2011 17th IEEE Pacific Rim International Symposium on Dependable Computing

Workload Adaptive Checkpoint Scheduling of Virtual Machine Replication

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*—Checkpoint-recovery based Virtual Machine (VM) replication is an emerging approach towards accommodating VM installations with high availability, especially, due to its inherent capability of tackling with symmetric multiprocessing (SMP) virtual machines, i.e. VMs with multiple virtual CPUs (vCPUs). However, it comes with the price of significant performance degradation of the application executed in the VM because of the large amount of state that needs to be synchronized between the primary and the backup machines. Previous research improving VM replication performance focused primarily on decreasing the amount of data transferred over the network, while relying on constant checkpoint frequency. Our goal is to investigate how and to what extent performance degradation can be mitigated by adjusting the checkpoint period dynamically. We provide a comprehensive analysis of various workloads from the aspect of VM replication, paying special attention to their behavior over the increasing number of vCPUs in the system. We propose several heuristics for scheduling replication checkpoints in order to improve quality of service. Our algorithm adapts dynamically to the properties of the workload being executed in the VM, such as changes in the number of dirtied memory pages, network and disk I/O operations, as well as to the network bandwidth available for replication. We evaluate our scheduling algorithm over two network architectures, Gigabit Ethernet and Infiniband, a high-performance interconnect fabric. We find that checkpoint scheduling has a great impact on the performance of replicated virtual machines, and show that replicated virtual machines with up to 16 vCPUs can attain performance close to the native VM execution, not only over high-performance, but also over commercial network architectures.**

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 [1]. 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 underlying 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 [2], [3]. 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 [4].

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 [5], [6], [7], [8]. 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 [5].

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* [5], [7], [8].

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 [9]. 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 [5], [6], [7].

One of 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 overhead of replication can be substantial even on uni-processor VM setups. Several recent studies have explored the idea of accelerating the failure free period

978-0-7695-4590-5/11 $26.00 © 2011 IEEE
DOI 10.1109/PRDC.2011.32

of replicated virtual machines, considering mainly how to decrease the amount of data transferred during synchronization [7], [8]. Nevertheless, they all rely on constant checkpoint frequency.

Different workloads, however, can exhibit considerably different behavior in terms of the number of dirtied memory pages, the number of transmitted network packets, and the number of disk I/O operations. Moreover, these attributes often change significantly in time and also with the number of vCPUs in the system. When and how often checkpoints should be taken is not evident, furthermore, it has a great influence on the performance of the replicated virtual machine. For instance, a long lasting computation without I/O would benefit from less frequent checkpoints, while a workload, sensitive to network latency, should be checkpointed frequently so that buffered network packets can be released as soon as possible.

In this paper we propose several heuristics for scheduling checkpoints dynamically according to the attributes of the workload being executed in the replicated VM. 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.
- A *checkpoint scheduling algorithm*, which adapts dynamically to the properties of the given workload, such as *the number of dirtied memory pages, the number of pending network packets, and the number of disk I/O operations* as well as to *the network bandwidth available for replication*.
- Finally, a *fine-grained copy-on-write* mechanism that eliminates the VM downtime during the checkpoints via protecting only those memory pages, whose value need to be retained so that changes can be transferred to the backup host, while allowing concurrent execution of the virtual machine.

We find that the price of highly available SMP virtual machines can be relatively modest when checkpoints are scheduled carefully, and close to native performance can be achieved not only over high-speed interconnects, but also with replication over a commercial network architecture.

