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 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 need 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.

Categories and Subject Descriptors D.4.4 [*OPERATING SYSTEMS*]: Communications Management—Network communication; C.2.5 [*COMPUTER-COMMUNICATION NETWORKS*]: Local and Wide-Area Networks—High-speed

General Terms Design, Performance

Keywords Virtual Machines, Migration, High Speed Interconnects, Xen, InfiniBand

1. Introduction

Virtual machine (VM) technologies are experiencing a resurgence in recent years in both industry and academia [25]. With the virtual hardware interfaces provided by virtual machine monitor (VMM), many different guest VMs can be hosted simultaneously in a single physical machine. Virtual machine environments provide 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 [6]. It serves as a base for modern system management frameworks to target performance, scalability, and system management problems caused by today's ultra-scale clusters [22, 12].

Recently, network interconnects providing low latency (less than 5μs) and very high bandwidth (multiple Gbps) are emerging, such as InfiniBand [11], Myrinet [16], Quadrics [24], 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 additionally allows processes on remote nodes to access certain memory buffers of a local process. Excellent performance and flexibility provided by these features make modern interconnects widely adopted in cluster environments, which typically host data center or high performance computing (HPC) applications.

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 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 in cluster environments. First, the intelligent NICs (Network Interface Card) required for these networks manage large amounts of location dependent resources which are kept transparent to both applications and operating systems and cannot be migrated with the OS instances. This makes the existing solutions of current VM technologies, which target TCP/IP networks, less applicable. Further, applications in cluster environments using special program interfaces of high speed interconnects typically expect reliable services at the NIC level. Thus, it is important to make migration transparent to applications and to avoid packet drops or out-of-order delivery during migration. Online maintenance may not be the only purpose of VM migration in cluster environment. VM migration can be carried out for load balancing, where jobs are migrated to servers with lighter loads, or for performance, where VMs involved in a large parallel job should be migrated to servers close by in net-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

VEE'07, June 13–15, 2007, San Diego, California, USA.

Copyright © 2007 ACM 978-1-59593-630-1/07/0006...$5.00

work topologies. As a result, efficiency is also an important factor of migration. Otherwise, benefits gained by migration can be easily overshadowed by the migration overhead.

In this paper, we 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 resources. We also devise a coordination protocol to avoid packet drops or out-of-order communication during migration. To demonstrate our design, 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] that extends the OS-bypass features to bypass both the OS and the hypervisor for time-critical I/O operations. Our implementation includes three parts: modified user level communication library, which allows us to suspend and resume communication during migration without application level modifications; modified device drivers in guest kernels, which frees and re-allocates communication resources before and after migration, respectively; and a coordination framework, which includes coordinators in the privileged domains and a central server. The user library and the guest kernel modifications also realize the namespace virtualization. Note that no changes to applications or the native device drivers in the privileged domain are needed. 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 discuss our experiences with Nomad and propose some hardware changes to OS-bypass networks which can ease migration. 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 the 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 [3]. It can also result in *QoS crosstalk* [26] 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, while other operations, such as setup and management operations, are often handled by OS kernels. 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 [11] 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)*. All other detailed complexities of communication are hidden from users. 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).

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[1], which represents a typical implementation of the InfiniBand specification, initiating data

[1] Please note that when discussing InfiniBand, we refer to the Mellanox approach in this paper.

transmission includes copying a work descriptor to the user-space queue pair (QP) buffer and ringing a doorbell. 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. 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.

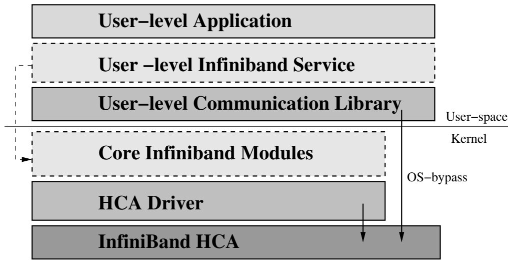

Figure 1. Architectural overview of OpenFabrics Gen2 stack [19]

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-virtualized [36] Xen architecture. This architecture is similar to the native hardware such as the x86 architecture, with only slight modifications to support efficient virtualization.

