossible.

Acknowledgments

This research is supported in part by the following grants and equipment donations to the Ohio State University: Department of Energy's Grant #DE-FC02-01ER25506; National Science Foundation grants #CNS-0403342 and #CCR-0509452; grants from Intel, Mellanox, Sun, Cisco, and Linux Networx; and equipment donations from Apple, AMD, IBM, Intel, Microway, Pathscale, Silverstorm and Sun.

References

[1] RFC 2002: Mobile IP. http://www.ietf.org/rfc/rfc2002.txt.

[2] R. A. F. Bhoedjang, T. Ruhl, and H. E. Bal. User-Level Network Interface Protocols. *IEEE Computer*, pages 53–60, November 1998.

[3] M. Blumrich, C. Dubnicki, E. W. Felten, K. Li, and M. R. Mesarina. Virtual-Memory-Mapped Network Interfaces. In *IEEE Micro*, pages 21–28, Feb. 1995.

[4] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4):412–447, 1997.

[5] C. Clark et al. Live Migration of Virtual Machines. In *Proceedings of 2nd Symposium on Networked Systems Design and Implementation*, 2005.

[6] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 164–177, October 2003.

[7] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. Merritt, E. Gronke, and C. Dodd. The Virtual Interface Architecture. *IEEE Micro*, pages 66–76, March/April 1998.

[8] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the xen virtual machine monitor. In *OASIS ASPLOS Workshop*, 2004.

[9] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum. Cellular disco: resource management using virtual clusters on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 18(3):229–262, 2000.

[10] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2.

[11] J. Wu, J. Liu, P. Wyckoff, and D. K. Panda. Impact of On-Demand Connection Management in MPI over VIA. In *Proceedings of Cluster '02*.

[12] K. Koch. How does ASCI Actually Complete Multi-month 1000-processor Milestone Simulations? In *Proceedings of the Conference on High Speed Computing*, Gleneden Beach, Oregon, 2002.

[13] J. Liu, W. Huang, B. Abali, and D. K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. In *Proceedings of 2006 USENIX Annual Technical Conference*, June 2006.

[14] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *Proceedings of 17th Annual ACM International Conference on Supercomputing (ICS '03)*, June 2003.

[15] Mellanox Technologies. http://www.mellanox.com.

[16] Myricom, Inc. Myrinet. http://www.myri.com.

[17] NASA. NAS Parallel Benchmarks. http://www.nas.nasa.gov-/Software/NPB/.

[18] Network-Based Computing Laboratory. MVAPICH: MPI for InfiniBand on VAPI Layer. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/index.html.

[19] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[20] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of the Supercomputing*, 1995.

[21] PCI-SIG. PCI I/O Virtualization. http://www.pcisig.com-/news_room/news/press_releases/2005_06_06.

[22] F. Petrini, D. J. Kerbyson, and S. Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *Proceedings of SC '03*, Washington, DC, USA, 2003.

[23] I. Pratt. Xen Virtualization. Linux World 2005 Virtualization BOF Presentation.

[24] I. Pratt and K. Fraser. Arsenic: A User-Accessible Gigabit Ethernet Interface. In *INFOCOM*, pages 67–76, 2001.

[25] Quadrics, Ltd. QsNet. http://www.quadrics.com.

[26] M. Rosenblum and T. Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *IEEE Computer*, pages 39–47, May 2005.

[27] S. M. Hand. Self-Paging in the Nemesis Operating System. In *Proceedings of the 3rd Operating Systems Design and Implementation (OSDI)*, pages 73–86, 1999.

[28] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI–The Complete Reference. Volume 1 - The MPI-1 Core, 2nd edition.* The MIT Press, 1998.

[29] G. Su and J. Nieh. Mobile Communication with Virtual Network Address Translation. Technical Report CUCS-003-02, Columbia University, Feb 2002.

[30] J. Sugerman, G. Venkitachalam, and B. H. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of USENIX*, 2001.

[31] J. S. Vetter and F. Mueller. Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2002.

[32] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In *ACM Symposium on Operating Systems Principles*, 1995.

[33] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *International Symposium on Computer Architecture*, pages 256–266, 1992.

[34] W. Huang, J. Liu, B. Abali and D. K. Panda. A Case for High Performance Computing with Virtual Machines. In *Proceedings of the 20th ACM International Conference on Supercomputing*, 2006.

[35] C. Waldspurger. Memory resource management in vmware esx server. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[36] A. Whitaker, M. Shaw, and S. Gribble. Denali: Lightweight virtual machines for distributed and networked applications. In *Proceedings of the USENIX Annual Technical Conference, Monterey, CA*, June 2002.

[37] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of 5th USENIX OSDI, Boston, MA*, Dec 2002.

[38] Xen Wiki. http://wiki.xensource.com/xenwiki/xenbus.

[39] XenSource. http://www.xensource.com/.