IBMon: Monitoring VMM-Bypass Capable InfiniBand Devices using Memory Introspection

Adit Ranadive, Ada Gavrilovska, Karsten Schwan

Center for Experimental Research in Computer Systems (CERCS)
Georgia Institute of Technology
Atlanta, Georgia, 30332
{adit262, ada, schwan}@cc.gatech.edu

Abstract

Active monitoring of virtual machine (VM) behaviors and their utilization of different resource types is critical for effective management of high end cluster machines, data centers, and cloud computing infrastructures. Unfortunately, for reasons of performance, certain types of existing and future devices support capabilities that provide VMs with direct access to device resources, thereby bypassing the virtual machine monitor (VMM). This presents significant challenges to the VMM due to its resulting inability to assess VM-device interactions.

This paper describes a monitoring utility, IBMon, which enables the asynchronous monitoring of virtualized InfiniBand devices – a sample VMM-bypass device heavily used in the HPC community. In the absence of adequate hardware supported monitoring information, IBMon uses dynamic memory introspection techniques to infer information regarding the VM-device interactions. Experimental results demonstrate that IBMon can asynchronously monitor VMM-bypass operations with acceptable accuracy, and negligible overheads, including for larger number of VMs, and for VMs with dynamic behavior patterns.

1. Introduction

Virtualization technologies like VMWare [23] and Xen [13] have now become a de facto standard in the IT industry. A main focus of these technologies has been to manage the virtual machines (VMs) run by the virtual machine monitor (VMM) or hypervisor, in order to ensure the appropriate allocation of platform resources to each VM, as well as to provide for isolation across multiple VMs. Such actions are necessitated by the fact that static resource allocation policies are not typically suitable, as they imply a priori knowledge about the VM's behavior – which may be impossible to determine in advance, and/or worst-case analysis of its resource requirements – which may lead to significant reductions in the attained resource utilization and platform capacity.

Managing virtualized platforms relies on mechanisms that dynamically adjust resource allocations, based on workload characteristics and operating conditions, as well as on current VM behavior and resource requirements [6, 20, 24]. Such techniques are particularly relevant to distributed clouds and virtualized data centers, to more effectively utilize aggregate platform resource and reduce operating costs, including for power. The latter is also one reason why virtualization is becoming important for high end systems like those used by HPC applications.

A basic requirement for effective resource management is the availability of runtime information regarding VMs' utilization of platform resources, including CPU, memory, and network and disk devices. Using this information, the VMM infers the behavior and requirements of VMs and makes appropriate adjustments in their resource allocations (i.e., it may increase or decrease the amount of resources allocated to a VM) [6, 20], while still ensuring isolation and baseline performance levels. Monitoring information is collected continuously in the hypervisor and/or its management domain (i.e., dom0 in Xen). For instance, for scheduling CPU resources, the VMM scheduler relies on monitoring the time a VM has spent executing or idling in order to adjust the corresponding scheduling priorities or credits [25], whereas for memory it monitors the VM's memory footprint, and if necessary, makes appropriate adjustments, e.g., by using the 'balloon-driver' in the case of the Xen VMM. The same is true for most devices, for which the VM's device accesses are 'trapped' and redirected to a centralized device manage-

3rd Workshop on System-level Virtualization for High Performance Computing (HPCVIRT'09) March 31, 2009, Nuremberg, Germany.

ment domain, i.e., following the 'split device driver model' in Xen [13, 19]. All information regarding the VMs' utilization of a particular device therefore, is centrally gathered and can be easily maintained and used to enforce limits and/or orderings on device accesses. Furthermore, management actions may span multiple types of resources, such as when monitoring information regarding the network buffer queues is used to adjust the CPU credits for a VM, so as to eliminate buffer overflow and ensure timely response [6, 20]. Stated more technically, the management of the virtualized platform relies on the ability of the virtualization layer (e.g., ESX server or the Xen hypervior plus dom0) to have the ability to monitor and control the utilization of each of the platform resources on behalf of a VM.

Unfortunately, some IO devices do not follow the 'split driver' model, and cannot be easily monitored via the same approach as above. Some such devices exhibit 'self-virtualization' capabilities [15], i.e., the management logic that controls which VM will be serviced by the device is offloaded onto the device itself. When used in the high performance domain, in order to meet the high bandwidth/low latency messaging requirements of HPC applications, these devices also use their RDMA capabilities to read/write data directly from/to guest VMs' address spaces. Specifically, the control path invoked during a VM's setup and device initialization is still routed through a management domain such as dom0, but subsequent device accesses are performed directly, thereby 'bypassing' the VMM. Devices virtualized in this manner include InfiniBand HCAs [4, 8], and they will also include next generation high performance NICs such as those used for high end virtualized IO [15, 16] and as facilitated by next generation device and board interconnection technology [18, 21].

