ReNIC: Architectural Extension to SR-IOV I/O Virtualization for Efficient Replication

YAOZU DONG, Intel Asia-Pacific Research and Development Ltd., P. R. China
YU CHEN and ZHENHAO PAN, University of Tsinghua, P. R. China
JINQUAN DAI and YUNHONG JIANG, Intel Asia-Pacific Research and Development Ltd., P. R. China

Virtualization is gaining popularity in cloud computing and has become the key enabling technology in cloud infrastructure. By replicating the virtual server state to multiple independent platforms, virtualization improves the reliability and availability of cloud systems. Unfortunately, existing Virtual Machine (VM) replication solutions were designed only for software virtualized I/O, which suffers from large performance and scalability overheads. Although hardware-assisted I/O virtualization (such as SR-IOV) can achieve close to native performance and very good scalability, they cannot be properly replicated across different physical machines due to architectural limitations (such as lack of efficient device state read/write, buffering outbound packets, etc.).

In this paper, we address those architectural limitations, by proposing ReNIC, an architectural extension to SR-IOV I/O virtualization for efficient I/O replications. We have extended Xen hypervisor and the Remus rapid checkpoint solution to support this new architectural extension. We developed a system simulator on multi-core systems to extensively evaluate ReNIC. The experimental results demonstrate that ReNIC achieves up to 54% CPU usage reduction, compared to software based I/O virtualization at runtime, and up to 16.2% performance advantage over software based I/O virtualization in rapid checkpoint. During migration, ReNIC reduces service shutdown time by about 50%, compared to device emulation and paravirtualized I/O, and over 71% compared to teaming driver.

Categories and Subject Descriptors: D.4.5 [**Operating Systems**]: Reliability; D.4.4 [**Operating Systems**]: Communications, Management—*Network communication*; D.4.8 [**Operating Systems**]: Performance

General Terms: Measurement, Performance, Design, Reliability, Experimentation

Additional Key Words and Phrases: Virtualization, replication, high availability, VM migration, single root I/O virtualization, SR-IOV

ACM Reference Format:
Dong, Y., Chen, Y., Pan, Z., Dai, J., and Jiang, Y. 2012. ReNIC: Architectural extension to SR-IOV I/O Virtualization for efficient replication. ACM Trans. Architec. Code Optim. 8, 4, Article 40 (January 2012), 22 pages.
DOI = 10.1145/2086696.2086719 http://doi.acm.org/10.1145/2086696.2086719

1. INTRODUCTION

Virtualization is the key enabling technology for cloud computing and has enabled a lot of new usage models, such as pay-as-you-use cloud service with service level agreement

This work is supported by the National Natural Science Foundation of China (Grant No. 61170050).
Authors' addresses: Y. Dong, J. Dai, and Y. Jiang, Intel Asia-Pacific Research & Development Ltd., Shanghai, P.R. China; Y. Chen and Z. Pan, Computer Science Department, University of Tsinghua, Beijing, P.R. China; email: dongyaozu@gmail.com.

DOI 10.1145/2086696.2086719 http://doi.acm.org/10.1145/2086696.2086719

(SLA), for example, service availability and responsiveness [Amazon EC2 2008]. In particular, virtual machine (VM) replication (such as live migration [Clark et al. 2007; Nelson et al. 2005] and rapid checkpoint [Cully et al. 2008]) improves the resource utilization reliability, and availability of cloud systems by replicating the virtual server state from one physical platform to another (to survive from hardware failures, such as fail-stop failures). For instance, live migration achieves better resource utilization by dynamically load balancing virtual servers among different physical platforms, and service availability by seamlessly moving a running VM from the (to-be-faulty) source machine to the (healthy) target machine, within the SLA. Rapid checkpoint synchronizes the VM state from the primary VM (running on the primary host) to the backup VM (running on the backup host) with very high frequencies (i.e., up to 40 times/sec in Remus [Cully et al. 2008]) and buffers outbound packets during each epoch until a successful checkpoint. Consequently, the service is able to seamlessly fail-over to the backup VM when the primary host suffers from fail-stop failures.

VM replication requires that the entire VM (CPU, memory, and I/O) state be transmitted over the network and replicated at the target or backup host. Although the replication of CPU and memory virtualization has been well studied in the literature [Clark et al. 2007; Hines et al. 2009; Nelson et al. 2005], I/O virtualization imposes several new challenges to VM replication. Software based I/O virtualizations (such as device emulation and paravirtualized I/O [Fraser et al. 2004]) can be properly replicated, but they suffer from large performance and scalability overhead (due to excessive hypervisor interventions in the bulk data processing path). To address these performance issues, we propose hardware-assisted I/O virtualizations (such as directed I/O and Single Root I/O Virtualization, named as SR-IOV). For instance, an SR-IOV-capable device has one or more Physical Functions (PFs), and each PF can create multiple "lightweight" instances of PCI function entities, known as Virtual Functions (VFs), each of which can be assigned to a guest for direct accesses [SR-IOV 2008]. While hardware-assisted I/O virtualizations can achieve close to native performance and very good scalability [Dong et al. 2010], they cannot be properly replicated across different physical machines due to several architectural limitations such as lack of efficient device register read/writes, etc.

For efficient VM replication, an I/O virtualization solution must have both great performance and scalability at runtime, and also have minimal impacts on SLA (such as service shutdown time) at replication time. To properly and efficiently replicate VM, hardware-assisted I/O virtualization should meet three requirements: (1) the state (such as registers) of the device needs to be efficiently read and written to support device state replication with very high frequencies; (2) the device needs to efficiently buffer the output packets until they can be released after successful checkpoints; (3) the dirty memory written by the device Direct Memory Access (DMA) needs to be efficient and tracked for lazy memory state transmission.

In this paper, we propose a replicable NIC (ReNIC), an architectural extension to SR-IOV network virtualization for efficient I/O replication support. We first extend an SR-IOV-capable NIC to implement an additional I/O clone mode/interface, besides the normal working mode. The clone mode and interface help easily clone VF state (i.e., reading and writing the entire VF state). We also enhance the normal working mode to efficiently buffer outbound packets in the device. To replicate the I/O state, memory pages modified by the device DMA operations must also be efficiently tracked. We thus modestly extend the I/O Memory Management Unit (I/O MMU) page table architecture to efficiently track dirty pages, by introducing an additional dirty bit. To support ReNIC for efficient VM replication, we extend the PF driver in SR-IOV for VF state clone and outbound packets buffering service, as well as the replication manager

in the hypervisor for I/O state clone and DMA dirty page migration. Although our implementation is done for Xen [Barham et al. 2003], it is generic in nature and can be applied to other hypervisors and rapid checkpoint solutions.

To evaluate our proposed architecture, we developed a system simulator that uses underlying multiple cores to provide close to hardware performance of a ReNIC device. We conducted extensive experiments to evaluate our proposed architecture ReNIC. The experimental results demonstrated that ReNIC can achieve up to 54% CPU usage reduction, compared to software based I/O virtualization at runtime, and up to 16.2% performance improvement over software based I/O virtualization in rapid checkpoint. During migration, ReNIC can achieve about 50% reduction of service shutdown time, compared to device emulation and paravirtualized I/O (from 1.4 second to 0.7 second), and over 71% reduction compared to teaming driver (from 3.5 seconds to 1.0 second), significantly reducing the impact on SLAs (service level agreements).

The rest of the paper is organized as follows. In Section 2, we introduce the I/O replication challenges and discuss potential solutions. We describe the ReNIC architectural extension on top of SR-IOV-capable NIC and the implementation of the software simulator for ReNIC in Section 3 and 4, respectively. We then present the experimental results and discuss the related work in Sections 5 and 6, respectively. Finally, we conclude the paper in Section 7.

2. I/O REPLICATION CHALLENGES

In this section, we first provide an overview of the I/O virtualization architecture in Xen in subsection 2.1, and then describe the replication challenges of the current I/O virtualization solutions in subsection 2.2. In subsection 2.3, we discuss potential approaches to address the replication challenges.

2.1. I/O Virtualization Architecture in Xen

Xen supports both paravirtualized virtual machine (PVM) and hardware virtual machine (HVM) [Barham et al. 2003]. PVM improves virtualization performance with moderate modifications to OS codes, which, however, is difficult to apply to proprietary OSs. HVM uses hardware virtualization assistance technologies, such as Intel Virtualization Technology [Uhlig et al. 2005], to run unmodified OS in guest. We have chosen HVM, in our study, to support a variety of OSs (both open source and proprietary OSs).