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 [9]. 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. To handle Ethernet devices, Xen suspends the network traffic of the migrating OS at the *suspend* stage and resumes communication at the *resume* stage on the new host. The IP and MAC addresses are migrated with the OS instance. Dropped or out-of-order packets caused by migration is handled by the TCP layer of the migrating OS.

2.3 Direct I/O in Virtual Machines

Traditional I/O accesses in virtual machines involve either the hypervisor [35] or a device domain [9] 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.

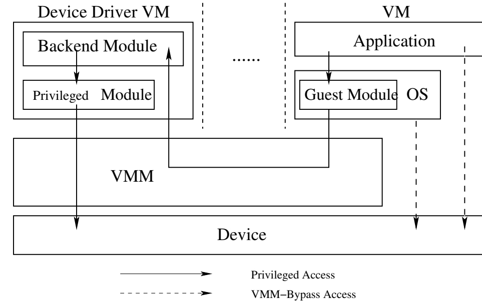

Figure 2. VMM-Bypass I/O

Our previous work proposed VMM-bypass I/O, which extends 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 2. 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.

3. Challenges for Migrating OS-bypass Networks

Figure 3 illustrates an environment using virtual machines for cluster management. Each physical machine hosts several VMs which can run serial or parallel jobs. System administrators can move the VMs across nodes for load balance or for online maintenance. Also, for long running parallel jobs it is desirable to move the participating VMs to physical nodes adjacent in network topology whenever possible. In either case, the basic requirements of VM migration in a cluster environment include the transparency to applications (keep alive the open network connections), as well as low impact on application performance.

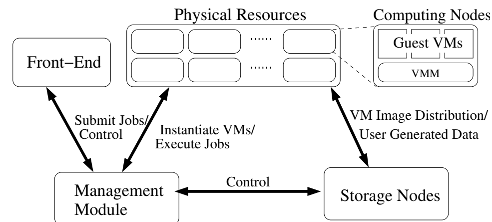

Figure 3. A VM-based cluster environment

Compared with traditional network devices such as Ethernet, migration of modern OS-bypass interconnects have been studied less. 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 migrating OS instance with application transparency.

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 use opaque handle to access HCA resources but cannot manage them directly. 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 associated 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. For example, each Xen domain has its own IP and MAC address that are associated with the physical Ethernet devices. Those addresses can be easily migrated with domains, and can be re-associated to the new hardware.

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 to an application since there is no software recovery mechanism.

In contrast, migration is easier for a traditional TCP stack on Ethernet. The connection 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. For example, Xen does not make special effort to recover any lost or out-of-order IP packet during migration, such IP level errors are recovered by the OS at the TCP layer.

4. Detailed Design Issues of Nomad

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 support 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, software can only use opaque handles to refer to HCA resources. All details of connection-level states are managed by HCAs and cannot be directly accessed or modified. Thus, we need to free the location dependent resources before the migration starts and re-allocate them after the migration. The major complexity comes from the location dependent opaque handles. These handles are assigned by the firmware and can only be used to access local HCA resources. It means they must be changed after the resources are re-allocated on the new host. However, they can be cached by user applications to access the HCA resources. How to approach these cached opaque handles is a challenging task. For example, a typical parallel application using InfiniBand cache the following opaque handles:

- LIDs (port addresses), which are exchanged among the processes involved after the application starts to address the remote peers.
- Queue Pair numbers (QPN) 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 that the application can still use the re-allocated resources with the old handles.

4.1.1 Nomad Namespace Virtualization

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 (VLID), which is unique within a cluster, to each VM once it is instantiated. The VLID 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 QPN.

In order to determine the real LID and QPN when the applications try to setup a connection, Nomad maintains a *destination mapping table* similar to Figure 4 at the front-end driver. When an application tries to setup a connection to a remote peer represented by VLID and VQPN, the front-end driver intercepts the connection request and replaces the VLID and VQPN according the content in the mapping table. The driver may not be able to locate the entry

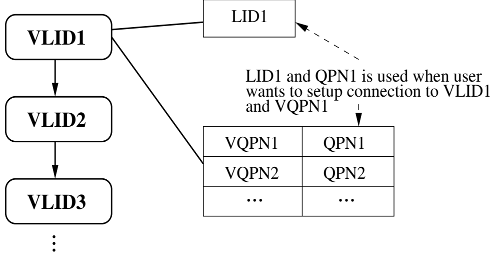

Figure 4. Destination mapping table

in the table, which happens the first time the connection is established. In that case it will send a lookup request to the front-end driver on the VM denoted by VLID to "pull" the real LID and QPN that should be used and create an entry in the mapping table.