The challenge addressed by our research is the acquisition of monitoring information for devices that use VMM bypass. The outcome is a novel monitoring utility for InfiniBand adapters, termed *IBMon*. IBMon's software-based monitoring methods can provide the information they collect to the management methods that require it. An example is the provision of information about communication patterns that is then used to adjust a VM's CPU or memory resources, or to trigger VM migration. IBMon can also supplement an IB device's monitoring logic, or it can compensate for issues that arise if device-level monitoring is not performed at the levels of granularity required by management. Most importantly, by using VM introspection techniques [3, 12], IBMon can acquire monitoring information for bypass IB devices, at small overheads and with minimal perturbation. Specifically, for such devices, IBMon can detect ongoing VM-device interactions and their properties (i.e., data sent/received, bandwidth used, or other properties of data transfers). Such determination of the VM's usage of its share of IB resources is critical to proper online VM management. Using introspection, of course, also implies that IBMon's implementation uses knowledge about internals of the IB stack and buffer layout in the VM's address space. Stated differently, due to the absence of hardware-supported device monitoring information, IBMon uses gray-box monitoring approaches to assess the VM-device interactions and their properties.

In the remainder of the paper, we first describe IBMon and evaluate its utility and costs. In the following sections, we then provide brief summaries of the techniques on which IBMon depends and next present its design and implementation. Section 3.3 describes the bandwidth estimation mechanism currently used in IBMon. We next show experimental results to show that with IBMon, we can asynchronously monitor VMM-bypass operations with acceptable accuracy and negligible overheads, including as we increase the number of VMs.

The current implementation of IBMon enable asynchronous monitoring of the VM's bandwidth utilization, however, as noted earlier, the same approach can be used to monitor other aspects of VMs' usage of IB resources. While IBMon is implemented for InfiniBand devices virtualized with the Xen hypervisor, its basic concepts and methods generalize to other devices and virtualization infrastructures. Of particular importance are the novel ways in which the virtualization layer asynchronously monitors VM-device interactions that bypass the VMM (i.e., for high performance), and the insights provided regarding requirements for hardware supported monitoring functionality.

2. Background

The design of the IBMon tool for the Xen hypervisor is specific to (1) the manner in which IB devices export their interfaces to applications (and VMs) and the memory management mechanisms they support, (2) the manner in which these devices are virtualized in Xen environments, and (3) the memory introspection techniques being used. This section briefly describes the aspects of (1)-(3) relevant to IBMon's design and implementation.

InfiniBand Memory Management. The communication model used in IB is based on the concept of queue pairs (Send and Receive Queues). A request to send/receive data is formatted into a Work Request (WR) which is posted to the queues. There are also Completion Queues (CQs) in which Completion Queue Entries (CQEs) containing information for completed requests are stored. Whenever a descriptor is posted to the queue pair (QP), certain bits called 'Doorbells' are set in User Access Region (UARs) buffers. The UARs are 4KB I/O pages mapped into the process' address space. When a request is issued, the doorbell is "rung" in the process' UAR. The HCA will then pick up these requests and process them.

The HCA maintains a table called the Translation and Protection Table (TPT). This table contains the mappings from

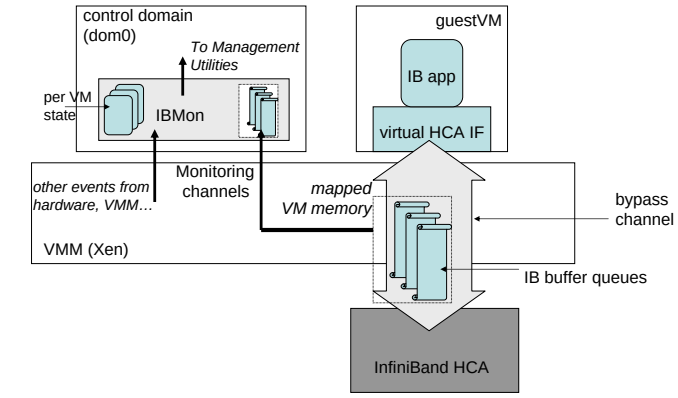


Figure 1. IBMon Utility

physical to virtual addresses of the buffers and queues described above. For InfiniBand, the buffers need to be registered with the HCA, and these registry entries are stored in the TPT. Also, these buffers must be kept pinned in the memory during data transfer so that the HCA can DMA the data directly in and out of host memory. Each entry in the TPT is indexed with a key that is generated during the registration process.