Both software virtualized I/O (including full device emulation and paravirtualized I/O) and hardware-assisted I/O (including direct I/O and Single Root I/O Virtualization or SR-IOV) are implemented in Xen as illustrated in Figure 1. In full device emulation guest I/O access from an unmodified native driver is trap-and-emulated in the device model (DM) so that a single hardware device can be shared among multiple VMs. In paravirtualized (PV) I/O [Fraser et al. 2004] a new FrontEnd (FE) driver running in the user domain or guest interacts (though shared-memory and asynchronous buffer-descriptor rings) with the BackEnd (BE) driver running in the service domain (Domain 0) which is responsible for multiplexing the I/O data of different VMs. PV I/O may also use hardware specific knowledge to access dedicated resources for performance such as VMDq [Santos et al. 2008]. In direct I/O the (pass-through) device is directly assigned to a particular VM which runs the native device driver to directly access the runtime resource without interceptions from the hypervisor utilizing I/O MMU to remap the DMA address from guest programmed physical address to host physical address.

In SR-IOV network virtualization domain0 runs the Physical Function (PF) driver to manage and configure the associated Virtual Functions (VFs) and each VF can be assigned to a guest, which runs a VF driver to directly access the VF runtime resource without interceptions from the hypervisor (again utilizing I/O MMU for the DMA address remapping).

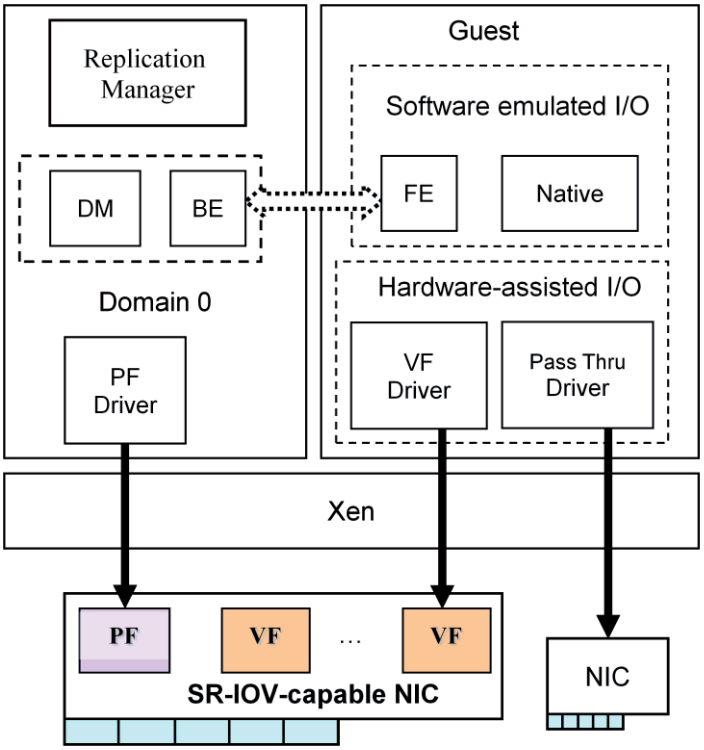

Fig. 1. Xen replication and I/O virtualization architecture.

2.2. Replication Challenges of I/O Virtualization

The replication manager in Xen is responsible for VM state replication. In live migration it transmits the entire VM state from the source to the target which involves multiple iterations of memory transmissions starting from a transmission of all memory pages followed by multiple iterations of delta memory page transmissions while the guest remains alive and a final iteration transmitting the dirty memory pages CPU and I/O state with the guest execution suspended [Clark et al. 2007]. In rapid checkpoint the replication manager periodically transmits delta VM state (including dirty memory pages CPU and I/O state) from the primary to the backup and the outbound packets are buffered until the checkpoint is successfully transmitted [Cully et al. 2008].

Although software emulated I/O can be properly replicated (by transmitting the I/O configuration state in the device model and Xen store respectively) it suffers from large performance and scalability overhead due to excessive hypervisor interventions in the bulk data processing path. On the other hand directed I/O and SR-IOV can achieve close to native performance and very good scalability but they cannot be properly replicated across different physical machines due to architectural limitations. In particular the state registers of a VF in an SR-IOV-capable network device may be read-only write-only or even not visible at all. Furthermore the dirty memory pages accessed by the DMA from a VF cannot be easily tracked.

2.3. Solution Discussions

In this subsection we discuss several potential solutions to address SR-IOV I/O replication challenges. If it is not impossible extending SR-IOV network virtualization

with software-only solutions to address the challenges is too expensive. One possible software-only solution is to dynamically switch between direct accessed VF and emulated VF at runtime and migration time respectively with a hardware-specific device model. It might be able to provide replication support as software emulated I/O does however the effort to develop a per device hypervisor specific model might be too costly for hypervisor vendors and it is almost impossible for hardware vendors to hook their device model to proprietary hypervisors which don't provide an open interface for the purpose.

Another alternative is to modify the VF driver to support replication with additional interfaces. The VF driver might be enhanced to buffer the outbound packets. However, it is hard to checkpoint the VF state, because the SR-IOV NIC is likely un-replicable, for example, the head register of the DMA descriptor ring in Intel 82576 NIC is a read-only register [Intel82576 2011]. Furthermore, the new interfaces talk with the migration manager and/or hypervisor and thus impose additional dependencies on the hypervisor, making the VF driver not only depend on the guest OS, but also on the hypervisor. It means a huge validation burden on hardware vendors when testing the configuration matrix.

In summary, software approaches suffer from highly expensive development costs and have their own major drawbacks. All of these motivate us to design an efficient I/O architecture to support efficient SR-IOV I/O replication.

3. RENIC

In this section, we propose our replicable NIC (ReNIC), an architectural extension to SR-IOV-capable NIC for efficient I/O replication. We elaborate our architectural designs in subsection 3.1. In subsection 3.2, we describe our extension in system software, such as hypervisor, replication manager, etc., in detail.

3.1. Replicable NIC

ReNIC is an architectural extension to SR-IOV-capable NIC and I/O MMU for efficient I/O replication. It extends SR-IOV-capable NIC to implement additional I/O interface to clone VF state (that is reading and writing the entire VF state) and to buffer the outbound packets. It also extends I/O MMU to efficiently track dirty pages. The hardware architecture of the ReNIC extension is shown in Figure 2.

Clone mode. The key challenge of the VF state clone is to guarantee the atomicity of read/write operations. In the working mode, the VF will have live traffic and may change the device state on the fly, which can cause the system software to see an inconsistent state during the clone and, consequently, may cause the failure of restoring the VF state on the target host.

ReNIC introduces a new clone mode for atomic read and write of the VF state. In the clone mode, the VF stops to process new packets (which relies on the network software stack to recover from packet loss), allowing the software to read/write the VF state atomically. The VF in ReNIC can return to the original working mode after the VF state is cloned.

Clone interface. When the VF is in clone mode, ReNIC provides a new interface in PF for reading and writing the entire VF state. By using the PF interface for VF state clone, ReNIC avoids modifications to the VF interface or driver of the original SR-IOV-capable NIC, which not only saves additional engineering efforts by re-using the unmodified VF driver, but also improves security by eliminating the dependencies on the VF driver and guest OS in the VF state replication process. For instance, the guest usually cannot freely configure the VF hardware behavior (such as changing the MAC address and VLAN settings) for security reasons. Therefore, in ReNIC, the VF

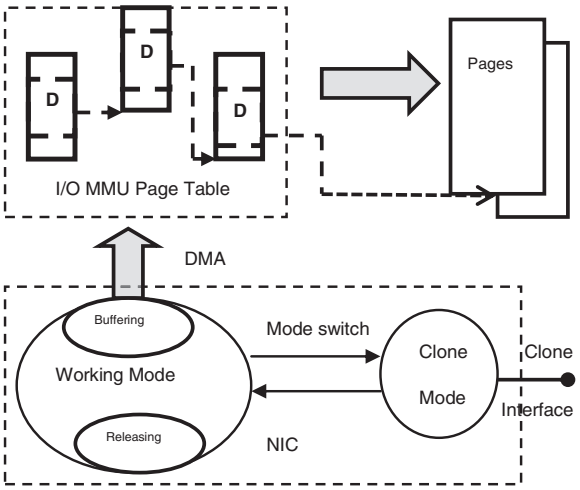

Fig. 2. ReNIC architectural extension.

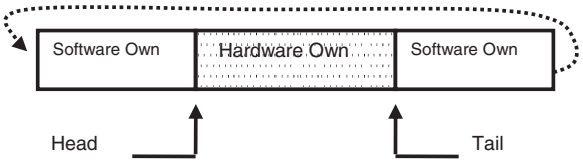

Fig. 3. DMA descriptor ring.

clone process is carried out by the PF driver, which is intended to be responsible for the configuration and management of the VFs in the first place [Dong et al. 2010].