We begin with characterizing various workloads in terms of memory usage and I/O patterns on SMP virtual machines in Section II. Section III gives background information on VM replication along with the description of the copy-on-write mechanism. Section IV introduces our scheduling heuristics and Section V provides details on the implementation. Experimental evaluation is given in Section VI. Section VII surveys related work, and finally, Section VIII 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

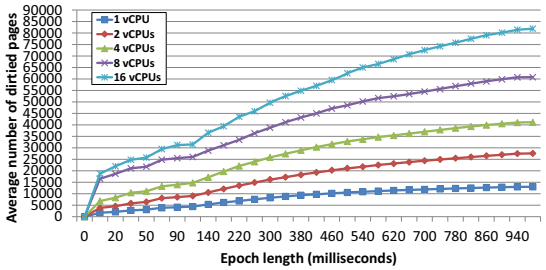
Reliable execution may be required by a diverse set of applications, such as long lasting computations or mission critical online services. We chose the following four workloads.

- **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 [10]. We chose the OpenMP version of two NPB benchmarks in order to assess the scalability of our replication mechanism over multiple vCPUs.
- **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.
- **Hadoop** [12] is an open-source implementation of the MapReduce (MR) [13] distributed data analysis tool, an emerging framework often used in cloud computing environments. A Hadoop MR cluster consists of several worker nodes (called *TaskTrackers*) that are coordinated by a central entity, the *JobTracker*. While *TaskTrackers* may fail any time without bringing the entire job down, the failure of the *JobTracker* is fatal. Our Hadoop cluster is composed of 16 worker nodes (i.e., TaskTrackers) and a separate VM which hosts the JobTracker. We evaluate the price of protecting the Hadoop master node via VM level replication through the MR/DB benchmark set [14]. MR/DB operates on random generated HTML data and it includes various common tasks that can be expressed either as SQL queries or as MapReduce computations, such as grep, select, aggregate and join.

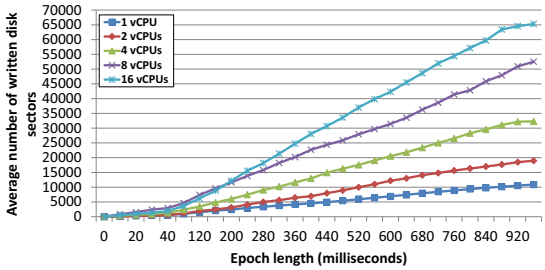
B. Analysis

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



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



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

Fig. 1: **Kernel Compile memory and disk I/O behavior.** Average number of dirtied pages and block sectors written as the function of replication epoch length and the number of virtual CPUs deployed in the VM.

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 VI.

1) Kernel Compile: We carried out measurements of the kernel compile workload on virtual machines configured with up to 16 virtual CPUs. We recorded the number of dirtied pages and the disk sectors written according to the replication epoch length up to 1000 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, moreover, at the end of each epoch the dirty-log is reinitialized and all TLB entries have to be invalidated. In Section VI we provide exact numbers to what extent the dirty-log mode affects execution time under different setups.

Figure 1a illustrates the numbers obtained. Kernel compile has good scaling properties with the number of vCPUs deployed in the virtual machine. As seen, the average number of dirtied pages increases steadily with the increasing number of virtual CPUs. The key observation, however, is the fact that regardless the number vCPUs, dirty pages tend to increase fast in the beginning of an epoch, but the increase declines once a certain set of pages is written. We call this the *dirty page set*. The *dirty page set* will play an important role in our scheduling algorithm, which will be discussed in Section IV.

Another observation is the amount of disk I/O involved in the kernel compile workload. Figure 1b depicts the average number of sectors written based on the number of vCPUs and the replication epoch length. 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. Disk I/O is also important from the aspect of replication, because it needs to be buffered first and can be only released when the backup machine acknowledged the corresponding update.

2) NAS Parallel Benchmarks: We performed the same set of experiments for various applications from the NPB benchmarks. We used their OpenMP version, due their excellent scalability over SMP configurations, and set the $OMP\_NUM\_THREADS$ environment variable in the VM to the number of virtual CPUs. We identify two groups of applications among the NPB benchmarks regarding their memory behavior. One touches an increasing amount of memory with the growing number of vCPUs, while the other does not dirty significantly more pages with regards to the number of vCPUs in the system. Figure 2 shows the memory behavior of the ep.B and sp.B benchmarks, two representatives of the two groups, respectively. Again, the key observation is that with long enough replication epochs, these workloads also converge to a stable *dirty page set*.

