RDMA based Replication of Multiprocessor Virtual Machines over High-Performance Interconnects

Balazs Gerofi* and Yutaka Ishikawa*
* *Graduate School of Information Science and Technology*
The University of Tokyo
Tokyo, JAPAN
{bgerofi@il.is.s, ishikawa@is.s}.u-tokyo.ac.jp

***Abstract*—With the growing prevalence of cloud computing and the increasing number of CPU cores in modern processors, symmetric multiprocessing (SMP) Virtual Machines (VM), i.e. virtual machines with multiple virtual CPUs, are gaining significance. However, accommodating SMP virtual machines with high availability at low overhead is still an open problem. Checkpoint-recovery based VM replication is an emerging approach, but it comes with the price of significant performance degradation of the application executed in the VM due to the large amount of state that needs to be synchronized between the primary and the backup machines. Advanced features of high performance interconnects, such as Remote Direct Memory Access (RDMA), on the other hand, offer extreme network throughput. As such feature may provide an opportunity for acceptable performance degradation even for multi-core replicated virtual machines, the impact of such technologies in the domain of VM replication is important to assess. In this paper, we take *a first look at the performance advantages of RDMA for SMP virtual machine replication*. Moreover, in order to alleviate VM downtime during replication, we propose *fine-grained copy-on-write* (COW), which protects only memory pages that need to be transferred to the backup host allowing simultaneous execution of the VM with the replication. We find that the performance of replicated virtual machines over high performance interconnects scales well with the number of vCPUs in multiprocessor virtual machines, and that RDMA based replication in conjunction with fine-grained COW imposes acceptable overhead compared to the native VM execution when applied to virtual machines with up to 16 vCPUs.**

I. Introduction

With the recent increase in cloud computing's prevalence, the number of online services deployed over virtualized infrastructures has experienced a tremendous growth. At the same time, the latest hardware trend of ever growing core number in modern CPUs makes virtual SMP environments, i.e., Virtual Machines (VM) with multiple virtual CPUs increasingly important [16]. Unfortunately, another implication of the growing component number in current computing systems is the fact that hardware failures have become common place rather than exceptional.

Replication at the Virtual Machine Monitor (VMM) layer is an attractive technique to ensure fault tolerance in virtualized environments, primarily, because it provides seamless failover for the entire software stack executed inside the virtual machine, regardless the application or the operating system.

There are currently two main approaches to primary-backup based replication of virtual machines. Log-replay records all input and non-deterministic events of the primary machine so that it can replay them deterministically on the backup node in case the primary machine fails [5], [18]. While this solution provides high efficacy to uni-processor virtual machines, its adaption to multi-core CPU environment is cumbersome, because it requires determining and reproducing the exact order in which CPU cores access the shared memory. It has been shown that this approach imposes superlinear performance degradation with the number of virtual CPUs on several workloads when applied to multi-core VM setups [10].

On the other hand, checkpoint-recovery based replication of virtual machines is attained by capturing the entire execution state of the running VM at relatively high frequency in order to propagate changes to the backup machine almost instantly [7], [8], [15], [23]. This solution, essentially, keeps the backup machine nearly up-to-date with the latest execution state of the primary machine so that the backup can take over the execution in case the primary fails [7].

Between checkpoints the VM executes in log-dirty mode, i.e., write accessed pages are recorded so that when the snapshot is taken only pages that were modified in the most recent execution phase need to be transferred, along with the vCPU context. One phase of dirty logging and transferring the corresponding changes is often called a *replication epoch* [7], [15], [23].

The main strengths of checkpoint-recovery based replication is its inherent ability to tackle with multi-core configurations. However, due to the large amount of state that needs to be synchronized between the primary and the backup machines, the imposed overhead over Gigabit Ethernet is substantial even on uni-processor VM setups. Consequently, there have been no studies so far focusing on checkpoint-recovery based replication of multi-core virtual machines.

On the other hand, high-speed interconnects, such as Infiniband [1] offer features including OS-bypass communication and Remote Direct Memory Access (RDMA). OS-bypass enables communication directly from user-space without the involvement of the underlying operating system, and RDMA allows direct data transfer from the memory of one computer to the other. VM replication can benefit from such facility in

978-0-7695-4516-5/11 $26.00 © 2011 IEEE
DOI 10.1109/CLUSTER.2011.13

several aspects. The very high throughput offered by RDMA can significantly reduce the time needed for transferring dirty pages to the backup machine at the end of each epoch and OS-bypass communication allows the running VM to exploit CPUs more efficiently.

Another obstacle, which stands in front of efficient checkpoint-recovery based replication, is the fact that the VM needs to be suspended at the end of each replication epoch so that a consistent view of its memory contents can be retained while changes are propagated to the backup host. Although dirty memory pages can be collected first in a separate buffer and transferred asynchronously, the imposed VM downtime can be significant with workloads touching memory rapidly [7]. We propose fine-grained copy-on-write (COW), which allows the VM proceeding with its execution while dirty data is collected, but protects only memory pages that correspond to the latest update.

In this paper, we study the feasibility of high performance interconnects in the domain of checkpoint-recovery based replication of SMP virtual machines, enhanced with an efficient copy-on-write mechanism. We make the following contributions:

- A *quantitative analysis of various workloads regarding their behavior patterns over SMP virtual machines* with respect to checkpoint-recovery based replication.
- The design and implementation of virtual machine replication on top of the Linux kernel virtual machine (KVM), with the *integration of Infiniband's Remote Direct Memory Access (RDMA) facility*.
- A *fine-grained copy-on-write* mechanism that eliminates the VM downtime during replication via protecting memory pages, whose value need to be retained so that changes can be transferred to the backup host, and allowing simultaneous execution of the VM.
- Finally, *a comparative performance evaluation between Gigabit Ethernet and Infiniband RDMA based replication* of SMP virtual machines with up to 16 virtual CPUs.

We find that the performance of replicated virtual machines over high performance interconnects scales well with the number of vCPUs in SMP virtual machines. The linux kernel compile workload, for example, suffers only 34% slowdown compared to the native virtual machine's performance when executed on an 8 vCPUs replicated VM.