Outbound packet buffering. Outbound packet buffering cannot be efficiently supported in state-of–the-art NICs, such as the Intel 82576 NIC [Intel82576 2011]. In these NICs, device DMA operations (such as packet transmission and receiving) are managed using a DMA descriptor ring, as well as head and tail registers, as shown in Figure 3. The software (or device driver) owns the DMA descriptors from the tail register to the head register, while the hardware owns the DMA descriptors from head to tail. To perform a DMA operation, the device driver first generates new DMA descriptors at the tail, and then updates the tail register, which will automatically notify the hardware to fetch the new DMA descriptors and start the DMA operation.

To efficiently buffer the outbound packets, ReNIC further divides the working mode of VF into two sub-modes, namely, buffering and releasing modes as illustrated in Figure 2. In addition, the tail register of the VF of SR-IOV-capable NIC becomes 2 separate registers in ReNIC: a guest tail register and a hardware accessible tail register. The guest tail register behaves exactly the same as the original tail register, which can be directly accessed by the VF driver. Consequently, the original VF driver of SR-IOV-capable NIC can still work in ReNIC. However, when the VF is set to the buffering mode by the PF, the guest tail register is not synchronized with the hardware tail register, which effectively buffers all the outbound packets between these two tail registers. When the VF is set to the releasing mode by the PF, these two tail registers are always synchronized by the hardware, which allows the buffered packets to be transmitted.

DMA dirty page tracking. To replicate the I/O state, memory pages modified by the device DMA operations must be efficiently tracked for rapid checkpoint and efficient

live migration. Unfortunately, DMA dirty page tracking is not supported in the existing I/O MMU.

To track the device DMA dirty pages, ReNIC extends the I/O MMU page table architecture to introduce an additional dirty bit ("D" bit in Figure 2). Whenever I/O MMU translates a DMA address and traverses the page table, I/O MMU generates an action to set the "D" bit of the page table, which will be executed once the DMA transaction is completed. However, this poses additional challenges due to the translation lookaside buffer (TLB) in the I/O MMU side and the translation cache enabled by address translation service (ATS) [SR-IOV 2008; Dong et al. 2010] in the PCIe function side. The I/O TLB intends to bypass page table traversal at translation time, and its benefit may be greatly reduced if page table traversal is needed for every translation. ReNIC addresses this issue by on-demand dirty page tracking. That is, the dirty page table tracking is turned on only when it is a must (that is at migration time).

Address translation service (ATS) provides a mechanism for ATS-capable PCIe function to cache the translation in PCIe function and uses the translated DMA address directly, which unfortunately bypasses the runtime I/O MMU translation and page table traversal. Consequently, ATS completely bypasses the "D" bit setting in ReNIC. Re NIC addresses this issue by configuring the I/O MMU to temporarily disable the ATS translation request for the in-migration VF and flush the translation cache in the VF, when dirty page tracking is turned on, without impacting other VFs or the benefits of ATS at runtime.

Whenever the I/O MMU detects a dirty page by the in- migration VF DMA, the "D" bits in the associated last level page table entry and the corresponding up level page table entries are set in memory, allowing efficient software traversal. However, it may impose additional latency due to the page table traversal and write back. The additional cost of each page table entry read and write can be estimated as up to 8 DRAM accesses (8 bytes per access), depending on the level of I/O MMU page table (typically 3 or 4) and number of "D" bits to be set. Because each DRAM burst access costs 100ns in modern computer systems (for example, with 3GHz CPU frequency), the additional latency due to a "D" bits setting for each memory page is less than 800ns. This impact is negligible [Zec et al. 2002] in modern computer systems.

3.2. Software Extension

To couple underlying architectural changes, the hypervisor needs to be enhanced to support our ReNIC for efficient VM replication. Figure 4 shows the extension to the hypervisor. The replication manager is responsible for the VM state (including the CPU, memory, and I/O state) replication in the live migration and rapid checkpoint.

For efficient delta memory transmission, the replication manager traverses the I/O MMU page table to indentify the DMA dirty pages, in addition to the pages dirtied by CPU. And for efficient VF I/O state transmission, the replication manager takes advantages of the clone service in the PF driver to first set the VF to the clone mode and read the VF state at the source, then transmit the state, and finally clone the VF state and set the VF to the working mode at the target host, as illustrated in Figure 5. In addition, the replication manager is also responsible for outbound packet buffering in rapid checkpoint (by utilizing the buffering service in the PF driver to set the VF to buffering or releasing mode, appropriately).

We have implemented the software extension for the ReNIC base on Xen and Remus [Cully et al. 2008] (using the SR-IOV-capable, Intel 82576 NIC) as an example, while simulating the hardware extensions (see section 4). In our implementation, The Intel igbvf driver, originally developed for the VF of the Intel 82576 NIC, is reused. The PF driver (Intel igb driver) is extended to support state clone and buffering services using the Linux ioctl interface.

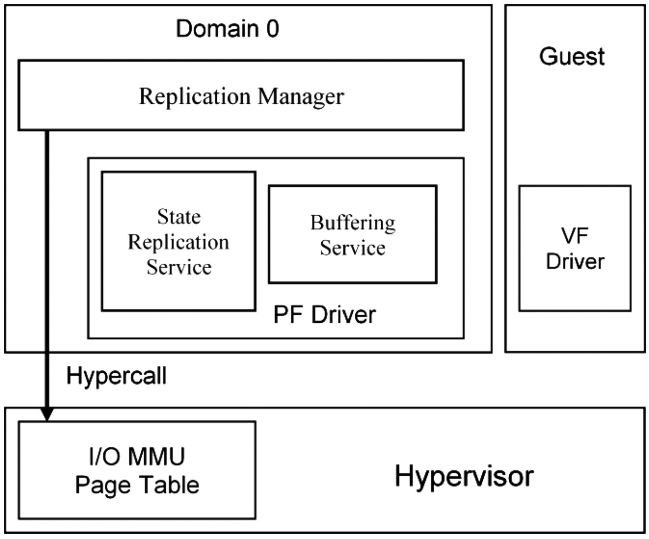

Fig. 4. Hypervisor extension for ReNIC.

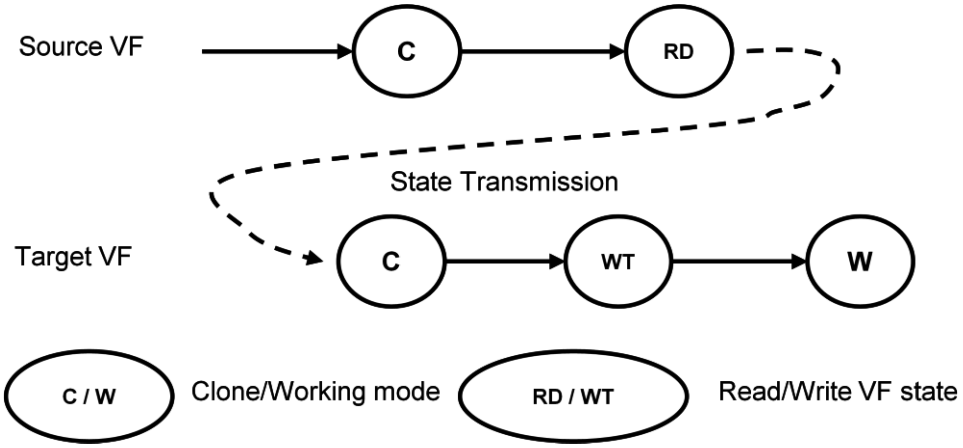

Fig. 5. Process of VF state replication.

4. SYSTEM SIMULATION

To evaluate our proposed architecture, in this section we develop a system simulator on multi-core platforms. Subsection 4.1 shows the challenges of traditional simulation. In subsection 4.2, we elaborate designs of the system simulator in detail and then present some performance optimizations in subsection 4.3.

4.1. Challenges of Simulation

Full system simulation and I/O device prototyping are two popular approaches used in I/O literature to evaluate a new I/O architecture [Liao et al. 2011; Shafer and Rixner 2007; Willmann et al. 2004]. However, they cannot provide proper system simulations for ReNIC. A full system simulator, such as simics [Magnusson et al. 2002], provides cycle accurate performance evaluation through software emulation, which can use thousands or even millions of cycles in the host to emulate one cycle in the emulated

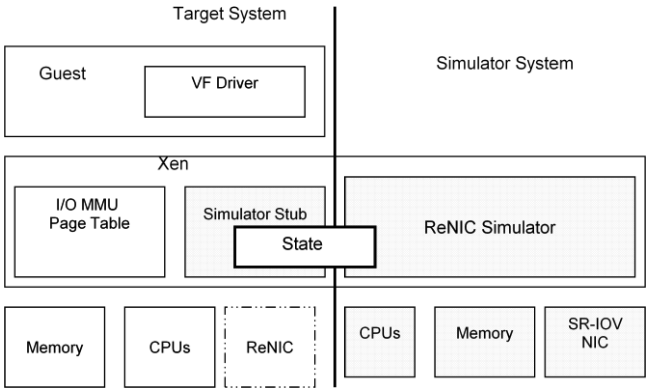

Fig. 6. Built-in ReNIC extension simulation.

system. It suffers from large performance overhead, which makes it very difficult, if not impossible, to simulate complicated and network-intensive real world applications (such as Web servers and rapid checkpoint). Nested virtualization may help in this case. Unfortunately, the virtualization overhead introduced by the additional layer of hypervisor is too high (35% and 45% CPU usage overhead for direct I/O and virtio, respectively) [Ben-Yehuda et al. 2010]. A prototype I/O device uses an existing programmable network interface or FPGA [Shafer and Rixner 2007; Watson et al. 2006; Willmann et al. 2004] to simulate new I/O architecture. However, supporting both advanced NIC features (such as SR-IOV) and chipset (such as I/O MMU) in simulation requires excessive hardware modifications. All of these considerations motivate us to design a system simulator to evaluate the new I/O architecture online on real multi-core platforms by leveraging multiple cores.