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 VLID then may correspond to a different QP number and LID. Nomad must make sure that any changes are reflected in the *destination mapping table* on each of the connected VMs. To achieve that, 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 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 mapping table. 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 RDMA. If we use similar approach as above, say generating globally unique keys for each memory registration, then we have to update mapping tables both locally and at the remote side for each memory registration. Otherwise, each time a new remote memory key is used for communication, there will be no entry in the mapping table, requiring a query to the remote VM. Since both memory registration and communication can be in the critical path of application, in either cases there may be significant and unacceptable delays.

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 mapping table, as shown in Figure 5. Note that we need a mapping table for the local keys too 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. After migration we will get a new set of "physical" memory keys for the communication buffers. We will then update all connected peer VMs with the new keys, which will be kept in their own mapping tables.

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 mapping. 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 an 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 selects 32-bit keys, making the possibility of such conflicts very low.

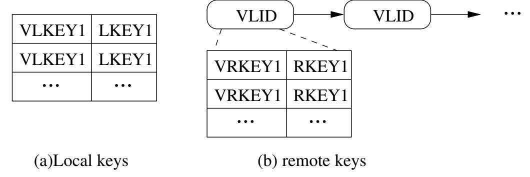

Figure 5. Nomad keeps: a) mapping table for local keys; b) list of mapping table for remote keys, updated upon peer's request

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 typically 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 application accesses 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 so that the requests are not issued to HCA. 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 communication without compromising 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 (QP) 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.

We must also avoid packets being sent to the migrating VM. Nomad achieves this by sending a suspend request to all the connected VMs. Upon receiving a suspend request, the connected VM will notify the user communication library to mark the correspond-

ing 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 all the migrating and the connected VMs are suspended and all the outstanding sends finish, the connection states are deterministic and 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 (UD) service. UD service is easier to manage since we do not need to suspend the remote communication, lost packets during migration will be recovered by 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 a 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 VLID, 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. Nomad Architecture for Xen and InfiniBand

In this section we present a prototype of Nomad over Xen and InfiniBand. 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 to further reduce the overhead of Nomad during migration. Please note that Nomad targets cluster environment, where all parties involved in the same parallel jobs are trusted to be secure and reliable.

5.1 Architecture

Figure 6 illustrates the overall architecture of Nomad, which consists of the following major components:

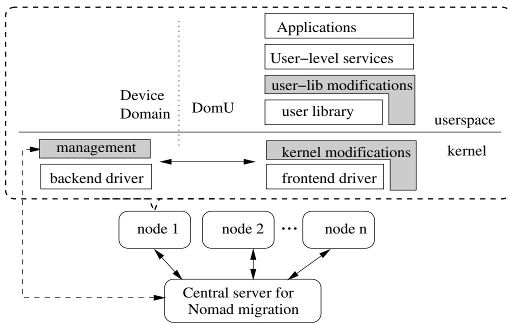

Figure 6. Architecture of Nomad

- 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. Note that 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 mapping tables as described as Section 4.1; re-allocation of opaque handles after migration; and memory key mapping tables 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 this forwarding 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 (which may not belong to the same parallel job) will not break the migration protocols. 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 additional complexity. Note that the central server does not necessarily affect system scalability on large node clusters. It is only accessed to resolve the actual location of VMs. All communication does not go through this central server.

5.2 Migrating a Single Virtual Machine

Figure 7 illustrates the protocol of Nomad to migrate a VM. We use a two stage protocol, following the model of migrating paravirtualized 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. At the *suspend* stage, the driver sends suspend requests to the connected VMs to suspend their communication. Local communication is then suspended in parallel. Once acknowledgments have been received from all connected VMs, location dependent resources are freed and the suspend stage is finished. In the resume stage, the driver will first re-allocate all location dependent resources and then send update messages to all VMs in the *registered list*. Upon receiving the update, 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.