We begin with characterizing various workloads in terms of memory usage and I/O patterns on SMP virtual machines in Section II. Section III describes the design of our RDMA based replication method with copy-on-write and Section IV provides details on the implementation. Experimental evaluation is given in Section V. Section VI surveys related work, and finally, Section VII presents future plans and concludes the paper.

II. Workloads and Analysis

In this section we describe each workload we investigated, which is then followed by a quantitative analysis regarding their memory usage and I/O patterns on SMP virtual machines.

A. Workloads

Reliable execution may be required by a diverse set of applications, such as long lasting computations or mission critical online services. Inspired by previous studies in the domain of high-availability [7], [10], [15] we chose three different workloads.

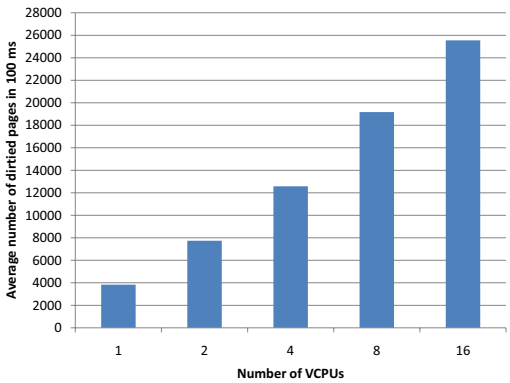
- **Linux Kernel Compile** is an elaborate workload with good scalability over SMP configurations, stressing mainly CPU and memory, but doing a fair amount of disk I/O as well. We compile the *bzImage* target of the vanilla Linux kernel version 2.6.31 with default configuration.
- **Nas Parallel Benchmarks (NPB)** is a collection of computationally intensive parallel applications performing various scientific computations [2]. We chose the OpenMP version of several benchmarks due to their excellent scalability over single node SMP configurations.
- **SPECweb 2005 Banking** emulates an Internet personal banking web-site, where clients are accessing their accounts, making transactions, etc. Requests are transmitted over SSL throughout the whole benchmark [11]. SPECweb is a real world like application and therefore a good candidate for fault tolerance.

B. Analysis

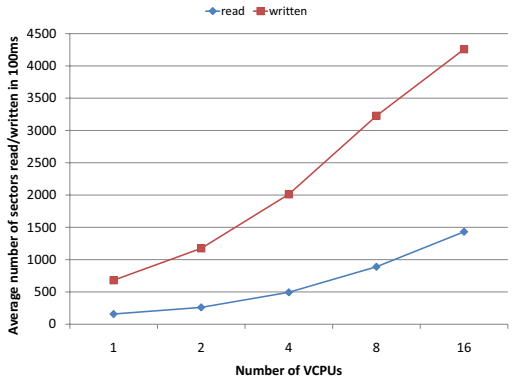
As mentioned earlier checkpoint-recovery based replication of virtual machines is delivered by capturing snapshots of the running VM at relatively high frequency so that changes can be reflected to the backup machine almost instantly. As it will be described in detail in Section III, the two main components that contribute to the overhead of checkpoint-recovery based VM replication are the memory pages dirtied and the sectors written on block devices during the execution phase of each replication epoch. In this section we analyze the chosen workloads from the perspective of memory usage and I/O patterns with respect to the number of virtual CPUs deployed in the virtual machine.

To put the numbers into context, all measurements presented in this Section took place on a 2.4GHz four CPU AMD Opteron ccNUMA machine, with four cores each CPU (i.e. 16 cores altogether) and 8GBs of RAM. The virtual machines had 1GB of RAM and the number of vCPUs will be indicated in the description of each experiment. For further technical details of our experimental framework, see Section V.

Kernel Compile. We carried out measurements of the kernel compile workload on virtual machines configured with 1 to 16 virtual CPUs. The VM executed under log-dirty mode, where we recorded the number of pages written in each 100 milliseconds. Note, that running the VM in log-dirty mode by itself introduces certain overhead due to the fact that the first write to each page causes a page fault in the beginning of every 100 milliseconds epoch. Moreover, at the end of each epoch the dirty-log is reinitialized and all TLB entries have to be invalidated. In Section V we provide exact numbers to what extent the dirty-log mode affects execution time under different setups.



(a) Memory pages dirtied (page size is 4kB).



(b) Block sectors read and written (sector size is 512 bytes).

Fig. 1: **Average number of dirtied pages, block sectors read and written in 100 milliseconds for the kernel compile workload according to the number of virtual CPUs deployed in the VM.**

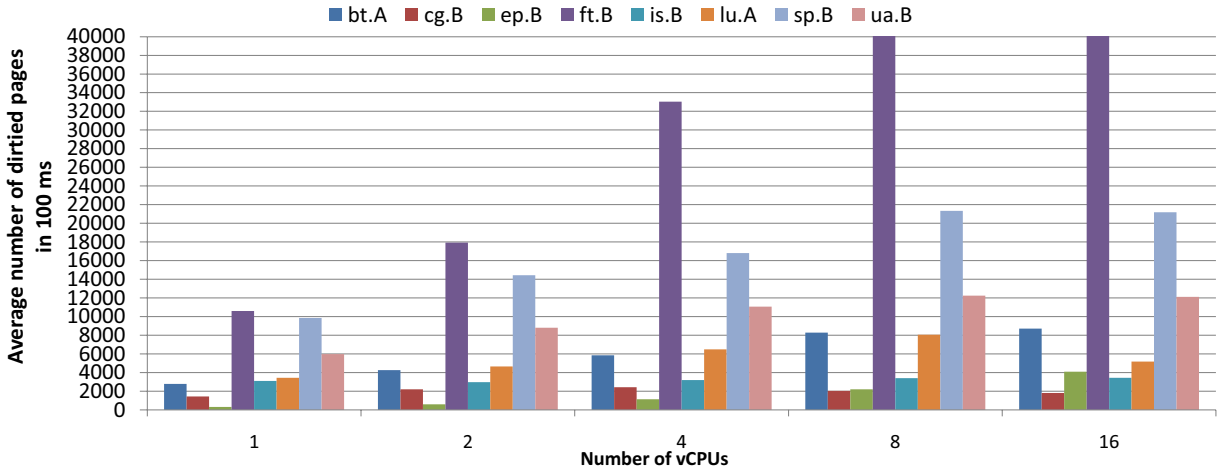


Fig. 2: **Average number of dirtied pages in 100 milliseconds for the NAS Parallel Benchmarks according to the number of virtual CPUs deployed in the VM (page size is 4kB).** The values for ft.B on 8 and 16 CPUs are truncated for clarity. They are 55181 and 46951, respectively.