4.2. System Simulation

From the software point of view, a device interacts with the software (or device driver) in three ways: register, interrupt, and shared memory. Software programs the device through registers, while the device notifies the processor through asynchronous interrupts. Shared memory is widely used for the device to communicate with the processor for massive data movement through the DMA.

We have implemented a system simulator in Xen for ReNIC, as shown in Figure 6. In the host, dedicated CPUs, memory, and the SR-IOV-capable NIC are reserved for the simulator system, so that it can simulate the ReNIC extension (including the NIC architectural extension and the I/O MMU extension) for the target system (running using the remaining resources in the host). A simulator stub residing in the target system traps the guest access of ReNIC state and provides the fast path emulation, leaving the slow path emulation to the simulator system. Consequently, the simulation overhead is greatly reduced and our system simulator can achieve close to native performance (refer to subsection 0 for the accuracy of the simulator).

Register. The ReNIC state is shared in memory between the ReNIC simulator and the simulator stub, providing fast path simulation. That is, when the guest reads or writes the ReNIC registers, the read/write operations are trapped-and-emulated by the hypervisor and can be directly simulated by the simulator stub (through reading or writing the shared state in memory). On the other hand, the ReNIC simulator is responsible for the slow path simulation (i.e., periodically updating the shared state, based on the VF states of the SR-IOV NIC, and monitoring the shared state to simulate the changes).

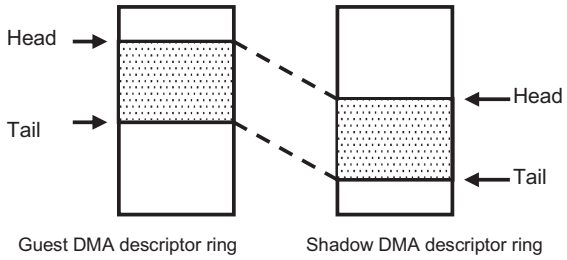

Fig. 7. Guest DMA descriptor ring is mapped to shadow DMA descriptor ring.

Consequently, the simulation overheads of reading or writing a ReNIC register in the fast path are limited to a VM-exit to the hypervisor and a memory read/write by the simulator stub, which cost only several thousand extra CPU cycles, according to our measurement. In addition, since accessing Memory Mapped IO (MMIO) registers is extremely slow in the modern platform (due to the involvement of slow external bus cycles), such register accesses are already minimized in the device driver. Consequently, this additional simulation overhead has very low impact on the performance of the target system.

DMA. The DMA descriptor ring of the SR-IOV NIC cannot be directly used by the VF driver running in the guest in our simulator. This is because the head register of the descriptor ring cannot be updated by the simulator (read only), and, consequently, it cannot be properly cloned at the target in live migration.

To properly simulate the DMA descriptor ring for ReNIC, the ReNIC simulator creates a shadow DMA descriptor ring for each guest DMA descriptor ring, as shown in Figure 7. In the fast path, the device driver in the guest directly performs the DMA operations using the guest ring, and the guest head and tail registers can be directly accessed in the target system. The shadow DMA descriptor ring is used by the SR-IOV NIC, and the ReNIC simulator is responsible for synchronizing the guest and shadow rings.

The synchronization overheads of the guest and shadow rings are very small in practice. Unlike the shadow page table of MMU, which needs to trap-and-emulate each update of the guest page table to synchronize the shadow and guest page tables [Adams and Agesen 2006], the guest and shadow descriptor rings only need to be synchronized when the guest tail register is updated. This is because the device only owns the portion of the ring from the head to tail, as Figure 3 shows, and therefore the guest device driver needs to update the tail register to notify the device of its changes to the descriptor ring.

Interrupt. Whenever an interrupt from the SR-IOV NIC arrives, the ReNIC simulator handles the interrupt (i.e., applying the semantics of the interrupt to the guest DMA descriptor ring, such as updating the guest descriptors), and then signals an inter-processor-interrupt to the CPU in the target system to simulate the corresponding interrupt of ReNIC. This may introduce several thousand CPU cycles of delay (that is, several microseconds in modern platforms) in interrupt dispatch, which has very low impacts on interrupt deferral, as the interrupt frequency is usually limited to several KHz in modern platforms.

I/O MMU. When the ReNIC simulator receives an interrupt from the SR-IOV NIC that indicates there are new incoming packets, the ReNIC simulator sets the "D" bit for both the page of the packet and the page of the descriptor ring (as the descriptor ring needs to be updated to reflect the completion of the packet, according to the hardware semantics) in the I/O MMU. As for the outbound packet, ReNIC only sets the "D" bit

Table I. Platform Configurations

Physical Platform	Dual-socket 3.0GHz quad-core Intel Xeon X5670 processor with SMT enabled, 24GB Memory per socket, an Intel 825761Gb NIC and an Intel 82598 [Intel82598 2011] 10Gb NIC
Simulator System	4 cores and 24 GB memory in socket 1, an Intel 82576 NIC
Target (ReNIC) System	4 cores with SMT enabled and 24 GB memory in socket 0, an Intel 82598 10Gb NIC and a simulated 1Gb Re NIC

for the page of the descriptor ring in I/O MMU. Because the simulation is performed completely by the simulator system, there are almost no performance impacts on the target system.

In addition, as discussed in Section 3.1, the additional cost of I/O page table traversal and update (setting "D" bit) can bring up to 800ns additional latency for each DMA memory page update (that is 4 read for page table traversal and 4 write for write back, in the worst case). To simulate this potential performance impact in ReNIC, the simulator system, running in approximately the same speed as real hardware, as demonstrated in subsection 0, introduces up to 1us latency, when processing the incoming packet (that is setting the "D" bit and updating the head register of guest DMA descriptor).

4.3. Optimizations

As described in the previous section, a guest MMIO register access in our simulator will trigger a VM-exit to the hypervisor, and the simulator stub needs to update the related memory state for the MMIO register. However, in today's computer system, the VM-exit handler is only aware of the fault guest IP (the instruction pointer), but not the MMIO address. Consequently, the simulator stub has to fetch the guest instruction and decode it during simulation. Unfortunately, the instruction fetch-and-decode is very expensive, which not only takes a large amount of CPU cycles, but also introduces a lot of additional cache flush.

To reduce this overhead, the simulator stub will cache the mapping between the guest IP of the VM-exit and the associated MMIO access. When a new VM-exit is triggered, the simulator stub will first look up its cache, and the expensive instruction fetch-and-decode can be avoided when a cache hit is detected. Since guest MMIO register accesses are usually minimized (e.g., only 2 in the Intel igbvf driver) and only 1 VF is assigned to the guest, this simple caching scheme works very well in practice.

5. PERFORMANCE EVALUATION

In this section, we evaluate the ReNIC system through extensive system simulation. We describe our experiment setup in subsection 5.1 and validate our system simulator methodology in subsection 5.2. Based on our simulator, we then present experimental results of live migration and rapid checkpoint in subsection 5.3 and 5.4, respectively.

5.1. Experiment Setup

In our experiment, we run the system simulator for ReNIC on two dual-socket Intel Xeon servers (one as the source/primary server, and the other as the target/backup server). The detailed platform configurations are shown in Table I. In each server, the simulator system runs on one dedicated processor socket (including both the CPU and memory), while the target system runs on the other socket. The detailed software configurations are shown in Table II.

In our experiment, both servers and a remote client are connected to a GbE switch (for running the benchmarks), and the two servers are also directly connected using the 10GbE NICs (for efficient rapid checkpoint).