Virtualized InfiniBand Driver Model. When virtualized in Xen, the device driver model is based on the 'split device driver' model as described in [13]. It consists of a Backend and Frontend Driver. The frontend driver resides in the guest OS kernel while the backend driver resides in the dom0 kernel. In the case of regular devices, all requests to access the device from the guest VM must travel from the frontend driver to the backend driver. Therefore, both the data and control path instructions are sent to Dom0.

For InfiniBand, the split-driver model is used slight differently. The control path operations from the guest VM, like memory registration and queue pair creation, must all go through the backend driver. Fast path operations like polling the CQ and QP accesses are optimized using VMM-bypass [8]. The data path is highly optimized since the HCA can write DMA data directly to the target buffer. As a result, IB platforms can be virtualized with negligible impact on the attainable latency and bandwidth.

Virtual Memory Introspection. With the Xen VMM, pages of one VM can be read by another VM simply by mapping the page into the target VM memory. This concept was first introduced by Garfinkel and Rosenblum [3]. Xen contains the XenControl library that allows accessing another VM's memory. Using the function *xc_map_foreign_range*, the target memory is mapped into the current application's virtual memory. We use this mechanism to map the physical pages which correspond to the IB buffers into the monitoring utility.

3. IBMon Design and Implementation

Next we describe in greater detail the components of the IBMon utility and its mechanisms.

3.1 Design

Figure 1 illustrates the design of the IBMon utility. It is deployed in the VMM's control domain, i.e., dom0 in Xen, and its goal is to asynchronously monitor the per VM utilization of the resources of the InfiniBand fabric, i.e., bandwidth utilization in our current realization.

As described above, the IB HCA does have the capability to export virtual interfaces, i.e., queue pairs and completion queues, directly to guest VMs. However, the hardware only exports counters about the aggregate usage of the communication resources used by all VMs, which is not adequate for the per-VM management decisions we would like to support with IBMon. Therefore, we rely on knowledge about the layout of the memory region used by each VM for its virtual HCA interface to map and monitor the queues residing across those memory pages. We then apply memory introspection techniques to detect changes in the page contents, and based on that make conclusions regarding the VMs use of IB bandwidth.

The monitoring process is triggered periodically, with a dynamically configurable frequency. The exact details of how the current implementation uses the accessed data to perform bandwidth estimations is described later in Section 3.3. Dynamic changes in monitoring frequency are necessary to adjust the sampling rate to the frequency at which events being monitored are actually occuring, so as to avoid degradation in the quality of the monitoring information.

Note that the in the case of IB, the HCAs do not maintain per work queue bandwidth or timing information in hardware. This, coupled with the asynchronous nature of both the monitoring operations as well as the VM's IO operations themselves, requires that we adopt the dynamic approach described above. The presence of such hardware supported information in next generation IB or other multi-queue, VMM-bypass capable devices will simplify the processing required by IBMon or similar monitoring utilities, but it will not eliminate the need for such information to be asynchronously accessed and then integrated with the platform level management mechanisms.

In addition, while the presence of adequate hardware support may obviate the need for the bandwidth estimation mechanisms integrated in IBMon, there will be other types of information regarding the VMs' device accesses which will still require that we integrate memory introspection mechanisms with the monitoring tool. Examples of such information include certain patterns in communication behavior, identification of the peers with which the VM communicates with (e.g., to determine that it should be migrated to a particular

node), IO operations involving certain memory region (e.g., to indicate anomalous behavior or security issues), etc.

Finally, the figure illustrates that IBMon can interface with other management or monitoring components in the system. This feature is part of our future work, and the intent is to leverage the tool to support better resource allocation (i.e., scheduling) policies in the VMM (e.g., by using the IBMon output to make adjustments in the credits allocated to VMs by the VMM scheduler), or to couple the monitoring information with large-scale 'cloud' management utilities.

3.2 Implementation

IBMon is implemented as a monitoring application which runs inside dom0 in Xen. It uses the XenControl library (libxc) to map the part of the VMs' address space used by the IB buffers. In the InfiniBand driver model, upon the front-end driver's request, the back-end driver allocates the buffers and then registers them with the HCA with the help of the dom0 HCA driver. IBMon augments the back-end operations to establish a mapping between the physical memory addresses and the types of buffers they are used for (CQ, QP, etc.), for each VM. The table is indexed by the VM's identifier and is accessed through the '/proc' interface provided by the back-end.

The IBMon utility reads the entries of the table when the guest VM starts running an IB application. It detects any new entries in the backend driver table. For each of the entries, the utility uses the xc_map_foreign_range function of libxc to map that IB buffer page to its address space. Once the page is mapped, the VM's usage of the IB device can be monitored. Note that these pages are mapped as read-only to IBMon's address space.