Figure 1a illustrates the numbers obtained. Kernel compile has excellent scaling properties with the number of vCPUs deployed in the virtual machine. As seen, the average number of dirtied pages in every 100 milliseconds increases steadily with the increasing number of virtual CPUs. Up to 4 CPUs it scales linearly with the CPU number, while the increase weakens somewhat with 8 and 16 CPUs. We believe this is the effect of the underlying ccNUMA architecture and the combination of the log-dirty execution mode. Notice, that a direct consequence from the aspect of virtual machine replication is the increasing need of network bandwidth so that the replication period can be maintained. Taking other components of the replication data (such as disk I/O), a socket based solution over Gigabit Ethernet can incur significant performance degradation due to its insufficient network bandwidth.

Another observation is the amount of block I/O involved in the kernel compile workload. Figure 1b depicts the average number of sectors read and written in 100 milliseconds. The kernel compile workload, again, scales well and shows steady increase in the number of I/O operations with the increasing number of virtual CPUs. The key observation here is the increase in the number of sectors written, because disk modifications also need to be transferred to the backup machine.

NAS Parallel Benchmarks. We performed the same set of experiments for various applications from the NPB benchmarks. We used their OpenMP version and set the $OMP\_NUM\_THREADS$ environment variable in the VM to the number of virtual CPUs. Figure 2 shows the results obtained from these experiments. There are several interesting observations regarding NPB. First, note, that all these applications scale well with the number of CPUs in terms of execution

time (see Section V), but this does not necessary imply an increase in the number of dirtied pages. Second, it is apparent that depending on the algorithm the number of pages dirtied shows rather big differences. For instance, FT and SP touch significantly more pages than the other benchmarks among the ones we investigated.

However, we were more interested in assessing the memory behavior as the function of the number of CPUs utilized. While BT, FT, SP and UA exhibit obviously growing demand in terms of dirtied pages when the number of CPUs increases, CG and IS for example touch approximately the same amount of pages regardless the CPU number in the system. As one could expect, memory behavior with respect to the number of CPUs visibly varies depending on the problem under consideration.

It is also worth noting that even the most scalable application (in terms of dirtied memory), such as FT or SP, show a decline in the number of pages when the number of CPUs lifts from 8 to 16. Again, we believe this is the effect of the log-dirty mode and the underlying ccNUMA architecture of the host machine.

SPECweb2005 Banking. Similarly to the previously discussed workloads, we carried out the same set of experiments for SPECweb Banking. SPECweb reports two values for a run, the percentage of tolerable and good answers it harnesses from the server during the test period. First, we tuned the SPECweb config file to obtain the highest number of simultaneous sessions that gives over 97% good and 100% tolerable answers on the native VM with one virtual CPU. Then, we started adding more virtual CPUs and increased the number of simultaneous sessions to see if we get better results. Our observation was that on this particular setup the bottleneck of the SPECweb benchmark was memory rather than computing power.

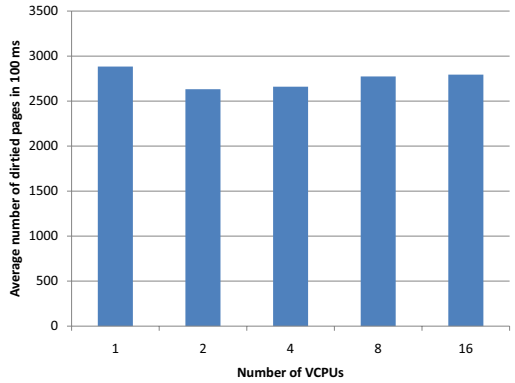


Fig. 3: **Average number of dirtied pages in 100 milliseconds for the SPECweb2005 Banking workload according to the number of virtual CPUs deployed in the VM.**

This is, in fact, reflected by Figure 3, which shows no increase in the number of dirtied pages with the increasing number of virtual CPUs in the VM. As for the I/O operations of SPECweb2005 we obtained similar results, the number of I/O write operations is relatively low, around 90 sectors per 100 milliseconds and it does not change with the number of CPUs.

However, SPECweb2005 is still a good candidate for RDMA based replication, due to the replication's consistency protocol which needs to hold back outside visible events, such as network packets, until the backup machine acknowledges the corresponding update of the given replication epoch. This mechanism will be discussed in detail in Section III.

III. RDMA based Virtual Machine Replication with Copy-On-Write

In this section we first describe the mechanism of checkpoint-recovery based virtual machine replication, following with the discussion of copy-on-write and Infiniband's RDMA integration.

As we have mentioned, checkpoint-recovery based replication of virtual machines is attained by capturing the entire execution state of the running VM at high frequency in order to reflect the changes to the backup machine almost instantly. Between checkpoints the VM executes in log-dirty mode, i.e., write accessed pages are tracked so that when the snapshot is taken only pages that were modified in the most recent execution phase need to be transferred. In order to reduce the overhead of transferring dirty pages, replication data can be buffered and transferred asynchronously, overlapping the VM's execution in the subsequent epoch [7].

However, any fault tolerant system needs to ensure that the state from which an output message is sent will be recovered despite any future failure, which is commonly referred to as the *output commit* problem [19]. As a consequence of such requirement, during the execution phase of each epoch, output of the running VM needs to be held back, i.e., disk I/O and network traffic have to be buffered and can be released only after the backup machine acknowledged the corresponding update [7], [8], [15].