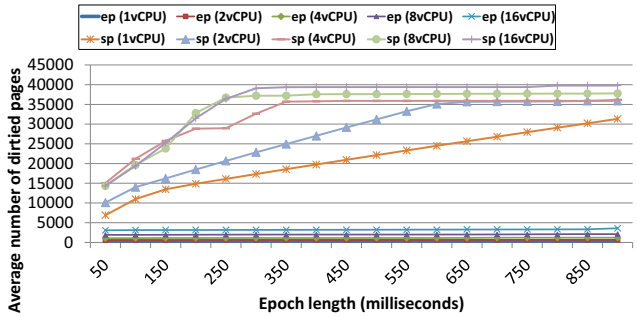
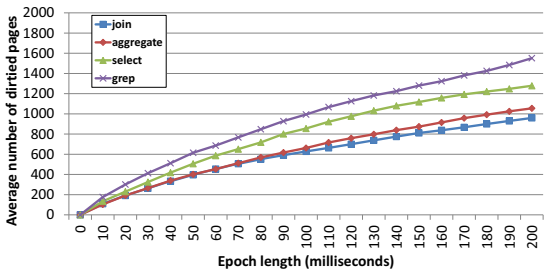


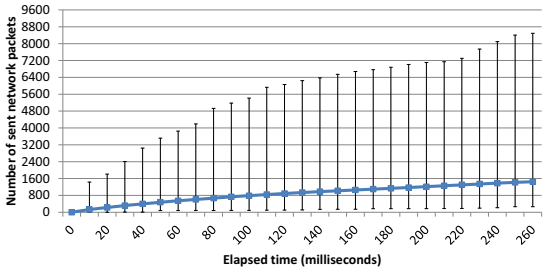
Fig. 2: **NPB EP and SP memory behavior.** Average number of dirtied memory pages according to replication epoch length and the number of vCPUs.

3) Hadoop JobTracker: In order to assess the feasibility of replicating the Hadoop master node, we examined its memory and network I/O behavior. In our experiments the JobTracker coordinates a rather small cluster (16 worker nodes), and thus, the number of vCPUs appeared to be irrelevant. Figure 3a shows the average number of dirtied pages for different tasks and replication epoch lengths. We present data up to 200 milliseconds in this configuration, because the JobTracker generates a fair amount of network traffic, and therefore, short replication epochs will be desired. As seen, the number of dirty pages is relatively small for each of the MR/DB jobs.

Figure 3b depicts the average number of network packets sent across all Hadoop jobs we examined. The error-bars represent the deviation in the number of packets for the given epoch length. The main observation is precisely the high deviation, which allows enough space for scheduling checkpoints in order to minimize the performance degradation due to the replication.



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



(b) Network packets sent.

Fig. 3: **Hadoop JobTracker memory and network I/O behavior.** Average number of dirtied pages and network packets sent (error-bars representing deviation) as the function of replication epoch length.

4) 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. We obtained relatively low numbers for the dirtied memory pages (compared to the kernel compile or the NPB workloads), and also for disk I/O operations.

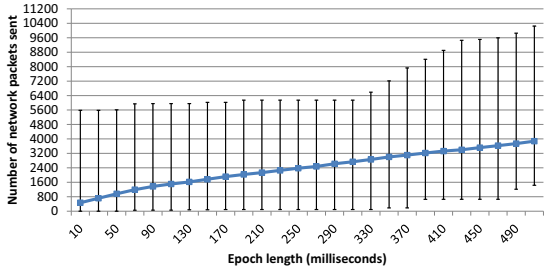


Fig. 4: **SPECweb Banking network behavior.** Average number of network packets sent (error-bars representing deviation) according to replication epoch length.