3.3 Estimating VMs' Bandwidth Utilization with IBMon

InfiniBand uses an asynchronous (polling-based) model to infer completion of communication. Hence, any IB-based application must poll the CQ to check whether a request has completed or not. These CQ buffers are also mapped to IBMon's memory space, which itself can then check and determine whether any work requests on behalf of a specific VM have completed. The CQ buffer contains several fields which can be used for bandwidth estimation. First is the number of completion entries (CQE) which indicates how many completion events have occurred. Second is the byte length (byte_len) field which indicates how many bytes were transferred for each of the completion event. These entries are written to directly by the HCA.

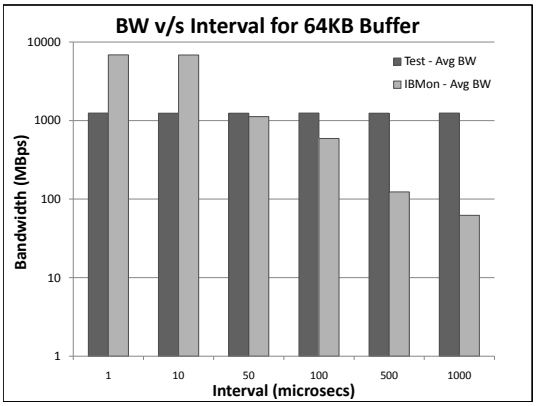
IBMon periodically inspects the newly posted CQEs, gathers the amounts of bytes transmitted during the latest time interval, and uses that to estimate the bandwidth. There are several challenges in this approach. First, the CQEs do not contain any timing information. Therefore, if IBMon is configured with a short monitoring interval, IBMon detects CQE which indicates that a work request for a read of a large buffer just completed, using a simple Bytes/time formula it will compute bandwidths which far exceed the fabric limits. Since we do not know when that read operation was initiated, we cannot just adopt a model where we compute the bandwidth by averaging across multiple monitoring intervals. We could extend the implementation of IBMon to require monitoring of the VM's original read request (by forcing traps whenever the read call memory page is accessed), or we can monitor and timestamp all work requests, and then match the CQE with the corresponding WQE to get a better estimate of the bandwidth. However, both of these alternatives still include asynchronous monitoring, and hence would not guarantee accuracy, and furthermore are computationally significantly more complex, or, in the case of the first one, even require inserting significant fast path overheads.

Therefore IBMon uses an approach that dynamically tunes the monitoring interval to the buffer size used by the VM. The interval is reduced in the event of small buffer sizes and increased when the VM reads/writes large buffers. Determining that the interval should be increased whenever IBMon determines bandwidth values above the IB limits is intuitive, however it is significantly more challenging to determine the adequate rate at which is should be modified, and when it should be decreased again. Assigning significantly large monitoring intervals, at the data rates supported by IB, will result in significant margins of error of the monitoring process, and will generate excessive amount of CQ entries to be processed in each monitoring iteration. Therefore we rely on the buffer size value, i.e., the *monitoring granularity*, to determine how to adjust the monitoring interval. For very small monitoring intervals the extra memory access can be avoided by using the fabric bandwidth limit as an indication that the interval should be increased. While the current bandwidth estimation mechanism used by IBMon uses this approach, we will further evaluate and compare other alternatives.

4. Experimental Results

Next we present the results from the experimental evaluation of IBMon, aimed at justifying the suitability of the design to asynchronously monitor VM's usage of VMM-bypass capable devices such as IB HCAs and the feasibility to attain estimates of such usage with reasonable accuracy.

Testbed. The measurements are gathered on a testbed consisting of 2 Dell 1950 PowerEdge servers, each with 2 Quad-core 64-bit Xeon processors at 1.86 GHz. The servers have Mellanox MT25208 HCAs, operating in the 23208 Tavor compatibility mode, connected through a Xsigo VP780 I/O Director switch. Each server is running the RHEL 4 Update



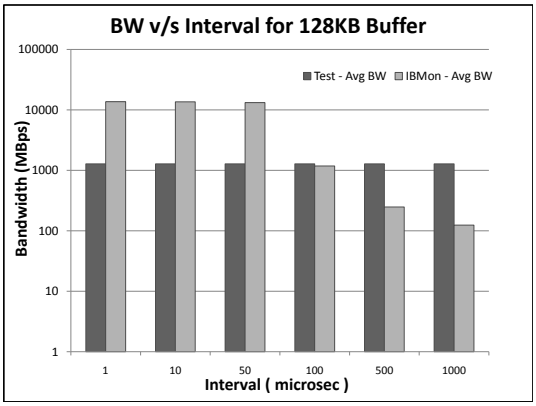


Figure 2. Buffer size vs Frequency

5 OS (paravirtualized 2.6.18 kernel) in dom0 with the Xen 3.1 hypervisor.

The guest kernels are paravirtualized running the RHEL 4 Update 5 OS. Each guest is allocated 256 MB of RAM. For running InfiniBand applications within the guests, OFED (Open Fabrics Enterprise Distribution) 1.1 [11] is modified to be able to use the virtualized IB driver. Both Xen and the OFED drivers have already been modified by us, so as to be able to efficiently support multiple VMs on IB platforms [17]. The workloads are derived from existing RDMA benchmarks part of the OFED distribution.