A. Fine-Grained Copy-On-Write

It is important to point out, that even in case of asynchronous data transfer, it has to be ensured that the update transferred to the backup machine holds a consistent view of the memory corresponding to the given replication epoch. One possible solution, introduced in *Remus* [7], is to suspended the VM, collect the changes into a separate buffer, and resume the VM before beginning the actual data transfer.

As we showed in Section II-B, several workloads touch an increasing number of dirty pages during the same period of time when the number of virtual CPUs is increased in SMP virtual machines. In Section V we will also show that copying this big amount of data into a separate buffer by itself takes a significant amount of time. In order to alleviate this downtime, we propose fine-grained copy-on-write, which works as the followings. When the VM is suspended at the end of a replication epoch so that the dirty page map is obtained, instead of simply reinitializing the dirty map, it is preserved by the VMM. During the next execution phase, each time a write page-fault occurs (the VM runs in dirty-log mode), the old

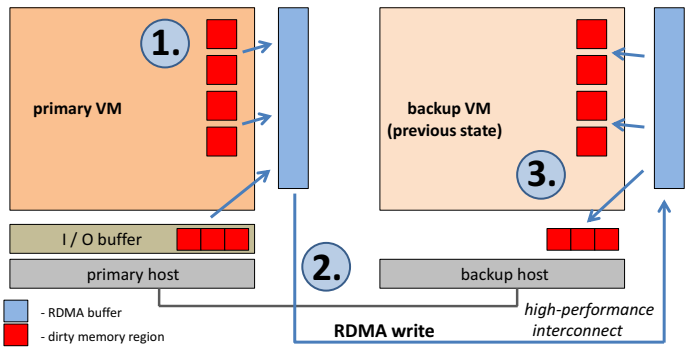


Fig. 4: **RDMA based VM replication with copy-on-write, high-level design.** *Three main steps of updating the backup VM during a replication epoch: 1.) Pause the VM, enable COW, resume the VM, copy dirty pages and dirty disk sectors into a buffer, and disable COW. 2.) Transfer the update to the backup machine via RDMA. 3.) Apply the changes to the backup VM's memory.*

bitmap is first consulted. If the write refers to a page present in the old dirty bitmap, then a copy of the original page is retained.

TABLE I: Ratio of dirty pages necessary to COW.

Workload	Kernel Comp.	ep.B	sp.B	ft.B	SPECweb
Ratio	41%	56%	30%	12%	40%

Notice, that pages which do not appear in the old dirty bitmap do not need to be COW protected, because they are not part of the update. Table I shows the ratio of pages which was necessary to be copied for some of the workloads, when executed over 8 vCPUs. As seen, the ratio scales between 12% and 56%, allowing the fine-grained COW mechanism to save significant amount of work via not copying unnecessary pages.

Our proposed mechanism lets the VM to be resumed immediately after the dirty bitmap is obtained and the VCPU context is captured. The replication engine, in turn, uses the old content of the pages during the data transfer.

B. RDMA

With workloads that touch memory rapidly, the time required to propagate changes at the end of an epoch may exceed the replication period itself, leading to substantial overhead, and causing significant performance degradation to the application, even if dirty content is transferred asynchronously [7]. This anomaly becomes rather severe in case the application is latency sensitive, such as several online services [11].

We alleviate the overhead of transferring the increased amount of data with the utilization of Infiniband's RDMA facility. Notice, however, that contrary to virtual machine migration [12], where the new content of each dirty page can be directly transferred to the corresponding remote address of the target machine, in case of VM replication the backup needs to buffer the updates first, otherwise, a failure during the data transfer would leave it in an inconsistent state. This implies that at the end of each replication epoch the backup machine needs to apply all changes together in a transactional fashion, only if all data were received successfully. For this reason, we maintain a large continuous buffer on the backup machine which can be accessed via RDMA from the primary host.

Before executing any RDMA operations, it is required that the target machine, in our case the backup host, registers the memory buffer where data will be transferred to and informs the primary machine of the buffer's remote key returned from the registration. We exchange this remote key in the initialization phase of the replication.

The high level design of the RDMA based VM replication with copy-on-write is shown in Figure 4. At the end of each replication epoch, the primary machine suspends the VM, saves the VCPU context, enables COW and resumes the VM immediately. In parallel with the next epoch's execution it collects all changes in a buffer. Once the changes are in place, COW can be disabled and the VM continues running in dirty-log mode. The primary machine does a handshake with the backup to ensure the buffer on the remote side is ready for receiving the updates. An RDMA write is then issued and all data are transferred to the backup machine. Once the transfer finished, buffered disk sectors and network packets can be released on the primary machine. Finally, the backup machine updates the VM's memory content and commits disk I/O.

Our current implementation uses a static buffer for the purpose of collecting updates due to better utilization of the network bandwidth during the RDMA transfer [12]. Otherwise, for each page a separate RDMA write would have to be issued. In the future we plan to improve this with dynamic buffer management.

IV. Implementation

This section discusses technical issues regarding our implementation.

A. KVM

We chose the Linux Kernel Virtual Machine (KVM) [13] as the platform of this study. KVM takes advantage of the hardware virtualization extensions so that it achieves nearly the same performance with the underlying physical machine.

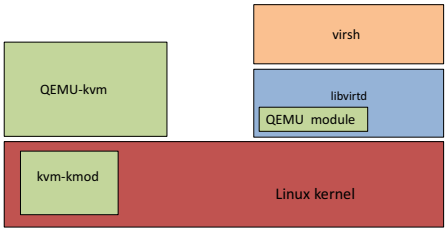


Fig. 5: **The Linux kernel virtual machine architecture.**

Figure 5 depicts the high-level KVM architecture. The most important components are the *kvm* kernel module and *qemu-kvm*, a KVM tailored version of QEMU. On top of these, *libvirtd* is an often used facility for managing virtual machines, for which *virsh* provides a command line interface. A major advantage of the KVM architecture is the full availability of user-space tools in the QEMU process, such as threading, libraries and so on.

B. Replication Logic and I/O Buffering

The replication logic is entirely implemented in *qemu-kvm*, leveraging a great amount of the live migration code.