The network behavior of SPECweb is shown in Figure 4. As seen, there is a substantial amount of network packets sent in the SPECweb workload. However, likewise Hadoop JobTracker's network behavior, SPECweb also shows significant deviation from the average number of packets. Such differences can be exploited in order to determine when to take a checkpoint with minimal impact on the performance.

III. Checkpoint-Recovery Based VM Replication

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 [5].

At the same time, 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 [9]. Consequently, 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 [5], [6], [7].

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* [5], is to suspended the VM, collect the changes into a separate buffer, and resume the VM before beginning the actual data transfer. In the rest of this paper we will refer to this solution as the *regular asynchronous replication*.

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. Although data transfer can overlap the next epoch's execution, copying this big amount of data into a separate buffer by itself takes a significant amount of time, during which the VM needs to be suspended in the regular asynchronous replication approach. In order to alleviate the 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 bitmap, it is preserved by the VMM. During the next execution phase, each time a write page-fault occurs (note that the VM runs in dirty-log mode), the old bitmap is first consulted. If the write refers to a page

present in the old dirty bitmap and the page is not yet copied to the transfer buffer, 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	SPECweb
Ratio	41%	56%	30%	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 30% and 56%, allowing the fine-grained COW mechanism to save significant amount of work via not copying unnecessary data.

Our proposed mechanism allows the VM to resume 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 for the data transfer. This approach essentially follows a similar idea to concurrent checkpointing [15], although, contrary to capturing the entire address space each time, it saves changes incrementally.

B. Infiniband

High-speed interconnects, such as Infiniband [16], are widely used in High Performance Computing (HPC). They 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. Besides Gigabit Ethernet, we also integrate Infiniband's RDMA feature into our replication engine, so that we can compare the behavior of the adaptive checkpoint scheduling over different network architectures.

IV. Checkpoint Scheduling Heuristics

In this section we introduce several heuristics for scheduling checkpoints during VM replication. Since the four main factors that affect replication performance are the number of dirtied memory pages, the number of pending disk and network I/O operations, and the available network bandwidth for replication, we now consider each factor one by one.

A. Dirty Memory

In Section II-B we analyzed the behavior of various workloads in terms of the number of memory pages they touch as the function of replication period length. One of the key observations was that after a certain period of time each workload tends to reach its *dirty page set*, i.e. a set of pages that grows slowly afterwards. In order to avoid retransferring the same pages frequently, one of our main heuristics is to continuously monitor the number of dirty pages and delay checkpoints until the dirty page set is reached. We define that the *dirty page set* is attained when the number of dirty pages grows less than 5% between subsequent dirty page updates. On the other hand, we also consider the condition of pending network and disk I/O, which may demand earlier checkpoints.

B. Disk I/O

We have seen previously in Section II-B that the number of written disk sectors can grow rapidly for certain workloads, especially with the increasing number of vCPUs in the system. We follow a very straightforward heuristic with respect to pending disk I/O. Once the disk buffer is nearly full, a checkpoint can be no longer delayed. Note, that the disk buffer size is a parameter of the system, and we use 50MB for this purpose in our experiments.

C. Network Packets

Pending network I/O is the most intricate factor in the checkpoint scheduling algorithm. As seen is Section II-B, the number of pending network packets shows significant variation with different replication epoch lengths. Unfortunately, the VMM is unable to determine to what extent a given application is sensitive for network latency. Thus, we simply assume that pending network I/O should be released as soon as possible. One approach would be to initiate a checkpoint immediately when network packets are buffered. However, this would result in continuous checkpointing leaving hardly any space for the application to progress. On the other hand, the longer we let the VM to progress, the more memory pages will be touched and consequently, the higher the network latency will be rendered. Note, that network packets can be only sent out once the ACK for the given update is received from the backup machine.