Monitoring frequency. The first set of measurements illustrates the relationship between the buffer sizes used by the VM's operation and the monitoring interval for two different buffer sizes. The measurements are gathered by monitoring a single VM and show the bandwidth reported by IBMon and the bandwidth reported by RDMA_Read benchmark executing in the VM. The benchmark performs 20000 iterations of an RDMA read operation for a given buffer size. We vary the monitoring interval from 1 to 1000μs – larger monitoring intervals resulted in excessive amounts of CQ entries.

As seen in the graphs in Figure 2, for very small intervals relative to the buffer size, and therefore the byte count used by IBMon to compute the bandwidth estimate, IBMon reports the maximum bandwidth supported by the IB fabric. This is also used by IBMon to trigger an increase in the monitoring interval. For monitoring intervals which are too large relative to the fabric bandwidth and therefore the buffer size transmitted with each read request, IBMon significantly under performs – at each iteration multiple CQEs are associate with the same, much later timestamp used to then estimate the bandwidth for the total number of bytes, resulting in up to an order of magnitude lower estimates for the largest time interval shown in Figure 2. More importantly, however, the graphs show that in both cases there are interval values for which IBMon performs very well, and only slightly underestimates the actual bandwidth usage by the VM. Using similar data for other buffer size we can either experimentally determine the adequate interval for a range of buffer sizes, which can then be used by IBMon, or compute its bounds based on the fabric bandwidth limits.

Feasibility. Next, for different monitoring granularities, i.e., different buffer sizes and the corresponding interval values, we evaluate the accuracy with which IBMon is able to estimate the VM's utilization of the device resources. The graph in Figure 3 indicates that at fine granularity, IBMon does face some challenges, and its accuracy is only within 27% of the actual value. However for larger buffer sizes (which is often the case for HPC applications), and therefore longer monitoring time intervals, IBMon is able to estimate the bandwidth quite accurately, within up to 5% for some of the data points in the graph.

Overheads. For each of the monitoring granularities used above, we next measure the overheads of using IBMon. IBMon is deployed as part of dom0, and as such it is executing on a core separate from the guest VM. First, it is likely that in virtualized manycore environments such management functionality will typically be deployed on designated cores. Second, a lot of the processing overhead associated with the current implementation of IBMon is due to the fact that it does have to perform additional computation to estimate the bandwidth, whereas with appropriate hardware support that overhead will be significantly reduced. One overhead which will remain is the fact that IBMon does rely on frequent memory accesses for the monitoring operations, and as such it will impact the application performance due to memory contention. Since the pages are mapped as read-only into IBMon's address space the issue of data consistency will not arise.

In order to evaluate this impact we compare the execution of the same benchmark with and without IBMon. The results shown in Figure 4 indicate any increase in the VMs execution time is virtually negligible, and that therefore IBMon's overheads are acceptable.