For disk I/O and network buffering we modified the virtio drivers of *qemu-kvm*. The disk I/O buffer behaves also as a hash table that operates on sector granularity so that read requests referring to sectors which are already buffered can be accessed consistently. As for network buffering we maintain an extra packet queue that captures outgoing packets during the execution phase of a replication epoch. Once the backup machine acknowledges the update both disk and network buffers are committed.

C. Transactional Updates

We have mentioned the transactional nature of updating the backup machine in Section III. Unfortunately, the network protocol of *qemu-kvm*'s live migration code doesn't support this by default.

For this reason we extended the *QEMUFile* object with a buffer and a flag that indicates that the file is in buffered mode. The primary machine toggles this flag on the file corresponding to the backup connection and all subsequent writes are first buffered. The backup machine, on the other hand, associates the replication receive buffer to the *QEMUFile* object referred in the VM state loaders. It then toggles the file's flag to indicate that subsequent read operations issued by *qemu-kvm* should access the buffer instead of trying to receive data from the network.

D. Copy-On-Write

On the lowest level, we extended the KVM kernel module to perform copy-on-write when it's requested by *qemu-kvm*. Copy-on-write is a well applied technique in operating systems, particularly for enforcing private access to an otherwise shared memory area among separate address spaces. However, in our case, COW is not as straightforward as it is with regular processes, because the replication mechanism and the running VM actually share the same address space. When a page is written and COWed, the VM still needs to access the most recent content, while the replication engine should see the previous epoch's value. In order to meet both requirements we copy the old content of the page to another address and maintain a translation table, which is queried by the replication engine to find out whether or not a page has been COWed. Note, that COW pages are recycled in each epoch after COW is disabled.

E. InfiniBand RDMA

We implement the RDMA transfer through OpenFabrics' native Infiniband verbs API [3]. The native InfiniBand verbs form the lowest software layer for the IB network, and allow direct user-level access from the *qemu-kvm* process to the IB host channel adaptor (HCA) resources while bypassing the operating system. At the IB verbs layer, we used the queue pair model for supporting channel-based communication semantics during the handshake between the primary and backup machines, as well as for memory-based communication semantics, i.e., RDMA.

V. Evaluation

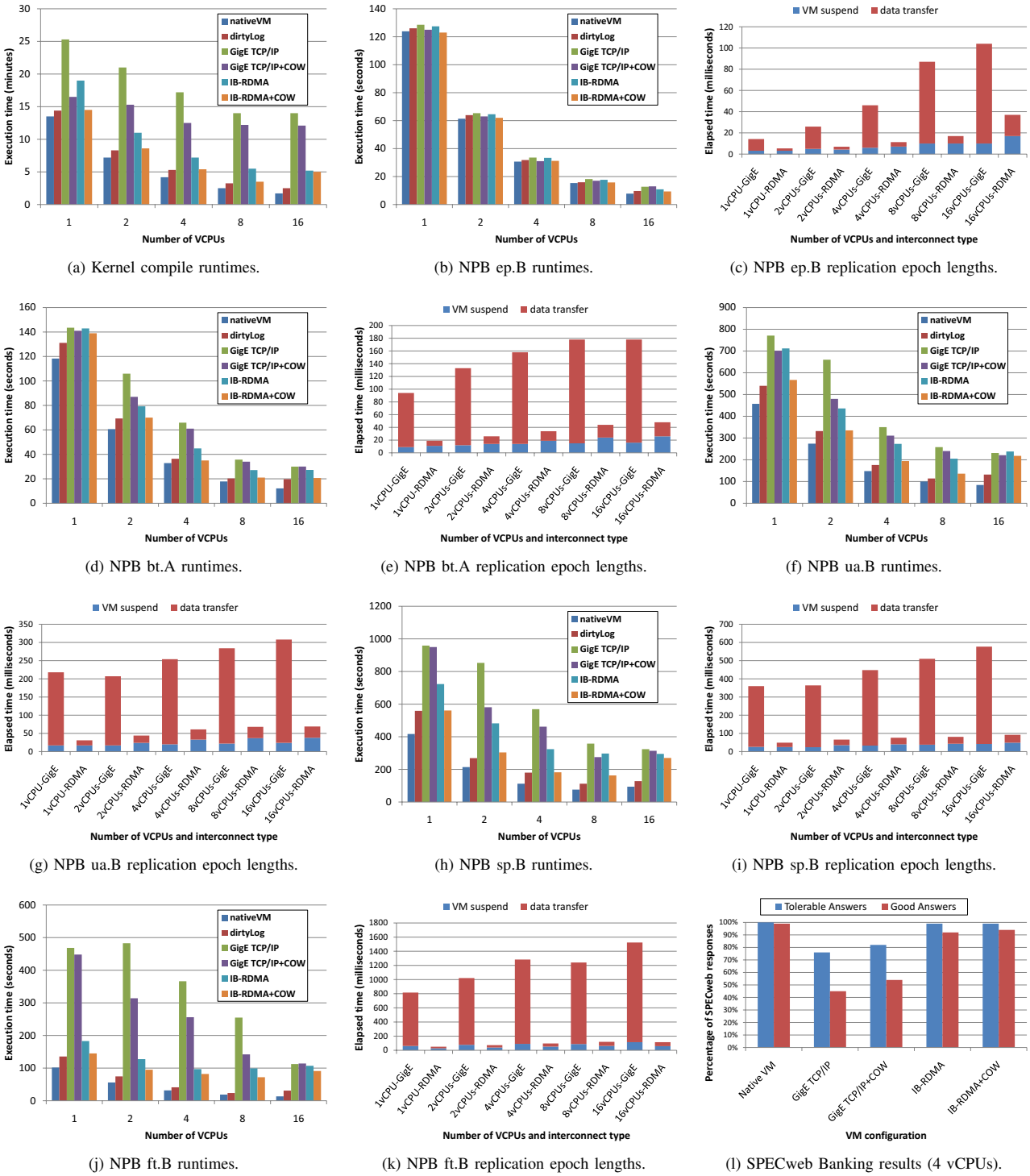
A. Experimental Setup

Throughout our experiments the host machine of the replicated VM was a 2.4GHz four CPU AMD Opteron ccNUMA machine, with four cores each CPU (i.e. 16 cores altogether), 8GBs of RAM and a 250GB SATA harddrive. The machine was equipped with two Intel 82546GB Gigabit Ethernet network interfaces. One of the physical network cards were bridged to the virtual machine and used for application traffic and the other was dedicated to the replication protocol for the experiments, when replication took place over Gigabit Ethernet. Moreover, a Mellanox MT26428 Infiniband QDR HCA was also present in both the primary and the backup hosts for the experiments utilizing RDMA.