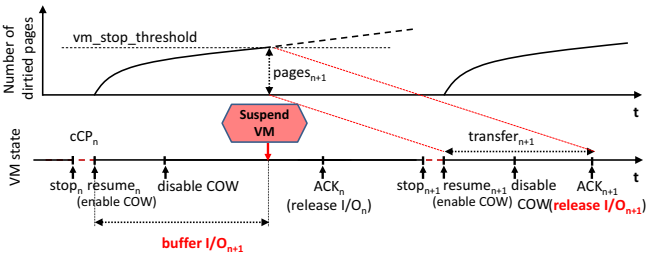


Fig. 5: **Trading throughput for latency.** Early VM suspend slows down the virtual machine so that fewer dirty pages will have to be transferred, which in turn lowers the latency of buffered network packets.

We propose a solution that attempts to make a compromise in between these boundaries. Figure 5 demonstrates our approach. The idea is to delay the checkpoint, but if pending network I/O is present, only wait until the number of pages reaches the lower limit where the replication bandwidth is already fully utilized. We call this *vm_stop_threshold* and it represents the minimum number of pages for which the maximum of the application's state change can be transferred per unit time. If it's necessary, the VM is slowed down (by means of early suspend) once *vm_stop_threshold* is reached, essentially, trading throughput for network latency. As it will be shown in Section VI, this approach yields significant im-

provements for latency sensitive workloads, such as SPECweb and Hadoop's JobTracker.

D. Replication Bandwidth

The network bandwidth available for replication is another important factor, therefore, our algorithm maintains an approximation of the available bandwidth. This value is updated after the transfer phase of each replication epoch, and in turn is used for determining the *vm_stop_threshold* variable discussed above.

E. Algorithm

Putting it all together, Figure 6 depicts the high-level scheduling algorithm. Note, that the pseudo code demonstrates the main execution steps, and the actual low-level implementation differs in some technical details. The scheduler's activities are as the followings. Once a checkpoint is requested, it tracks the number of dirty pages, the number of network packets and disk sectors buffered. If there are pending network packets and the number of dirty pages are bigger than *vm_stop_threshold*, it suspends the VM.

```
do forever
        request checkpoint;

        while !(received ACK)
                track dirty_pages;
                track net_tx_packets;
                track disk_wr_sectors;
                update wait_time;

                if (net_tx_packets &&
                   dirty_pages > vm_stop_threshold)
                        stop VM;
                end
        end

        update bandwidth_approx;
        update vm_stop_threshold <- bandwidth_approx;

        while (wait_time < WAIT_MAX_THRESHOLD &&
              !network_tx_packets &&
              !(disk_wr_sectors > DISK_BUF_LIMIT) &&
              !(dirty_pages reached working set size))

                track dirty_pages;
                track net_tx_packets;
                track disk_wr_sectors;
                update wait_time;

        end
end
```

Fig. 6: **Checkpoint scheduling algorithm pseudo code.**

When the ACK of the update is received from the backup machine, it updates bandwidth approximation and *vm_stop_threshold* accordingly. Unless there are pending network packets, it enters a waiting period where it continues tracking the number of dirty pages, and the number of buffered network packets and disk sectors. In case it detects pending network packets, the disk sectors exceed the disk buffer limit, or the overall wait time is above a predefined limit (currently WAIT_MAX_THRESHOLD is 2 seconds in our implementation) it requests a checkpoint and restarts its activities.

V. Implementation

A. KVM

We chose the Linux Kernel Virtual Machine (KVM) [17] 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.

The most important components of KVM 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

Another particular issue worth mentioning is the transactional nature of updating the backup machine. When replication data are sent to the backup host, *qemu-kvm* cannot just read and apply the changes directly, because a failure during the update would leave the backup machine in an inconsistent state.

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 [18]. 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.

VI. Evaluation

This section provides performance results of our adaptive replication method. We follow the evaluation scheme of prior studies in the context of VM replication [7], [8] and focus on the overhead to the failure free execution.

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. Results

1) 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.