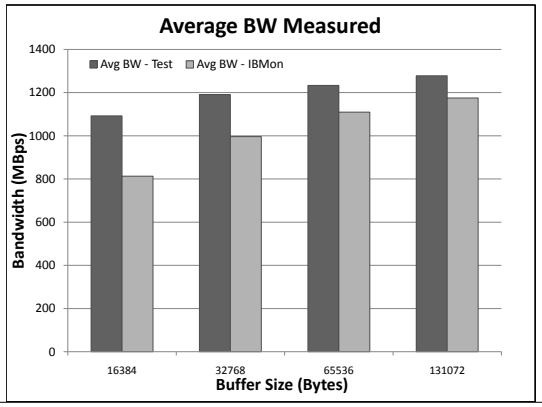


Figure 3. Accuracy of IBMon for different monitoring granularities

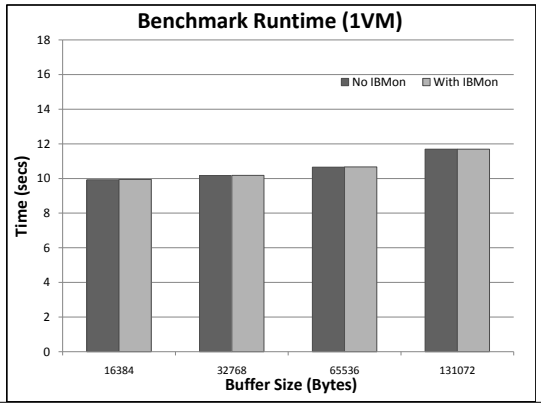


Figure 4. IBMon overheads for different monitoring granularities

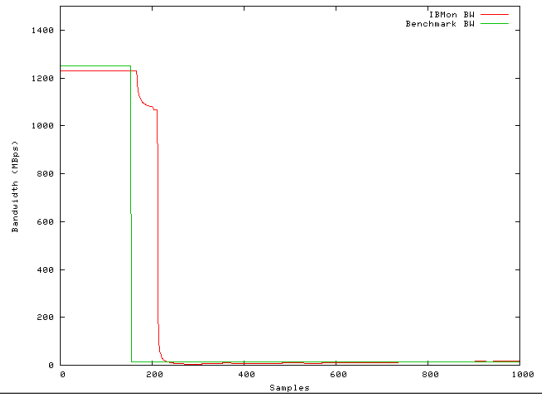


Figure 5. Ability to respond to dynamic changes in the VM behavior

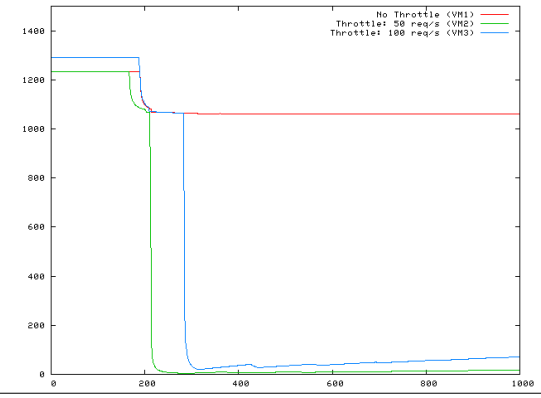


Figure 6. Ability to monitor multiple VMs with different behaviors

Ability to respond to changes in VM behavior. By dynamically monitoring the frequency of the completion events and the buffer sizes used in the RDMA operations, IBMon attempts to determine the monitoring granularity and to adjust the rate at which it samples the VMs' buffer queues. In order to evaluate the feasibility of the approach we modified the RDMA benchmark to throttle the rate at which it issues the read requests and thereby lowered its bandwidth utilization. The graphs in Figure 5 compare the bandwidth measured by the benchmark vs by IBMon and we observe that IBMon is able to detect the changes in the VM behavior reasonably quickly.

Scalability with multiple VMs. The above measurements focused on evaluating IBMon's ability to monitor a single VM. Next we conduct several experiments involving multiple VMs in order to assess the feasibility of concurrently monitoring multiple buffer queues and to track monitoring information corresponding to multiple VMs.

First, the graphs in Figure 7 show IBMon's accuracy in concurrently monitoring multiple VMs. We observe that although IBMon now is responsible for tracking multiple queues and maintaining information for multiple VMs, it is still capable of estimating the behavior of each of the VMs with acceptable accuracy levels. The current estimates are within 15% of the actual value, we do believe that there are significant opportunities in this initial implementation of IBMon to further improve these values.

Next, the measurements in Figure 6 show that in the multi-VM case IBMon is able to respond to dynamic changes in individual VMs. For these measurements we use the modified RDMA_Read benchmark and for VMs 2 and 3 we throttle the request rate to a lower value. The measurements do show that IBMon does not respond to the behavior changes very rapidly, but we believe that evaluating methods to more aggressively tune the frequency interval (e.g., such as those used in AIMD in TCP) will result in additional improvements in its responsiveness.

Finally, the results presented in Figure 8 demonstrate the acceptable overheads of IBMon as it concurrently monitors multiple VMs, again by comparing the benchmarks execution time with and without IBMon. While, as expected, the overheads do increase as we increase the number of VMs, and queues IBMon has to monitor, their impact on the VMs execution is still on average within 5% of the original performance levels.