The host machines run Ubuntu server 9.10 on Linux kernel 2.6.37 and we used qemu-kvm 0.14.50 with kvm-kmod 2.6.37 as the basis of our implementation. For the virtual machines in each experiment we used the KVM virtio disk and network drivers. We do not present performance results on the native host machine, because in virtualized environments direct access to the underlying machines is normally not available. However, it is worth noting that in all experiments we had AMD's hardware MMU virtualization support, i.e. Nested Page Tables (NPT) enabled. Unless stated otherwise, the VM had 1 GB of RAM allocated.

B. Kernel Compilation

Our first target is the kernel compilation workload. In this experiment we compile the *bzImage* target of Linux kernel version 2.6.31 using the default configuration. We repeated each experiment three times for each VM setup and report the average wall-clock time measured.



(a) Kernel compile runtimes.

(b) NPB ep.B runtimes.

(c) NPB ep.B replication epoch lengths.

(d) NPB bt.A runtimes.

(e) NPB bt.A replication epoch lengths.

(f) NPB ua.B runtimes.

(g) NPB ua.B replication epoch lengths.

(h) NPB sp.B runtimes.

(i) NPB sp.B replication epoch lengths.

(j) NPB ft.B runtimes.

(k) NPB ft.B replication epoch lengths.

(l) SPECweb Banking results (4 vCPUs).

Fig. 6: **Performance results of various applications executed on native VM, dirty-log VM, GigE TCP/IP, and Infiniband RDMA based replication, with and without COW, according to the number of vCPUs in the system.**

Figure 6a illustrates the execution times on the native virtual machine, the VM running in log-dirty mode, replicated over Gigabit Ethernet and Infiniband RDMA, with and without COW enabled. Similarly to related work [7], we set the replication period in this experiment to 100 milliseconds so that failover from the user's point of view remains entirely transparent. As seen, having the VM run in log-dirty mode by itself imposes a certain level of performance degradation, however, this is inevitable in case of checkpoint-recovery based replication.

Figure 1a indicated previously, that the kernel compile workload exhibits a growing demand in terms of dirtied memory and the number of I/O operations when executed over multiple CPUs. Consequently, as seen on Figure 6a, RDMA based replication attains an increasing performance improvement compared to Gigabit Ethernet with the growing number of CPUs in the system. The key observation, however, is the combination of RDMA and copy-on-write that achieves nearly the same performance with the dirty-log mode up to 8 vCPUs. Again, since dirty-log mode is unavoidable, it is impossible for the replication to achieve higher performance than that. As for 16 vCPUs, the kernel compilation completes almost 3 times faster on a VM replicated over Infiniband as opposed to GigE, regardless whether or not COW is enabled. We believe this is due to the necessary synchronization during COW, which introduces a growing overhead with the increasing number of vCPUs in the system. Compared to the native VM performance, replication over RDMA with COW suffers 7%, 13%, 29%, 34%, and 2.94 times slowdown in case of 1, 2, 4, 8 and 16 vCPUs, respectively.

C. NAS Parallel Benchmarks

The NAS Parallel Benchmarks revealed several interesting properties of VM replication with respect to the performance attained over different network interconnects. For the sequence, in which we discuss the various benchmarks from the NPB set, we follow the order of growing number of dirtied pages presented in Figure 2. For all experiments we set 100 milliseconds as the intended replication period. We measure each benchmark on the native VM, VM executing in log-dirty mode and on replicated VMs over Gigabit Ethernet and Infiniband RDMA, with and without copy-on-write.

As seen previously on Figure 2, the ep.B benchmark touches relatively low amount of memory regardless the number of CPUs. Figure 6b illustrates the runtimes for ep.B. Due to the small amount of changes that need to be synchronized, even the bandwidth of Gigabit Ethernet is sufficient for keeping the backup machine up-to-date according to the replication period. Consequently, the results show very similar performance degradation in case of both GigE and Infiniband, regardless COW is enabled or not. This phenomenon is further verified by Figure 6c, which shows the average duration of the VM suspend and the data transfer phase per replication epoch, when COW is disabled. As seen, while GigE spends an increasing amount of time on transferring data to the backup machine, both GigE and Infiniband completes the transfer within close to 100 milliseconds regardless the vCPU number.

This observation, however, does not stand for bt.A, where data transfer time (see Figure 6e) exceeds 100 milliseconds in case of 2 virtual CPUs and above. Figure 6d shows clear improvements of the RDMA replication method compared to GigE for this benchmark.

Continuing our investigation, we now focus on the ua.B benchmark. In Figure 2, we saw that ua.B touches a growing amount of memory with the increasing number of CPUs in the virtual machine. Although a similar effect could be already observed on the kernel compile workload, as seen on Figure 6f, ua.B achieves slightly better performance over GigE with COW enabled than RDMA without COW on 1 vCPU. One of the reasons for this, besides that COW enables more efficient execution, is the fact that the prolonged data transfer phase (see Figure 6g) in case of GigE renders the frequency of checkpoints lower than that in case of RDMA. Nevertheless, when RDMA is combined with COW it achieves substantially better performance than GigE. Again, RDMA and COW shows close to dirty-log mode performance up to 8 vCPUs.