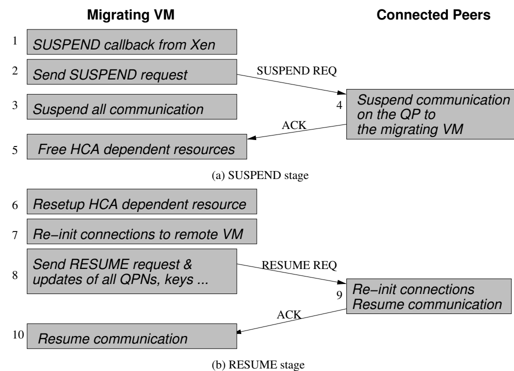

Figure 7. Protocol for migrating one VM

In some cases users need to migrate a group of virtual machines to new hosts. In this case, because of the existence of the central

server as a coordinator, we 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 the suspend stage, each migrating VM will query the server to get the list of connected VMs which are also migrating. Suspension requests are not sent to those VMs, because they will suspend their communication regardless. Instead, all migrating VMs will send the suspend acknowledgments directly to each other. During the 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 (before step 7 in Figure 7) with the correct resource handles.

5.3 Optimization

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

Fortunately, earlier studies [2] reveal that not necessarily all process pairs communicate between each other even though the connections are created. To further reduce the synchronization overhead, we introduce a concept called *active connections*.

Once a connection (QP) is created, by default it is in “non-active” mode, with no communication is allowed on “non-active” QPs. Thus during migration, we need not handshake with VMs connected with “non-active” QPs, which may significantly reduce 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 retrieve 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 few HPC benchmarks directly written with InfiniBand verbs, we use benchmarks on top of MPI [27] (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 Gigabit 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. Because Xen migration transfers memory pages over TCP/IP networks, which requires heavy CPU resources, we host one VM on each physical node. In this way, there is one spare CPU to handle the memory page transfer, which separates the overhead of Nomad from other migration costs. This allows us to better evaluate our implementation.

6.2 Micro-benchmark Evaluation

In this subsection, we evaluate the impact of Nomad on micro-benchmarks. We use *Perftest* benchmarks provided with the OpenFabrics 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.

We first measure performance numbers with all the *Perftest* benchmarks without migration. We observe no noticeable overhead caused by Nomad as compared to our original VMM-bypass I/O in [13], on either latency or bandwidth. This means the overhead of the namespace virtualization is negligibly small.

Next we measure the migration downtime using RDMA latency test. It is a ping-pong test that a process RDMA writes to the peer and the peer acknowledges with another RDMA write. This process repeats one thousand times and the worst and median half round-trip time are reported. We modify the benchmark to keep measuring the latency in loops. Figure 8 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 microseconds. This approximates the migration cost when migrating simple programs.

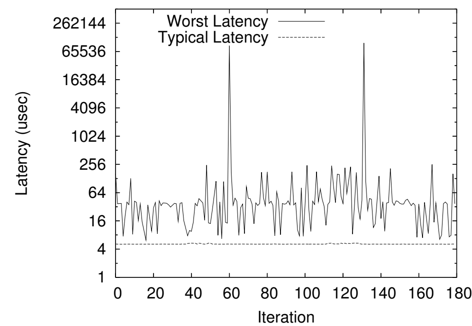

Figure 8. Impact of migration on RDMA latency

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, from the ones hardly communicate (EP) to communication intensive ones like CG and FT. This allows us to better evaluate the overhead of Nomad.

We run the benchmarks on 8 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 9 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 if there is no migration, which conforms to our earlier evaluation on XenIB [34]. Each migration causes 0.5 to 3 seconds increase of total execution time, depending on the benchmark. This overhead will be marginal for longer running applications.

We now 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.