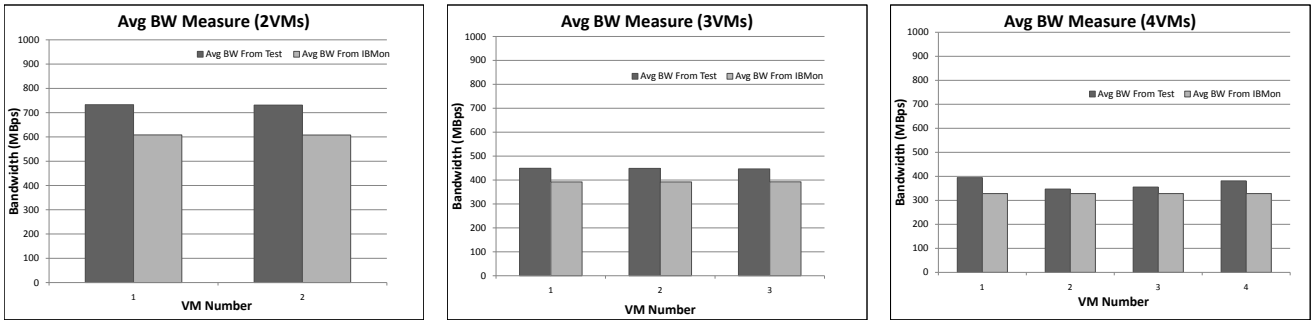


Figure 7. Accuracy of IBMon when monitoring multiple VMs

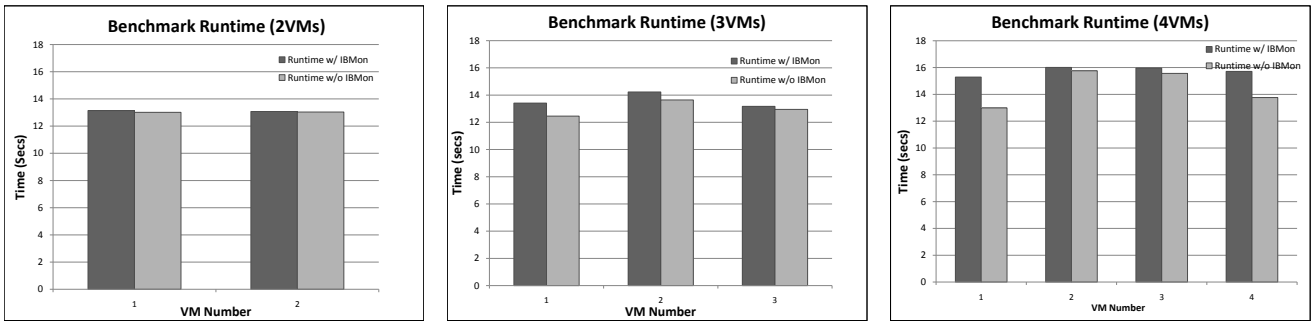


Figure 8. Overheads of IBMon when monitoring multiple VMs

5. Related Work

Several research efforts by our own group as well as others focus on dynamic monitoring and analysis on the behavior of applications deployed on a single system or across distributed nodes [1, 2, 6, 10, 20], in order to better support management functionalities such as node-level power management, CPU scheduling and memory allocation [6, 10] or VM deployment and migration in cluster settings [2, 7, 24]. A key distinction in IBMon is that it enables monitoring of devices and VM interactions with such devices which occur 'outside' of the virtualization layer. To the best of our knowledge, there hasn't been prior work focusing on dynamic monitoring of the VMs or applications' use of devices such as the IB HCAs, when accessed in a manner which bypasses the operating system or the hypervisor layer.

Our own prior work presented in [6] does rely on monitoring of InfiniBand communication channels in virtualized environments, however in this setting the monitoring information is provided externally, by a fabric-level monitor residing in the fabric routing component. IBMon differs from this in two ways. First, by not requiring interactions with external entities, and by not requiring external entities to maintain fine-grain information regarding the InfiniBand resource utilization of every one of the VMs, our approach is more suitable for building scalable monitoring infrastructures, and for supporting low-latency, responsive management solutions. Second, an external monitor will typically be more statically configured, to monitor only certain types of events and gather only certain types of statistics. As a software entity part of the individual nodes' virtualization layer, IBMon can more easily be extended to support a broad range of application-specific management policies. Finally, an external entity cannot extract any information about the VM which is not part of the actual data exchange. IBMon integrates VM introspection mechanisms, which can be used to make various observations regarding a VM's behavior.

There has been much research in memory introspection for VMs in virtualized environments. VMSafe [22] is a security framework provided by VMWare for their industry-standard hypervisor, ESX Server, which includes sets of APIs to perform introspection of different resources of the system. [3] discusses a technique called Virtual Machine Introspection (VMI) used to construct an Intrusion Detection System (IDS). VMI monitors resources like CPU, memory, and network used by the virtual machine, and the IDS detects any attacks originating from the VM. Similarly, Payne et al. [12] develop a framework for virtual memory introspection and virtual disk monitoring for the Xen hypervisor. Other approaches to memory introspection include the use of additional hardware. For instance, in [5] a specialized co-processor is used to monitor the memory of running kernels for any malicious changes. Finally, our work targets InfiniBand devices, as a sample device with VMM-bypass capabilities. Devices with similar capabilities have been extensively used in other high performance interconnect technolo-

gies [9, 14], with more recent developments targeted at commodity networks [18]. Furthermore, recent research on virtualization solutions for high performance networking and IO, further advocate similar solution [16], and future hardware platforms and device interconnection technologies will offer capabilities to enable direct VM-device interactions for an even broader range of devices. As virtualized platforms will continue to require even more sophisticated management methods, the importance to asynchronously monitor all of these types of devices will continue to grow.