The two heaviest benchmarks we encountered were sp.B and ft.B. As shown previously in Figure 2, both sp.B and ft.B touch substantially higher number of pages than other benchmarks on the same VM setup. Despite the fact that the data transfer phase lasts considerably longer over GigE for sp.B and ft.B (Figure 6i and Figure 6k, respectively), rendering the number of checkpoints taken lower than in case of RDMA, due to the large amount of data that needs to be transferred, RDMA based replication shows significantly better performance than GigE (illustrated by Figure 6h and Figure 6j). Although, in case of ft.B, even RDMA with COW is unable to converge to the performance of dirty-log mode when run over 4 vCPUs and suffers a near 4 times slowdown at 8 vCPUs already.

D. SPECweb

The last application we investigate is SPECweb's Banking workload. The SPECweb configuration requires at least three machines for running the experiments [11]. One of the server hosts is the actual SPECweb application server, which is accompanied by a backend machine. These were deployed in two VMs residing on two separate physical machines. Besides these, a desktop machine was utilized for running the SPECweb client side scripts.

We replicate only the main SPECweb application server and ran five different setups: native VM, replication over GigE and Infiniband, with and without COW for each. As mentioned in Section II-B, we did not observe any significant change in the amount of dirtied pages with the increasing number of CPUs for this particular benchmark, and all results reported here were obtained on a VM configured to have 4 virtual CPUs. As we discussed before, network I/O must be held back during the execution phase of each replication epoch and can be released only after the backup machine acknowledges the corresponding update. Since RDMA transfers the dirty data in

much shorter period of time, for a latency sensitive workload, such as SPECweb, substantial performance improvements can be expected. On the other hand, presumably COW doesn't play a significant role in this case, because network I/O can be released only after the update is acknowledged regardless COW is enabled or not.

We obtain our measurements by first tuning the SPECweb configuration so that 99% of the responses are categorized as "good" when executed on the native VM. Both the Gigabit Ethernet and the Infiniband RDMA based replicated VM setups were then measured with the same configuration and we compare the average percentage of "good" and "tolerable" responses reported by the SPECweb client script. The replication period is set to 50 milliseconds in these experiments, because SPECweb is more sensitive to network latency. Note, that changing the replication period doesn't affect the fairness of the comparison itself, because the same epoch length is used in all setups.

Figure 6l shows the results we obtained from these experiments. As seen, the performance attained by Gigabit Ethernet based replication is low, 45% and 76% without COW, 54% and 82% with COW, for "good" and for "tolerable" answers, respectively. This degradation is alleviated by the RDMA based replication, which achieves 92% and 98% without COW, 94% and 99% with COW, for "good" and "tolerable" answers, respectively.

VI. Related Work

A. Virtual Machine Migration

Checkpoint-recovery based fault tolerance captures snapshots of the running VM at high frequency, often leveraging the live migration support of the underlying Virtual Machine Monitor (VMM). Thus, VM live migration is closely related to checkpoint-recovery based replication. Solutions, such as *Xen* [6], *KVM* [13], and *VMware's VMotion* [17] all provide the capability of live migrating VM instances. Pre-copy is the dominant approach to live VM migration [6], [17]. It initially transfers all memory pages then tracks and transfers dirty pages in subsequent iterations. When the amount of data transferred becomes small or the maximum number of iteration reached, the VM is suspended and finally, the remaining dirty pages and the VCPU context is moved to the destination machine. VM replication, on the other hand, leaves the VM running in pre-copy mode at all times so that dirty pages are logged and the entire execution state can be reflected to the backup node at the end of each replication epoch [7], [8].

Performance improvement to VM migration has been the focus of several prior studies. Xian et al. showed how data deduplication can be exploited to accelerate live migration [22], while *Microwiper* [9] proposed ordered propagation of dirty pages to transfer them according to their rewriting rates, reducing service downtime during the migration.

High performance interconnects have also been used in the context of virtual machine migration, Huang et al. presented RDMA based migration over Infiniband [12], Note however, that they only consider uni-processor VMs, besides, VM replication involves various different technical issues which distinguishes our work from this study,

B. Virtual Machine Replication

Bressoud and Schneider [5] introduced first the idea of hypervisor-based fault tolerance by executing the primary and the backup VMs in lockstep mode, i.e., logging all input and non-deterministic events of the primary machine and having them deterministically replayed on the backup node in case of failure. While Bressoud and Schneider demonstrated this technique only for the HP PA-RISC processors VMware's recent work implements the same approach for x86 architecture [18]. These works, however, can handle only uni-processor environments. Deterministic-replay imposes strict restrictions on the underlying architecture and its adaption to multi-core CPU environment is cumbersome, because it requires determining and reproducing the exact order in which CPU cores access the shared memory.

In the context of deterministic (i.e. replayable) SMP execution, solutions on different abstraction levels have been proposed. *Flight Data Recorder* [21] is a hardware extension that enables deterministic replay for SMP environments, but it is unclear what degree of concurrency they can handle without significant performance degradation. Runtime system level solutions, such as *Respec* [14] and *CoreDet* [4] ensure deterministic execution of multi-threaded applications, but their main weakness compared to VM level solutions is the inability to provide fault tolerance for an entire software stack (including the operating system), which is encompassed by a virtual machine. *SMP-ReVirt* [10] exploits hardware page protection to detect and accurately replay sharing between virtual CPUs of a multi-core virtual machine, however, their experiments report superlinear slowdown with the increasing number of virtual CPUs.

Checkpoint-recovery based solutions such as *Remus* [7] and *Paratus* [8] can overcome the problem of multi-core execution by capturing the entire executions state of the VM and transferring it to the backup machine. Although most of the data transfer can be overlapped with speculative execution, transferring updates to the backup machine at very high frequency still comes with great performance overhead. *Kemari* [20] follows a similar approach to *Remus*, but instead of buffering output during speculative execution, it updates the backup machine each time before the VM omits an outside visible event.

Improving the performance of checkpoint-recovery based VM replication has become an active research area recently. Lu et al. [15] proposed fine-grained dirty region identification to reduce the amount of data transferred during each replication epoch, while Zhu et al. [23] improved the performance of log-dirty execution mode by reducing read- and predicting write-page faults. All the above mentioned studies in the domain of checkpoint-recovery based VM replication, however, deal only with uni-processor environments. By contrast, we focus on multi-core virtual machines in conjunction with high-performance interconnects.