6.3.1 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 to suspend communication and the time to free local resources. Suspending local

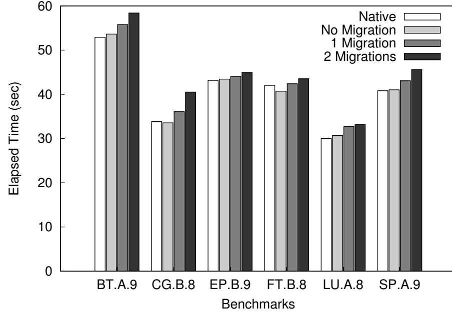

Figure 9. Impact of migration on NAS benchmarks (migrating one VM out of eight)

	Num. of Msg.	Avg. Size (KBytes)
BT	615	100.4
CG	6006	49.4
FT	48	3844.8
EP	5	0.024
LU	15763	3.8
SP	1214	74.9

Table 1. NAS Communication patterns: number of total messages to the most frequently communicated peer and the average message size to that peer

communication occurs in parallel with suspending remote communication. Suspension of remote communication typically takes a relatively 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 (*remote suspension time*). Please note that the results are based on multiple runs.

We profile each of these stages, as in Figure 10. We observe that the remote suspension time vary largely depending on the communication patterns. Table 1 characterizes the communication patterns observed on the MPI process hosted in the migrating VM. As we can see, CG has the longest *remote suspension time*, because it communicates frequently with relatively large messages, thus likely takes long time waiting for the outstanding communications. Following CG are LU and FT, which has large number of small messages and small number of large messages, respectively. EP has extremely low communication volume, and its remote suspension time is almost unobservable in the figure. With additional synchronization time, the migrating VM takes typically a few to tens of milliseconds waiting for communication suspension.

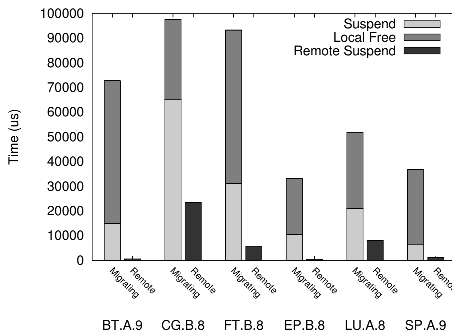

Figure 10. Suspend time running NAS benchmarks

Another major cost observed is the time to free local resources, which takes 22 to 57 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 57 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 22 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.

6.3.2 Resume Stage of Nomad

The cost at the resume stage 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.

As shown in Figure 11, 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 105ms for NAS-FT to 22ms for NAS-EP in our studies. This suggests pre-registration of memory pages can help reducing the migration cost too.

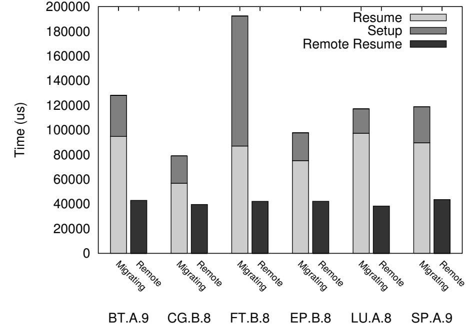

Figure 11. Resume time running NAS benchmarks

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. Also, total time spent in the suspend and resume stages is smaller than the overhead we observed in Figure 9. We believe that the extra overhead is the cost of live migration, e.g., the migrating OS is running in a shadow mode, with extra cost to dirty a page.

6.4 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 to distinct nodes.

Figure 12 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. Since the applications have longer execution time with four

processes, the impact of migration looks much smaller. Despite this, we observe larger variation of the results we collected. We believe that the skew between processes is the major cause for the variation. The skew can be mainly due to two reasons:

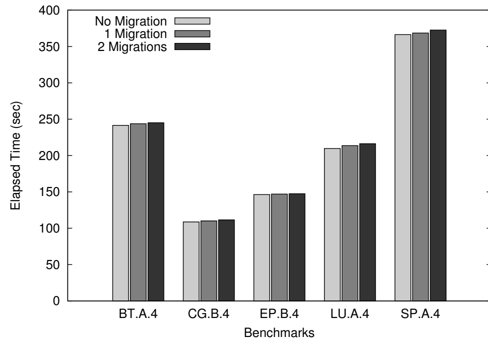

Figure 12. Impact of migration on NAS benchmarks (migrating all four VMs)