Table II. Software Configurations

Domain 0	Linux 2.6.18 64 bits OS is run with 4 VCPUs bounded to 4 threads of socket 0, respectively. IGB driver version 1.3.21.5 is used as base of ReNIC PF driver, and igbe driver version 3.0.12-NAPI is used for 10 Gb Intel 82598 NIC
Guest	RHEL 5.4 Linux updated to 2.6.32 kernel with 4 VCPUs bound to the other 4 threads of socket 0, 1 GB memory from socket 0, as well as a dedicated VF
Xen	Changset 21046

Table III. Ping Latency

	Average Latency (ms)	Standard Deviation
Real System	0.27754	0.000413
Simulated System	0.27744	0.000174

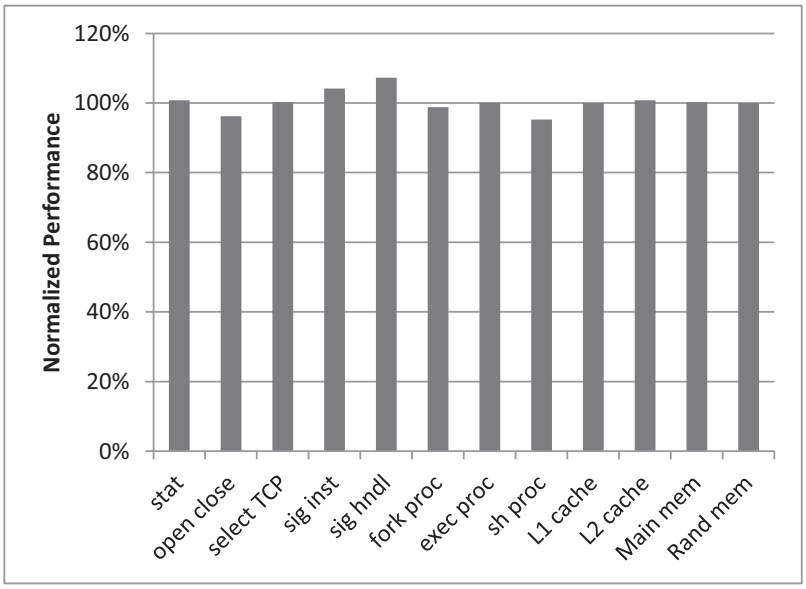

Fig. 8. CPU and memory performance between simulated and real systems.

5.2. Validating Accuracy of the Simulator

To validate the accuracy of our system simulator, we compare the performance between the simulated target (ReNIC) system and the real system, with an SR-IOV NIC along different dimensions: CPU, memory, and I/O.

Figure 8 compares the CPU and memory performance between the target system and the real system using the Lmbench microbenchmark. The maximum performance difference between the ReNIC system and the real one is within 7.29% and 0.76%. The average performance difference is 2.79% and 0.34% for CPU intensive workload and memory intensive workload, respectively.

In addition, Figure 9 compares the I/O performance between the target system and the real SR-IOV system, using the netperf microbenchmark with 1472-byte packets. The maximum bandwidth difference between the simulated system and real one is negligible (the maximum difference is 2% and average difference is 0.5%), and there is almost no latency difference between the two systems, either (as shown in Table III). The CPU usage difference between the simulated system and real one varies from 2.3%

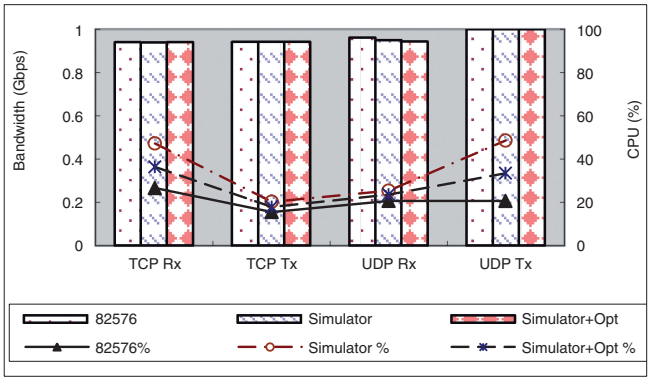

Fig. 9. TCP/UDP performance between real and simulated systems running netperf.

in TCP transmit configuration to 12.9% in UDP Transmit configuration. The average CPU usage difference between simulated system and the real one is about 7%.

The above results reveal that our system simulator is accurate enough for the performance evaluation of ReNIC, mostly due to the lightweight simulation in the fast path. The extra CPU usage in the simulated system comes from the lengthened MMIO access path and additional cache pollution, due to the simulator stub. We have instrumented the device driver in guest, and the instrumentation results show that the average MMIO access cost increases from 3.5K CPU cycles in real system to 11.9K cycles in the simulated system. The caching optimization helps reduce the expensive instruction fetch-and-decode and decrease the average guest MMIO access cost to 8.4K cycles (about 3-4% extra CPU usage), as shown in Figure 9. In addition, the target system has a higher Last Level Cache (LLC) miss rate (except for the TCP transmit case) due to the simulator stub, which accounts for the remaining CPU usage increase.

5.3. Live Migration

By using our system simulator, we compare the performance of software virtualized I/O and ReNIC for live migration, using the netperf and WebBench benchmarks. In our experiment, the migration starts at the 4.5^{th} second.

We don't compare our solution with VMDq because the VMDq driver for the Intel 82576 NIC is not ready yet and the additional gain brought by VMDq (from 11K to 8K cycles/packet as reported in [Santos et al. 2008]) over PV I/O may not bring too much difference (3K cycles/packet savings means 8% CPU cycles of a core) to our conclusion, which achieves 54% CPU usage reduction, in 1 Gbps network.

Netperf. Figures 10–13 show the performance and CPU usage of migrating an HVM running netperf with vE1000, PV I/O, teaming driver and ReNIC, respectively. vE1000 is an implementation of full device emulation of Intel PRO/1000 NIC device, and it can only achieve 788 Mbps bandwidth at runtime with the servicing CPU (in domain 0) almost saturated. PV I/O can achieve line rate (1 Gbps), with 58% CPU usage in domain 0 to process the guest I/O requests.

The teaming driver [Zhai et al. 2008] uses the VF of the SR-IOV NIC during runtime and, therefore, can achieve similar performance with ReNIC. However , it introduces additional service shutdown to switch the network interface at the time of live migration and puts dependencies on guest OS support of hotplug and teaming driver framework, which are not supported in many existing OSs, such as Windows. In addition, teaming driver introduces an additional 0.6 second service down time (at the 4.5^{th} second). This is because it needs to switch network interface from VF to software

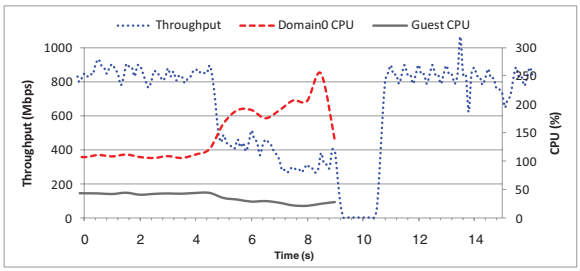

Fig. 10. Migrating an HVM running netperf on top of vE1000: migration starts at 4.5th sec, and the service shuts down at 9.4th sec and restored at 10.8th sec.

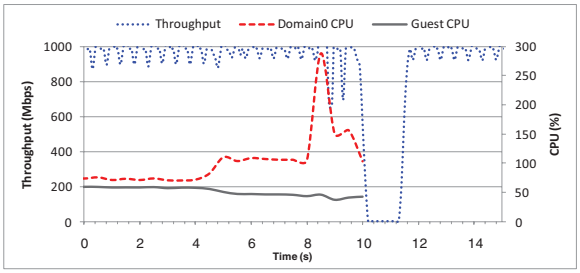

Fig. 11. Migrating an HVM running netperf with PV I/O: migration starts at 4.5th sec, and the service shuts down at 10.4th sec and restored at 11.8th sec.

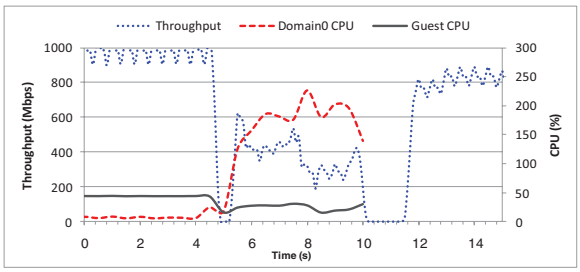

Fig. 12. Migrating an HVM running netperf with teaming driver: migration starts at 4.5th sec with additional service shutdown time of 0.6 second, and the service shuts down at 10.3th sec and restored at 11.8th sec.

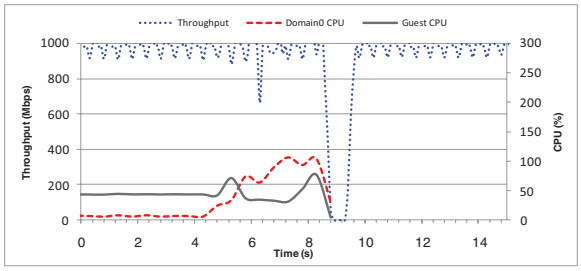

Fig. 13. Migrating an HVM running netperf with ReNIC: migration starts at 4.5th sec, and the service shutdown at 9th sec, and restored at 9.7th sec.