6. Conclusion

The challenge addressed by our research is the acquisition of monitoring information for devices that use VMM bypass. The outcome is a novel monitoring utility for InfiniBand adapters, termed IBMon, which enables asynchronous monitoring of virtualized InfiniBand devices – an example of VMM-bypass devices heavily used in the HPC community. In the absence of adequate hardware-supported monitoring information, IBMon relies on VM introspection techniques to detect ongoing VM-device interactions and their properties (i.e., data sent/received, bandwidth utilized, or other properties of the data transfers). Such information can then be used for platform management tasks, such as to adjust a VM's CPU or memory resources, or to trigger VM migrations. Experimental results demonstrate that IBMon can asynchronously monitor VMM-bypass operations with acceptable accuracy, and negligible overheads, including for larger number of VMs, and for VMs with dynamic behavior patterns.

While IBMon is implemented for InfiniBand devices virtualized with the Xen hypervisor, its basic concepts and methods generalize to other devices and virtualization infrastructures. Of particular importance are the novel ways in which the virtualization layer asynchronously monitors VM-device interactions that bypass the VMM (i.e., for high performance), and the insights provided regarding requirements for hardware supported monitoring functionality.

References

[1] S. Agarwala and K. Schwan. SysProf: Online Distributed Behavior Diagnosis through Fine-grain System Monitoring. In *ICDCS*, 2006.

[2] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering and retrieving system history. In *SOSP*, 2005.

[3] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the 2003 Network and Distributed System Symposium*, 2003.

[4] W. Huang, J. Liu, and D. Panda. A Case for High Performance Computing with Virtual Machines. In *ICS*, 2006.

[5] N. L. P. Jr., T. Fraser, J. Molina, and W. A. Arbaugh. Copilot -a coprocessor-based kernel runtime integrity monitor. 2004.

[6] M. Kesavan, A. Ranadive, A. Gavrilovska, and K. Schwan. Active CoordinaTion (ACT) - Towards Effectively Managing Virtualized Multicore Clouds. In *Cluster 2008*, 2008.

[7] S. Kumar, V. Talwar, P. Ranganathan, R. Nathuji, and K. Schwan. M-Channels and M-Brokers: Coordinated Management in Virtualized Systems. In *Workshop on Managed Multi-Core Systems, in conjunction with HPDC'08*, 2008.

[8] J. Liu, W. Huang, B. Abali, and D. K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. In *ATC*, 2006.

[9] Myricom, Inc. Myrinet. www.myri.com.

[10] R. Nathuji and K. Schwan. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In *SOSP*, 2007.

[11] OpenFabrics Software Stack - OFED 1.1. www.openfabrics.org/.

[12] B. D. Payne, M. D. P. de Carbone, and W. Lee. Secure and Flexible Monitoring of Virtual Machines. In *Computer Security Applications Conference*, 2007.

[13] I. Pratt, K. Fraser, S. Hand, C. Limpach, A. Warfield, D. Magenheimer, J. Nakajima, and A. Mallick. Xen 3.0 and the Art of Virtualization. In *Ottawa Linux Symposium*, 2005.

[14] Quadrics, Ltd. QsNet. www.quadrics.com.

[15] H. Raj and K. Schwan. High Performance and Scalable I/O Virtualization via Self-Virtualized Devices. In *HPDC*, 2007.

[16] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner. Achieving 10Gbps using safe and transparent network interface virtualization. In *VEE*, 2009.

[17] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan. Performance Implications of Virtualizing Multicore Cluster Machines. In *Workshop on HPC System Virtualization, in conjunction with Eurosys'08*, 2008.

[18] RDMA-enabled NIC. www.rdmaconsortium.org.

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

[20] B. Urgaonkar and P. Shenoy. Sharc: Managing CPU and Network Bandwidth in Shared Clusters. In *IPDPS*, 2004.

[21] Virtual Machine Device Queues. www.intel.com/network-/connectivity/vtc_vmdq.html.

[22] VMSafe. www.vmware.com/technology/security/vmsafe.html.

[23] The VMWare ESX Server. http://www.vmware.com/products/esx/.

[24] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. In *NSDI*, 2007.

[25] Xen Credit Scheduler. wiki.xensource.com/xenwiki/CreditScheduler.