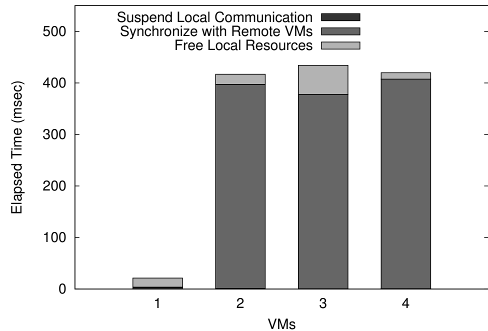

Figure 13. Suspend time running NAS-CG

- Though we start migration of all 4 VMs at the same time, Xen may take a varied amount of time to pre-copy 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 13 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. Discussion

In this paper we present a software design for migrating OS-bypass networks in virtual machines. Many of the design choices are due to the “unfriendly” nature of OS-bypass hardware for virtual machine environments. In this section, based on our experiences, we propose features for future OS-bypass devices that may provide better support for virtualization at hardware level.

First, one physical device should be able to associate with multiple addresses. In this case, there is no need to manually manage the globally unique IDs for the virtual devices in each domain. The next generation of InfiniBand devices, Mellanox ConnectX [15], has already added this capability of presenting multiple virtual end points to software. This satisfies the need for easier address management in virtual machine environment.

Second, for migration purposes, the hardware addresses should be able to migrate to another device dynamically. As in Ethernet, once the IP address is associated to another Ethernet card, packets will be routed correctly after an ARP packet is sent. However, to the best of our knowledge, there is no flexible support in today’s typical OS-bypass networks. This could be potentially realized by modifying the vendor-maintained routing software. We will further explore along this direction.

Finally, the major complexity of Nomad is due to the difficulty of getting the connection state during migration, which is transparent to software. Thus we have to handshake with remote side to suspend communication. We would like to propose `Extract` and `Inject` operations to device management interfaces. `Extract` saves the connection state of a specific connection (similar as sequence number, etc., for TCP) to a memory buffer, while `Inject` sets the connection to a specific state defined by the contents in a memory buffer. In this way, device on the new host can be synchronized to the exact state before the VM is migrated, which seamlessly handles the reliability during migration. Such support with careful software management of other resources, such as registered buffers, can provide transparent migration with less overhead.

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. After significant effort of academic research including [31, 28] and prototypes such as active messages [33], U-Net [32], FM [21], VMMC [4], and Arsenic [23], it has been adopted by the industry [8, 11] and has been incorporated into various commercial products [16, 24].

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. It 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 [9]. Instead, VMM-bypass makes use of intelligence in modern high speed network interfaces, targeting a relatively small range of devices which are used mostly in high-end systems.

Current virtual machine technologies have provided several solutions for migration of traditional network devices like Ethernet. The solution used in Xen [6] 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. An unsolicited ARP reply will be generated to advertise that the IP has been moved. However, the location dependent resources and the need for hardware level reliable service leads to 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 [20] 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 [5, 10, 37, 7]. 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 also plan to use high speed interconnects to accelerate the Nomad control and the Xen migration traffic because they currently go through Ethernet. Also, current implementation requires both peers involved in communication handshake during migration, thus requiring both of them running Nomad. We plan to explore solutions to achieve interoperability of Nomad and unmodified hosts running native operating systems, where the handshake will be impossible.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments, which help to improve the final version of this paper.

This research is supported in part by the following grants and equipment donations to the Ohio State University: Department of Energy's Grant #DE-FC02-06ER25749 and #DE-FC02-06ER25755; 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] Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures. In *IPDPS*, page 27.2, Washington, DC, USA, 2002. IEEE Computer Society.

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

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] InfiniBand Trade Association. InfiniBand Architecture Specification.

[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 and other RDMA Interconnects. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/index.html.

[19] OpenFabrics Alliance. http://www.openfabrics.org.

[20] 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.

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

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

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

[25] M. Rosenblum and T. Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *IEEE Computer*, May 2005.

[26] S. M. Hand. Self-Paging in the Nemesis Operating System. In *Proceedings 3rd OSDI*, pages 73–86, 1999.

[27] 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.

[28] A. Z. Spector. Performing remote operations efficiently on a local computer network. *Commun. ACM*, 25(4):246–260, 1982.

[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] C. A. Thekkath, H. M. Levy, and E. D. Lazowska. Separating Data and Control Transfer in Distributed Operating Systems. In Proceedings

of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 2–11, San Jose, California, 1994.

[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*, 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.