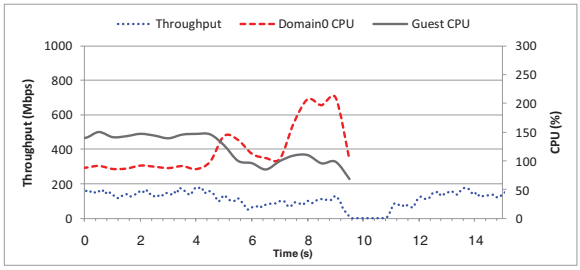

Fig. 14. Migrating a VM running WebBench with vE1000: migration starts at 4.5th sec, and the service shuts down at 9.8th sec and restored at 11.1th sec.

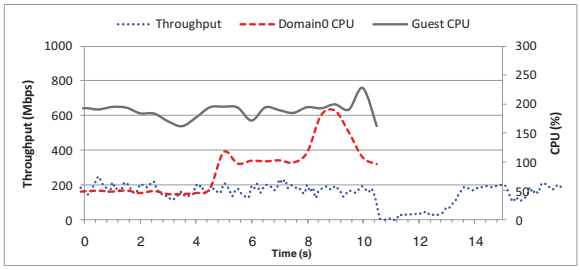

Fig. 15. Migrating a VM running WebBench with PV NIC: migration starts at 4.5th sec, and the service shuts down at 10.8th second and restored at 11.4th sec with low throughput till 13.1th sec.

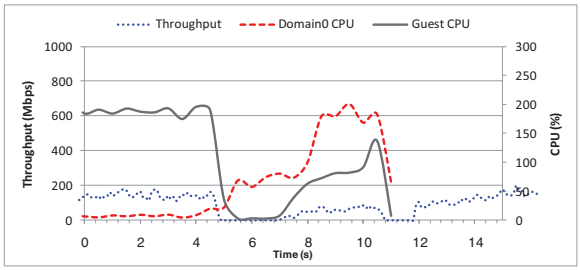

Fig. 16. Migrating a VM running WebBench with teaming driver: migration starts at 4.5th sec with an additional service shutdown time of 2.4 seconds, and the service shuts down at 10.9th sec, restored at 12.0th sec.

emulated network, which may drop pending Tx packets in the VF side and drop Rx packets, before the virtual switch in domain 0 re-routes the incoming packets to the emulated network device. In addition, compared to ReNIC, the teaming driver also suffers from extra CPU usage, as well as longer service shutdown time (1.1 seconds at the 10th second, resulting in a 1.7 second total service down time) during migration time.

ReNIC can achieve line rate with only 4% CPU usage in domain 0 at runtime, and 74% CPU usage at migration time (due to the involvement of replication manager). That is, ReNIC achieves 54% CPU usage reduction, compared to software based I/O virtualization (58% vs. 4%) at runtime. In addition, ReNIC also reduces the service shutdown time (due to the VM state transmission in the final iteration), from about 1.4 seconds in vE1000 and PV I/O to only 0.8 seconds.

WebBench. The performance and CPU usage of migrating an HVM running WebBench with vE1000, PV I/O, teaming driver and ReNIC is shown in Figures 14–17.

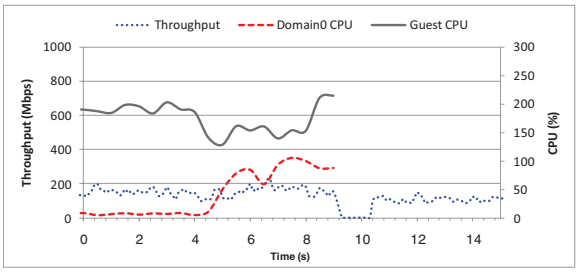

Fig. 17. Migrating a VM running WebBench with ReNIC: migration starts at 4.5th sec, and the service shuts down at 9.3th sec and restored at time 10.3th sec.

At runtime, vE1000, PV I/O, teaming driver and ReNIC achieve an average of 133 Mbps, 174 Mbps, 141 Mbps, and 143 Mbps throughput, respectively. In the meantime, vE1000 and PV I/O consumes 90% and 47% CPU usage in domain 0, respectively, while ReNIC consumes only 7% CPU usage. PV I/O has higher CPU usage due to packet copy, but achieves higher performance than ReNIC, due to the improved cache locality (which is critical to memory/cache intensive workloads, such as WebBench). On the other hand, its performance may be not sustainable as the CPU in domain 0 may be saturated as the system scales up (for example, PV I/O saturates CPU at 3.6 Gbps in a 10 Gbps network environment, while SR-IOV can maintain line rate, achieving more than 3x throughput advantage over PV I/O [Dong et al. 2010]). At migration time, throughput of vE1000 decreases significantly due to the contention of CPU, memory, and cache, between the guest (which is running the workload) and domain 0 (which is running the device model and the replication manager). In PV I/O, the throughput decreases slightly, reflecting the fact that CPU usage in domain 0 is yet not saturated. The teaming driver incurs much longer service shutdown time (2.4 seconds) to switch the network interface (which may drop pending Tx packets in the VF side and drop Rx packets before the virtual switch in domain 0 re-routes the incoming packets to the emulated network device), in addition to the service shutdown time at the final stage (1.1 seconds starting from 10.9th second), because the guest is much busier when running WebBench, compared with netperf. ReNIC has the same throughput during both the run time and migration time, reflecting the fact that ReNIC does not resort to domain 0 in its I/O path.

In summary, ReNIC achieves line rate with up to 54% CPU usage reduction in domain 0 (58% vs. 4%) than PV I/O at run time, which can lead to more than 3x throughput advantage in higher bandwidth networks such as 10GbE (as demonstrated in the original SR-IOV study [Dong et al. 2010]). In addition, during migration time, ReNIC can achieve about 50% reduction of service shutdown time, compared to vE1000 and PV I/O (from 1.4 second to 0.7 second), and over 71% reduction of service shutdown time, compared to teaming driver (from 3.5 seconds to 1.0 second), a significant reduction of impact to SLA (service level agreement).

5.4. Rapid Checkpoint

In this subsection, in addition to live migration, we also compare the performance of rapid checkpoint running WebBench with ReNIC and with software emulated I/O. We use vE1000 instead of PV I/O as our base, because the current rapid checkpoint implementation does not support PV I/O in HVM yet. In addition, because the network bandwidth usage in WebBench is only 150 Mbps (as shown in Figure 14 and Figure 15), the performance difference between PV I/O and vE1000 in WebBench is minimized. Furthermore, in rapid checkpoint, the network bandwidth is further throttled due to

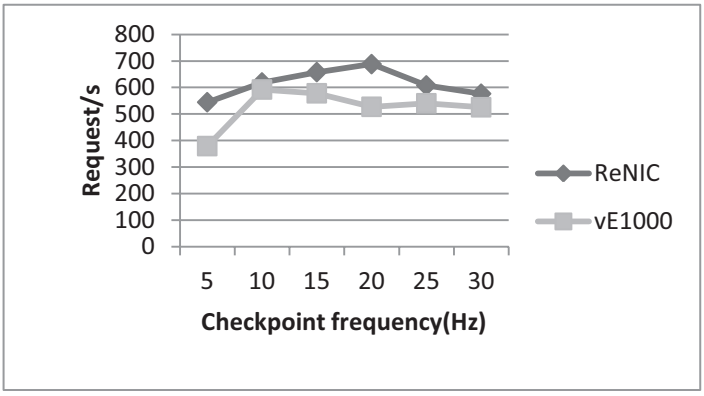

Fig. 18. Throughput of Webbench in rapid checkpoint.

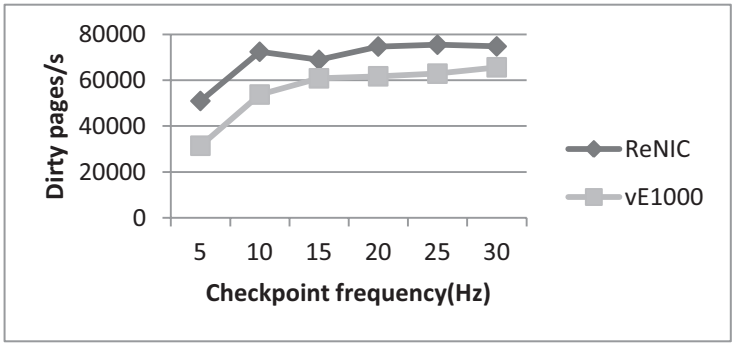

Fig. 19. Dirty page numbers generated when running WebBench in rapid checkpoint.

the outbound packet buffering [Cully et al. 2008], and therefore, the performance difference between vE1000 and PV I/O is further reduced. In addition, teaming driver cannot be properly used in rapid checkpoint, because it relies on the OS hotplug framework and cannot support rapid checkpoint with up to 40 Hz frequency (the latencies of virtual hot removal/add-on and the shadow recovery time are both on the order of one second).