VII. Conclusions and Future Work

Checkpoint-recovery based virtual machine replication is attractive, it provides high availability for the entire software stack executed in the VM, and it is inherently capable of dealing with SMP configurations. However, it comes with great overhead due to the large amount of state that needs to be synchronized frequently between the primary and the backup hosts.

In this paper we have analyzed various workloads from their memory usage and I/O patterns, focusing particularly on their behavior as the function of the increasing number of CPUs in SMP virtual machines. We found that the number of pages and the number of disk sectors written in a unit period of time vary widely with the workloads under consideration. Nevertheless, they often increase rapidly with the number of CPUs in the system, imposing a growing demand for the network bandwidth available for replication.

In order to eliminate VM downtime during the replication we have proposed fine-grained copy-on-write, that retains the original values only for pages that belong to the given update, allowing the VM to proceed with its execution simultaneously with the replication mechanism. To alleviate the overhead of frequent synchronization, we have designed and implemented VM replication over Infiniband, utilizing OS-bypass communication and Remote Direct Memory Access. Our experiments showed that checkpoint-recovery based replication over RDMA in conjunction with fine-grained COW offers scalable fault-tolerance when applied to SMP virtual machines. For example, the kernel compile workload on an RDMA replicated VM with 8 vCPUs shows only 34% slowdown compared to the native VM's performance, while SPECweb's Banking workload achieves around 94% of the native score.

We have mentioned the potential impacts of the checkpoint frequency on the achievable performance. Long running computations without I/O would benefit from longer replication epochs (i.e., lower checkpoint frequency), while those, sensitive to network latency, require short epochs so that network I/O can be released as soon as possible. In the future, we intend to investigate how the optimal length of replication epoch could be determined automatically, based on application characteristics, such as memory usage and I/O patterns.

Acknowledgment

This work has been supported by the CREST project of the Japan Science and Technology Agency (JST).

References

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

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

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

[4] Bergan, T., Anderson, O., Devietti, J., Ceze, L., and Grossman, D. CoreDet: a compiler and runtime system for deterministic multithreaded execution. In *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems* (2010), ASPLOS '10, ACM, pp. 53–64.

[5] Bressoud, T., and Schneider, F. B. Hypervisor-based fault tolerance. In *Proceedings of the fifteenth ACM symposium on Operating systems principles* (New York, NY, USA, 1995), SOSP '95, ACM, pp. 1–11.

[6] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. Live Migration of Virtual Machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation* (Berkeley, CA, USA, 2005), USENIX Association, pp. 273–286.

[7] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., and Warfield, A. Remus: high availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (2008), NSDI'08, pp. 161–174.

[8] Du, Y., and Yu, H. Paratus: Instantaneous Failover via Virtual Machine Replication. In *Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing* (2009), GCC '09, IEEE Computer Society, pp. 307–312.

[9] Du, Y., Yu, H., Shi, G., Chen, J., and Zheng, W. Microwiper: Efficient Memory Propagation in Live Migration of Virtual Machines. In *Proceedings of the 2010 39th International Conference on Parallel Processing* (Washington, DC, USA, 2010), ICPP '10, IEEE Computer Society, pp. 141–149.

[10] Dunlap, G. W., Lucchetti, D. G., Fetterman, M. A., and Chen, P. M. Execution replay of multiprocessor virtual machines. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2008), VEE '08, pp. 121–130.

[11] Hariharan, R., and Sun, N. Workload Characterization of SPECweb2005. http://www.spec.org/workshops/2006/papers/02_Workload_char_SPECweb2005_Final.pdf, 2006.

[12] Huang, W., Gao, Q., Liu, J., and Panda, D. K. High performance virtual machine migration with RDMA over modern interconnects. In *Proceedings of the 2007 IEEE International Conference on Cluster Computing* (Washington, DC, USA, 2007), CLUSTER '07, IEEE Computer Society, pp. 11–20.

[13] Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. kvm: the Linux virtual machine monitor. In *Ottawa Linux Symposium* (July 2007), pp. 225–230.

[14] Lee, D., Wester, B., Veeraraghavan, K., Narayanasamy, S., Chen, P. M., and Flinn, J. Respec: efficient online multiprocessor replayvia speculation and external determinism. ASPLOS '10, ACM, pp. 77–90.

[15] Lu, M., and cker Chiueh, T. Fast memory state synchronization for virtualization-based fault tolerance. In *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on* (2009), pp. 534 –543.

[16] McDougall, R., and Anderson, J. Virtualization performance: perspectives and challenges ahead. *SIGOPS Oper. Syst. Rev. 44* (December 2010), 40–56.

[17] Nelson, M., Lim, B. H., and Hutchins, G. Fast transparent migration for virtual machines. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), USENIX Association, p. 25.

[18] Scales, D. J., Nelson, M., and Venkitachalam, G. The design of a practical system for fault-tolerant virtual machines. *SIGOPS Oper. Syst. Rev. 44* (December 2010), 30–39.

[19] Strom, R.E. and Bacon, D.F. and Yemini, S.A. Volatile logging in n-fault-tolerant distributed systems. In *Fault-Tolerant Computing, Eighteenth International Symposium on* (Jun 1988), pp. 44 –49.

[20] Tamura, Y. Kemari: Virtual Machine Synchronization for Fault Tolerance using DomT. Technical report, NTT Cyber Space Labs, 2008.

[21] Xu, M., Bodik, R., and Hill, M. D. A "flight data recorder" for enabling full-system multiprocessor deterministic replay. In *Proceedings of the 30th annual international symposium on Computer architecture* (2003), ISCA '03, ACM, pp. 122–135.

[22] Zhang, X., Huo, Z., Ma, J., and Meng, D. Exploiting Data Deduplication to Accelerate Live Virtual Machine Migration. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on* (2010), pp. 88 –96.

[23] Zhu, J., Dong, W., Jiang, Z., Shi, X., Xiao, Z., and Li, X. Improving the Performance of Hypervisor-based Fault Tolerance. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on* (2010), pp. 1 –10.