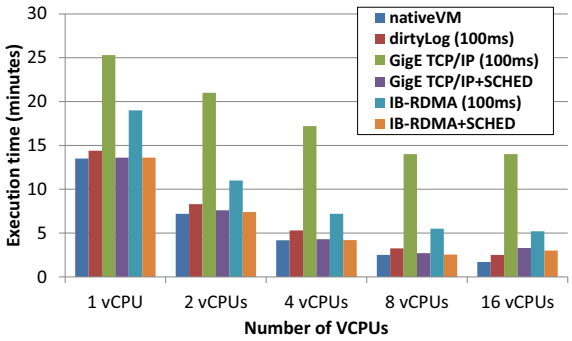


Fig. 7: **Kernel Compile runtimes.**

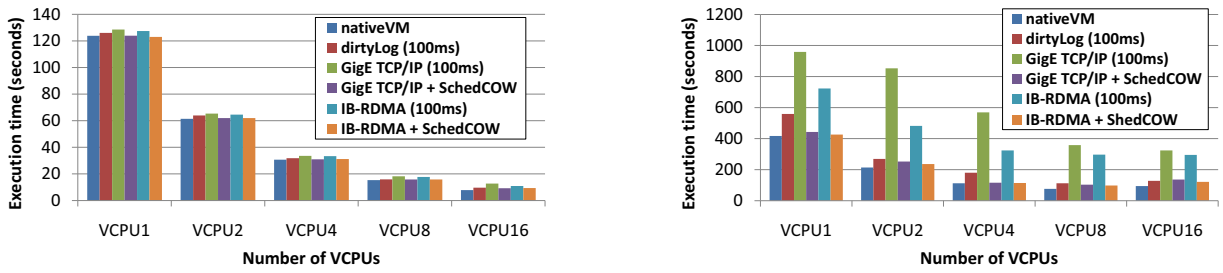
Figure 7 illustrates the execution times on the native virtual machine, the VM running in log-dirty mode, replicated over Gigabit Ethernet and Infiniband RDMA, with either fixed checkpoint frequency or dynamic scheduling. Similarly to related work [5], we set the fixed replication period in this experiment to 100 milliseconds. 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 VM replication, especially, if the checkpoint frequency is set beforehand.

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 7, RDMA based replication attains an increasing performance improvement compared to Gigabit Ethernet with the growing number of CPUs in the system, when the replication period is fixed. The key observation, however, is the effect of adaptive checkpoint scheduling, which can dynamically decrease checkpoint frequency so that data transfer can be entirely overlapped with VM execution. The scheduling algorithm detects the absence of pending network I/O and adjusts checkpointing frequency based on the condition of the disk buffer. For instance, we observed an average 1.2 seconds replication epoch length in case of 8 vCPUs. Accordingly, the achieved performance is much closer to the native VM even over Gigabit Ethernet.

On a 16 vCPUs replicated VM with dynamic checkpoint scheduling, the kernel compilation suffers 32%, and 20% slowdown over Gigabit Ethernet and Infiniband, respectively, while with 8 vCPUs the performance degradation is only 8% for GigE and 2% for Infiniband.

2) NAS Parallel Benchmarks: We present runtimes for two benchmarks from the NPB benchmark set. Figure 8a illustrates the results for ep.B. As shown before in Section II-B, ep.B is a light workload in terms of the number of dirtied memory pages. This is well reflected on the runtimes, which show nearly no performance degradation regardless the replication method and the network connection used.

On the other hand, as seen in Figure 8b, replicated sp.B attains significant performance improvements from adaptive scheduling compared to fixed checkpoint frequency. Up to 4



(a) ep.B runtimes.

(b) sp.B runtimes.

Fig. 8: **NAS Parallel Benchmarks runtimes.**

vCPUs it achieves close to native VM performance, and yields runtimes slightly better than the 100ms fixed dirty log mode. Nevertheless, performance degradation for 8 and 16 vCPUs is substantial, 35% and 44%, respectively, when replicated over Gigabit Ethernet.

3) SPECweb: 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.