Figure 18 shows the performance of WebBench in rapid checkpoint, with different checkpointing frequencies. Lower checkpoint frequency minimizes the overhead of VM state checkpoint, which may help performance. However, lower checkpoint frequency imposes longer latency to network response due to the outbound packet buffering. The peak performance of ReNIC (688 requests/s) happens at 20 Hz checkpointing frequency, about 16.2% higher than that of the software emulated I/O (592 requests/s at 10 Hz checkpointing frequency), reflecting the fact that ReNIC consumes far fewer CPU cycles in I/O path. In addition, Figure 19 shows the dirty page numbers generated per second for different checkpointing frequencies. ReNIC generates more dirty pages due to its higher throughput.

Figures 20 and Figure 21 show the breakdown of the costs of various operations in rapid checkpoint. ReNIC spends more CPU time in transmitting the dirty pages compare with software I/O. In the meantime, ReNIC achieves high throughput (as shown in Figure 18), less CPU cycle cost, reflecting the fact it uses fewer CPU cycles in I/O data processing.

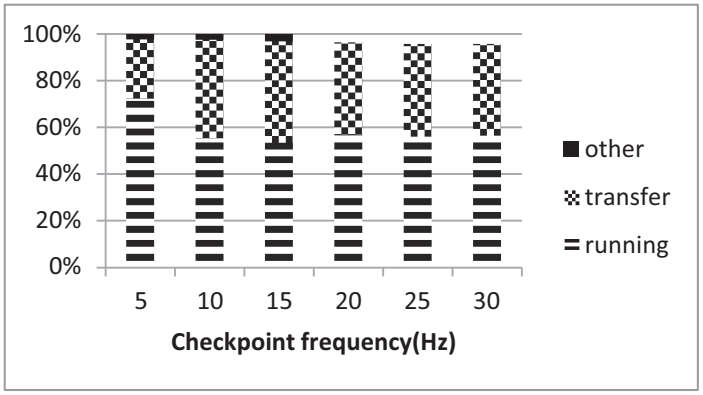

Fig. 20. Time cost in VM replication of vE1000.

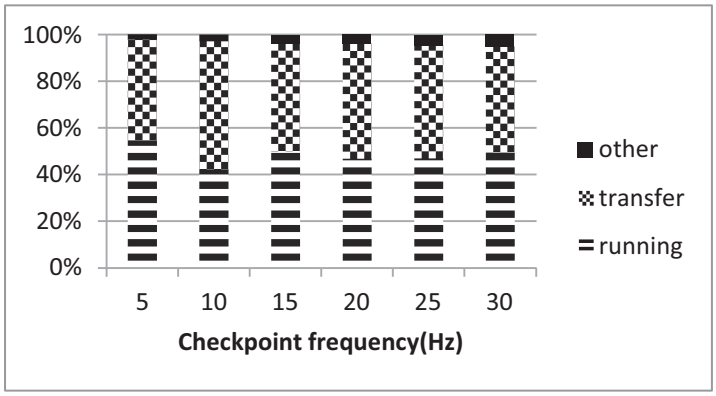

Fig. 21. Time cost in VM replication of ReNIC.

6. RELATED WORK

There is a lot of research in the literature on the virtual device based VM migration and its optimization. Clark et al. [2005] studied live VM migration base on Xen paravirtualization. Hines et al. [2009] explored post-copy live migration of virtual machines. Sapuntzakis et al. [2009] optimized VM migration to quickly move the state of a running computer across the network Nelson et al. [2005] studied fast transparent migration of virtual machines. Bradford et al [2007] explored live wide-area migration of virtual machines. Voorsluys et al. [2009] studied the cost of virtual machine migration in Clouds, which cannot be disregarded in systems where service availability and responsiveness are governed by strict service level agreements. However, none of them can use advanced hardware features of modern NIC, such as SR-IOV, and suffer from great overheads of I/O virtualization.

There is also a lot of research regarding VM replication and new usage models based on VM replication. Hypervisor-based fault tolerance is achieved through coordinating a primary virtual machine with its backup [Bressoud and Schneider 1996]. SMP-Revirt replays execution for multiple-processor VM [Dunlap et al. 2008]. Intrusion analysis is enabled through virtual-machine logging and replay [Dunlap et al. 2002]. Live migration of virtual machine through full system trace and replay is studied in Liu et al. [2009]. Asynchronous virtual machine replication or rapid checkpoint is proposed in

Cully et al. [2008]. Fast memory state synchronization to rapid checkpoint is explored based on speculative execution of backup VM [Lu and Chiveh 2009]. However, none of these existing solutions and usage models takes advantage of hardware-assisted I/O virtualization. Consequently, they cannot leverage both performance and VM replication capability.

Since the rebirth of virtualization technology, there are lots of studies performed to improve its performance [Guo et al. 2009; Liao et al. 2010; Santos et al. 2008; Raj and Schwan 2007]. In software, Liao et al. [Guo et al. 2009; Liao et al. 2010] proposed some scheduling optimizations to improve I/O performance. Besides software optimizations, hardware-assisted solutions, such as VMDq [Santos et al. 2008], self-virtualized devices [Raj and Schwan 2007] and SR-IOV [Dong et al. 2010] are proposed in virtualization to achieve high-performance. Solarflare [Mansley et al. 2007] hooks hardware specific driver modules in the Xen PV driver framework to take the advantage of specific NIC features. It can directly access hardware resource at runtime for performance and fall back to PV I/O path at migration time. However, the cost of I/O path fallback is not clear, nor can it provide a generic virtualization solution or rapid checkpoint.VMM-bypass I/O base on InfiniBand network and its migration support with RDMA are explored in Huang et al. [2007]; and Liu et al. [2006]. However, they require service from backend guest for security and address remapping, which may become a performance bottleneck in high bandwidth communication, and they have never explored the support for rapid checkpoint. In addition, these solutions are also tightly coupled with InfiniBand specific technology and it is unclear if they can be applied to industry standard solutions (such as direct PCIe device and SR-IOV). In summary, none of the above solutions can support live I/O state replications, nor can they support rapid checkpoint efficiently.

Migration of direct I/O and SR-IOV is explored, as well. Teaming driver and virtual hot removal/add-on of the pass through device across migration is used to avoid I/O state replication in migration [Zhai et al. 2008]. However, the hot removal/add-on and teaming driver framework may not be properly supported by some OSs (such as Windows). Furthermore, the teaming driver introduces additional service shutdown time to switch network interface at migration, which may significantly impact the SLA (service level agreement). The shadow driver of the pass through device [Kadav and Swift 2009] provides another solution to live migration by logging-and-replaying the operations between OS kernel and device driver. However, it requires guest OS source code modifications, which are unavailable for proprietary OSs. Furthermore, neither virtual hot removal/add-on nor shadow driver can support rapid checkpoint with up to 40 Hz frequency, because the latencies of virtual hot removal/add-on and the shadow recovery time are both on the order of one second, which cannot meet the frequency requirement of rapid checkpoint. None of the above solutions can support live I/O state replications, nor can they support rapid checkpoint efficiently for unmodified guest OS.

SLA (service level agreement), such as service availability and responsiveness, in a virtualized environment and impact of live migration are studied. The real time virtual machine is explored to meet appropriate timeliness guarantees to real-time applications running in a VM [Cucinotta et al. 2008]. Techniques to meet the strict timing constraints of (soft) real-time virtualized applications when the VM is underlying a live migration are proposed [Checconi et al. 2009]. Cost of VM live migration in clouds to meet SLA is studied as well [Voorsluys et al. 2009]. They don't address the I/O induced service shutdown issue in live migration.

7. CONCLUSION AND FUTURE WORK

We have proposed ReNIC, an architectural extension to SR-IOV network virtualization for efficient I/O replications among platforms with the same NIC (common in the environment of Clouds and a well managed resource pool [Gmach et al. 2008;

Vmware 2011]). We have extended the Xen hypervisor and the Remus rapid checkpoint solution to support the new architectural features to provide both great virtualization performance and high availability (through live migration and rapid checkpoint). We have also developed a system simulator to evaluate the benefits of ReNIC architectural extension. The experimental results show that ReNIC can achieve up to 54% CPU usage reduction compared to software based I/O virtualization at runtime, and up to 16.2% performance advantage over software based I/O virtualization in rapid checkpoint. In addition, during migration time, ReNIC can achieve up to 71% reduction of service shutdown time compared to teaming driver (from 3.5 seconds to 1.0 second), which significantly minimizes the impact to service level agreement. Our future work includes I/O replication support for synchronous VM replication (that is log-and-replay) [Bressoud and Schneider 1996], extending ReNIC to support paravirtualized virtual machine, optimizing the performance of rapid checkpoint, and studying the possibility of minimal (but standardized) architectural requirements across NIC models or even vendors, without sacrificing the sufficient hardware differentiation.

REFERENCES

Adams, K. and Agesen, O. 2006. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12)*. IEEE, 2–13.

Amazon EC2 2008. Amazon EC2 service level agreement. Amazon EC2, http://aws.amazon.com/ec2-sla/.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. 2003. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*. ACM, 164–177.

Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., and Yassour, B. A. 2010. The Turtles Project: Design and Implementation of Nested Virtualization. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI'10)*. USENIX, 423–436.

Bradford, R., Kotsovinos, E., Feldmann, A., and Schioberg, H. 2007. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'07)*. ACM, 169–179.

Bressoud, T. C. and Schneider, F. B. 1996. Hypervisor-based fault tolerance. *ACM Trans. Computer Syst.* 14, 1, 80–107.

Checconi, F., Cucinotta, T., and Stein, M. 2009. Real-time issues in live migration of virtual machines, In *Proceedings of the IEEE International Conference on Parallel Processing (Euro-Par'09)*. IEEE, 454–466.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. 2005. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Networked Systems Design and Implementation (NSDI'05)*. USENIX, 273–286.

Cucinotta, T., Anastasi, G., and Abeni, L. 2008. Real-time virtual machines. In *Proceedings of the 29th IEEE Real-Time System Symposium (RTSS'08)*, (Work in Progress Session). IEEE.

Cully, B., Lefebvre, G., Meyer, D., and Feeley, M. 2008. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th Conference on Networked Systems Design and Implementation (NSDI'08)*. USENIX, 161–171.

Dong, Y., Yang, X., Li, X., Li, J., Tian, K., and Guan, H. 2010. High performance network virtualization with SR-IOV. In *Proceedings of the 16th IEEE International Symposium on High-Performance Computer Architecture (HPCA'10)*. IEEE, 271–280.

Dunlap, G. W., King, S. T., Cinar, S., Basrai, M. A., and Chen, P. M. 2002. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*. USENIX, 211–224.

Dunlap, G. W., Lucchetti, D. G., Fetterman, M. A., and Chen, P. M. 2008. Execution replay of multiprocessor virtual machines. In *Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'08)*. ACM, 121–130.

Fraser, K., Hand, S., Neugebauer, R., Pratt, I., Warfield, A., and Williams, M. 2004. Safe hardware access with the Xen virtual machine monitor. In *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On Demand IT InfraStructure (OASIS'04)*.

GMACH, D., ROLIA, J., CHERKASOVA, L., BELROSE, G., TURICCHI, T., AND KEMPER, A. 2008. An integrated approach to resource pool management: policies, efficiency and quality metrics. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'08)*. IEEE, 326–335.

GUO, D., LIAO, G., AND BHUYAN, L. N. 2009. Performance characterization and cache-aware core scheduling in a virtualized multi-core server under 10GbE. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'09)*. IEEE, 168–177.

HINES, M. R., DESHPANDE, U., AND GOPALAN, K. 2009. Post-copy live migration of virtual machines. *SIGOPS Operat. Syst. Rev. 43*, 3, 14–26.

HUANG, W., GAO, Q., LIU, J., AND PANDA, D. K. 2007. High performance virtual machine migration with RDMA over modern interconnects. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'07)*. IEEE, 11–20.

INTEL82598 2011. Intel® 82598 10 Gigabit Ethernet Controller. http://ark.intel.com/products/41282/Intel-82599ES-10-Gigabit-Ethernet-Controller.

INTEL82576 2011. Intel® 82576 Gigabit Ethernet Controller. http://ark.intel.com/products/37166/Intel-82576EB-Gigabit-Ethernet-Controller.

KADAV, A. AND SWIFT, M. M. 2009. Live migration of direct-access devices. *ACM SIGOPS Operat. Syst. Rev. 43*, 3, 95–104.

LIU, H., JIN, H., LIAO, X., HU, L., AND YU, C. 2009. Live migration of virtual machine based on full system trace and replay. In *Proceedings of the 18th ACM International Symposium on High performance Distributed Computing (HPDC'09)*. ACM, 101–110.

LIAO, G., ZHU, X., AND BHUYAN, L. N. 2011. A new server I/O architecture for high speed networks. In *Proceedings of the 17th IEEE International Symposium on High-Performance Computer Architecture (HPCA'11)*. IEEE, 255–265.

LIAO, G., BHUYAN, L. N., WU, W., YU, H., AND KING, S. R. 2010. A new TCB cache to efficiently manage TCP sessions for Web servers. In *Proceedings of the 6th ACM/IEEE Symposium on Architecture for Networking and Communication Systems (ANCS'10)*. ACM, 1–10.

LIU, J., HUANG, W., ABALI, B., AND PANDA, D. K. 2006. High performance VMM-bypass I/O in virtual machines. In *Proceedings of the USENIX Annual Technical Conference (USENIX'06)*. USENIX, 3–3.

LU, M. AND CHIUEH, T. 2009. Fast memory state synchronization for virtualization-based fault tolerance. In *Proceedings of the 39th IEEE/IFIP International conference on Dependable Systems & Networks (DSN'09)*. IEEE, 534–543.

MAGNUSSON, P. S., CHRISTENSSON, M., ESKILSON, J., FORSGREN, D., HALLBERG, G., HOGBERG, J., LARSSON, F., MOESTEDT, A., AND WERNER, B. 2002 2002002022. Simics: A full system simulation platform. *IEEE Computer. 35*, 2, 50–58.

MANSLEY, K., LAW, G., RIDDOCH, D., BARZINI, G., TURTON, N., AND POPE, S. 2007. Getting 10 Gb/s from Xen: Safe and fast device access from unprivileged domains, In *Proceedings of the Conference on Parallel Processing (Euro-Par'07)*. 224–233.

NELSON, M., LIM, B., AND HUTCHINS, G. 2005. Fast transparent migration for virtual machines. In *Proceedings of the USENIX Annual Technical Conference (USENIX'05)*. USENIX, 391–394.

RAJ, H. AND SCHWAN, K. 2007. High performance and scalable I/O virtualization via self-virtualized devices. In *Proceedings of the 16th International Symposium on High Performance Distributed Computing (HPDC'07)*. ACM, 179–188.

SR-IOV 2008. PCI special interest group. http://www.pcisig.com/home.

SAPUNTZAJIS, C. P., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M. S., AND ROSENBLUM, M. 2002. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*. USENIX, 377–390.

SANTOS, J. R., TURNER, Y. JANAKIRAMAN, G., AND PRATT, I. 2008. Bridging the gap between software and hardware techniques for I/O virtualization, In *Proceedings of the USENIX Annual Technical Conference (USENIX'08)*. USENIX, 29–42.

SHAFER, J. AND RIXNER, S. 2007. RiceNIC: A reconfigurable network interface for experimental research and education. In *Proceedings of the Workshop on Experimental Computer Science (ExpCS'07)*. ACM.

UHLIG, R., NEIGER, G., RODGERS, D., SANTONI, A. L., MARTINS, F. C. M., ANDERSON, A. V., BENNETT, S. M., KAGI, A., LEUNG, F. H., AND SIMTH, L. 2005 2005005055. Intel Virtualization Technology. *IEEE Computer 38*, 5, 48–56.

VMWARE 2011. vSphere resource management guide. http://www.vmware.com/pdf/vsphere4/r41/vsp_41_resource_mgmt.pdf.

VOORSLUYS, W., BROBERG, J., VENUGOPAL, S., AND BUYYA, R. 2009. Cost of virtual machine live migration in Clouds: A performance evaluation. In *Proceedings of the 1st International Conference on Cloud Computing (CloudCom'09)*. IEEE, 254–265.

WATSON, G., MCKEOWN, N., AND CASADO, M. 2006. NetFPGA: A tool for network research and education. In *Proceedings of Workshop on Architecture Research Using FPGA Platforms (WARFP)*.

WILLMANN, P., BROGIOLI, M., AND PAJ, V. S. 2004. Spinach: A libertybased simulator for programmable network interface architectures. In *Proceedings of the ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04)*. ACM, 20–29.

ZEC, M., MIKUC, M., AND ZAGAR, M. 2002. Estimating the impact of interrupt coalescing delays on steady state TCP throughput. In *Proceedings of the 10th International Conference on Software, Telecommunications and Computer Networks (SoftCOM'02)*. IEEE, *Croatia, Italy*.

ZHAI, E., CUMMINGS, G. D., AND DONG, Y. 2008. Live migration with pass-through device for linux vm. In *Proceedings of Ottawa Linux Symposium (OLS'08)*. Vol 2, 261–268.

Received July 2011; revised October 2011, November 2011; accepted December 2011