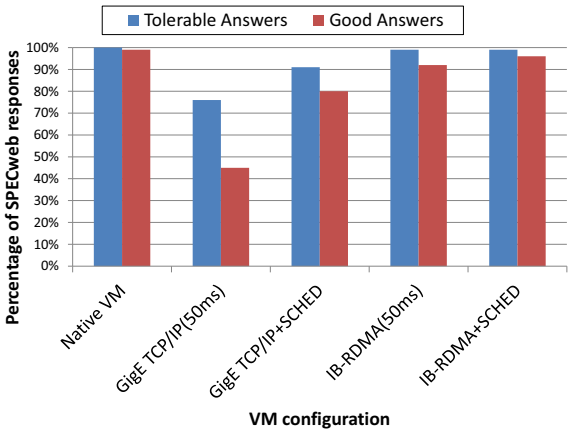


Fig. 9: **SPECweb Banking scores.**

We replicate only the main SPECweb application server and ran five different setups: native VM, replication over GigE and Infiniband, with fixed and adaptive replication frequency. 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. Therefore, both Infiniband's faster data transfer and the adaptive scheduling algorithm are expected to yield significant performance improvements.

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. All the different 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.

Figure 9 shows the results we obtained from these experiments. As seen, the performance attained by fixed checkpoint frequency over Gigabit Ethernet based replication is low, 45% and 76% for "good" and for "tolerable" answers, respectively. Infiniband's extreme network speed achieves 92% and 99% with the same checkpoint frequency. However, the performance degradation over GigE is mitigated by the adaptive checkpoint scheduling, which scores 79% and 91%, for "good" and "tolerable" answers, respectively. When adaptive scheduling is utilized in conjunction with Infiniband, the attained performance is 96% and 99% of the native scores.

4) Hadoop Jobtracker: The last application we investigate is Hadoop. In this experiment we replicate Hadoop's master node, the JobTracker, which is responsible for coordinating the MapReduce job among the worker nodes. The JobTracker does not perform any computation related to the job itself, but its role is crucial for orchestrating the parts of the whole execution, and therefore, a good candidate for fault tolerant execution.

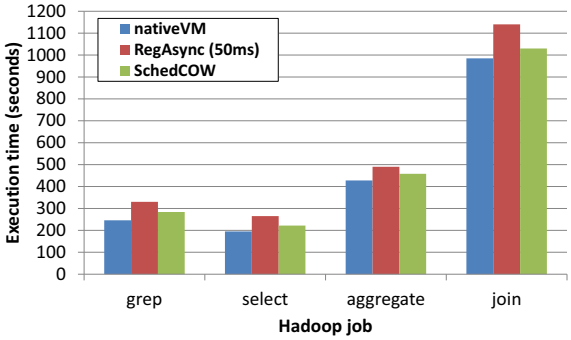


Fig. 10: **Hadoop Job runtimes.**

We used four jobs from the MR/DB benchmark set [14], grep, select, aggregate, and join. Our Hadoop cluster consists of 16 worker nodes, each of them is a 1 vCPU VM with 1 GB of RAM, residing on four different physical hosts

equipped with Intel Xeon 2.2GHz Quad-core CPUs. We use approximately 2GB of raw data on each node as input. Figure 10 shows the results we obtained from each experiment. We replicate the JobTracker over Gigabit Ethernet with fixed (50 milliseconds) checkpoint frequency and with our adaptive scheduling algorithm.

As seen, the price of replication in this experiment is modest, with fixed checkpoint frequency the worst case performance degradation is around 33%, in case of *grep*. However, adaptive scheduling alleviates the performance degradation and renders the worst case overhead to 15%, while the best performance it achieves is a 4% difference from the native VM execution in case of the *join* job.

VII. 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* [19], *KVM* [17], and *VMware's VMotion* [20] all provide the capability of live migrating VM instances. Pre-copy is the dominant approach to live VM migration [19], [20]. 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 [5], [6].

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 [21], while *Microwiper* [22] 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 [23], Note however, that they only consider uni-processor VMs, besides, VM replication involves various additional technical issues.

B. Virtual Machine Replication

Bressoud and Schneider [2] 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 [3]. 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* [24] 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* [25] and *CoreDet* [26] 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* [4] 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* [5] and *Paratus* [6] 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* [27] 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. [7] proposed fine-grained dirty region identification to reduce the amount of data transferred during each replication epoch, while Zhu et al. [8] 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. To the contrary, we focus on multi-core virtual machines in conjunction with workload adaptive scheduling of checkpoints.

VIII. 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 have proposed several heuristics for scheduling checkpoints during VM replication in a workload adaption fashion. Our algorithm dynamically adjusts the checkpoint frequency based on the properties of the given workload, such as the number of dirtied memory pages, the number of disk I/O operations and the number of transferred network packets. Moreover, it also takes the network bandwidth available for replication into account. It attempts to minimize the overhead of replication by delaying checkpoints and it trades throughput for latency if there are pending network packets. 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.

We have evaluated our checkpoint scheduling algorithm over two different network architectures and showed that the price of replicated virtual machines with up to 16 vCPUs can be modest even over commercial Gigabit Ethernet, in case checkpoints are carefully scheduled. For instance, the kernel compile workload suffers only 32% slowdown when run over a 16 vCPUs replicated VM, while the price of a replicated Hadoop master node is approximately a 10% slowdown compared to the native execution. SPECweb Banking, in turn, attains 80% of the native score with workload adaptive scheduling of replication over Gigabit Ethernet.

Reduction of replication data is an orthogonal approach to scheduling, and therefore it may provide further improvements. In the future we intend to investigate how a lightweight compression method could be integrated into our scheduling algorithm, and to what extent it could yield further performance improvements.

ACKNOWLEDGMENT

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

REFERENCES

[1] R. McDougall and J. Anderson, "Virtualization performance: perspectives and challenges ahead," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 40–56, December 2010.

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

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

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

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

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

[7] M. Lu and T. cker Chiueh, "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.

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

[9] 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.

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

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

[12] "Hadoop." http://hadoop.apache.org.

[13] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008.

[14] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "MapReduce and parallel DBMSs: friends or foes?" *Commun. ACM*, vol. 53, pp. 64–71, January 2010.

[15] K. Li, J. F. Naughton, and J. S. Plank, "Low-Latency, Concurrent Checkpointing for Parallel Programs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, pp. 874–879, August 1994.

[16] "InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2."

[17] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Ottawa Linux Symposium*, July 2007, pp. 225–230. [Online]. Available: http://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf

[18] "OpenFabrics Alliance." http://www.openfabrics.org.

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

[20] M. Nelson, B. H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2005, p. 25. [Online]. Available: http://portal.acm.org/citation.cfm?id=1247360.1247385

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

[22] Y. Du, H. Yu, G. Shi, J. Chen, and W. Zheng, "Microwiper: Efficient Memory Propagation in Live Migration of Virtual Machines," in *Proceedings of the 2010 39th International Conference on Parallel Processing*, ser. ICPP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 141–149. [Online]. Available: http://dx.doi.org/10.1109/ICPP.2010.23

[23] W. Huang, Q. Gao, J. Liu, and D. K. Panda, "High performance virtual machine migration with RDMA over modern interconnects," in *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, ser. CLUSTER '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 11–20. [Online]. Available: http://dx.doi.org/10.1109/CLUSTR.2007.4629212

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

[25] D. Lee, B. Wester, K. Veeraraghavan, S. Narayanasamy, P. M. Chen, and J. Flinn, "Respec: efficient online multiprocessor replayvia speculation and external determinism," ser. ASPLOS '10. ACM, 2010, pp. 77–90.

[26] T. Bergan, O. Anderson, J. Devietti, L. Ceze, and D. Grossman, "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*, ser. ASPLOS '10. ACM, 2010, pp. 53–64.

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