Checkpoint-Restart for a Network of Virtual Machines

Rohan Garg, Komal Sodha, Zhengping Jin, Gene Cooperman*
Northeastern University
Boston, MA / USA
{*rohgarg,komal,jinzp,gene*}*@ccs.neu.edu*

***Abstract*—The ability to easily deploy parallel computations on the Cloud is becoming ever more important. The first uniform mechanism for checkpointing a network of virtual machines is described. This is important for the parallel versions of common productivity software. Potential examples of parallelism include Simulink for MATLAB, parallel R for the R statistical modelling language, parallel_blast.py for the BLAST bioinformatics software, IPython.parallel for Python, and GNU parallel for parallel shells. The checkpoint mechanism is implemented as a plugin in the DMTCP checkpoint-restart package. It operates on KVM/QEMU, and has also been adapted to Lguest and pure user-space QEMU. The plugin is surprisingly compact, comprising just 400 lines of code to checkpoint a single virtual machine, and 200 lines of code for a plugin to support saving and restoring network state. Incremental checkpoints of the associated virtual filesystem are accommodated through the Btrfs filesystem. Experiments demonstrate checkpoint times of a fraction of a second by using forked checkpointing, mmap-based restart, and incremental Btrfs-based snapshots.**

I. Introduction

An approach for providing fault-tolerance to complex distributed applications is demonstrated. It is based on checkpointing a network of virtual machines. Such a network can be started locally, and later checkpointed for re-deployment (restart from checkpoint images) in the Cloud. This is especially important to support *fault tolerance* and *load balancing* in the Cloud.

The approach also provides *flexibility*. It employs DMTCP, an unprivileged, purely user-space checkpointing package. Potential examples of flexible application-specific policies are: incremental checkpointing, declaration of *cutouts* (regions of memory that don't require checkpointing); application-specific memory compression during checkpoint (for example, conversion of double to float), and so on. End users can write application-specific DMTCP plugins to support flexible checkpointing.

* This work was partially supported by the National Science Foundation under Grant OCI-0960978.

978-1-4799-0898-1/13/$31.00 © 2013 IEEE

Further, the *maintainability* of a proposed architecture is important. Here, we measure maintainability by the number of lines of *new* code required, beyond the base code of a checkpoint-restart package, or the base code of the virtual machine itself. The proposed architecture relies on just 600 lines of new code: 400 lines of code for a KVM-specific plugin used to checkpoint the virtual machine, and 200 lines of code for a TUN/TAP plugin. The two DMTCP plugins above are external libraries loaded into an unmodified DMTCP. Source code can be found in the contrib directory of the DMTCP repository. (See Section II for further details of plugins.)

The approach described here saves the state of an arbitrary guest operating system, which runs within a virtual machine under a Linux host operating system. The primary virtual machine described in this work is KVM/QEMU [1]. However, to demonstrate the generality of the approach, a plugin was also developed for Lguest [2]. That plugin required about 100 lines of code, as well as about 40 lines of modifications to the Lguest kernel driver to extend its API. The methodology was also applied to pure user-space QEMU [3]. Surprisingly, DMTCP was able to checkpoint user-space QEMU "out-of-the-box" (without the use of additional plugins).

Experiments in Section IV-C demonstrate compatibility with DMTCP's performance optimizations: *forked checkpointing* and *mmap-based fast restart*. Forked checkpointing enables virtual machine snapshot in 0.4 seconds when running with the Btrfs filesystem, while mmap-based fast restart allows resuming from the snapshot in 0.3 seconds. In addition, Section IV-D shows the run-time overhead to be too small to measure when running the nbench2 [4] benchmark program.

Snapshots (including the filesystem): In VM terminology, a *snapshot* saves not only the state of the virtual machine, but also the filesystem used by that virtual machine. The Btrfs filesystem [5] can be used to implement copy-on-write *incremental snapshots*. Thus, during checkpoint of a virtual machine, one can also create either a full snapshot or an incremental snapshot of the guest filesystem.

On computers where the host operating system does not provide the Btrfs filesystem, it is still possible to employ Btrfs. An "inner" KVM/QEMU virtual machine can be run nested inside an "outer" KVM/QEMU virtual machine, which in turn runs under the host operating system. The outer VM provides Btrfs and DMTCP runs inside the outer VM, checkpointing the inner VM.

In the rest of this paper, Section II provides background on DMTCP plugins. Section III describes a generic mechanism for checkpoint-restart for single virtual machines. Section IV provides experimental running times over a variety of scenarios, Section V describes related work, and Section VI provides the conclusion.

II. DMTCP, KVM, AND TUN/TAP: EXTENDING CHECKPOINT-RESTART TO VMS

DMTCP (Distributed MultiThreaded CheckPointing) [6] is used to checkpoint and restart a network of virtual machines. DMTCP provides a facility for third-party plugins, as well as using them in its own internal architecture. The work described here is based on the svn revision 1967 of DMTCP [7].

DMTCP implements transparent user-space checkpoint-restart. It does this by saving to a checkpoint image all of user-space memory, along with pertinent process state (thread information, open file descriptors, associated terminal device, stdin/stdout/stderr, sockets, shared memory regions, etc.). Internal DMTCP plugins employ specific algorithms to checkpoint the state of open files, network sockets, shared memory regions, and other special cases.

This work uses the plugin mechanism to extend DMTCP in two directions: support for KVM, and support for the virtual-network kernel devices TUN and TAP. TUN/TAP is used for networking of multiple KVM-based virtual machines. First, DMTCP is extended to support checkpointing of a single KVM/QEMU virtual machine. Second, DMTCP is extended to support checkpointing of the TUN/TAP network, including any network data "in flight".

In order to checkpoint KVM/QEMU, it is launched under the control of DMTCP. A typical example of launch, checkpoint, and restart is as follows:

```
% dmtcp_checkpoint --with-plugin \
      dmtcp_kvm_plugin.so \
      dmtcp_tun_plugin.so qemu ...
% dmtcp_command --checkpoint
% dmtcp_restart qemu_*.dmtcp
```

Section II-A discusses handling of the KVM/QEMU virtual machine, while Section II-B discusses network handling and the use of TUN/TAP.

A. Checkpointing the KVM/QEMU Virtual Machine

QEMU uses KVM to run user-space code natively on hardware that supports virtualization. It uses KVM's API to initialize and control the guest virtual machine. This API is based on the `ioctl` system call.

For the rest of this discussion, the term QEMU is used both to refer to the QEMU virtual machine monitor (VMM), and the virtual machine itself (including the guest operating system).

DMTCP plugins offer two primary mechanisms to extend checkpoint-restart: a run-time mechanism (wrapper functions around library calls made by the application); and customization of checkpoint/restart to save and restore the state of external objects. In this case, QEMU is the target application being checkpointed, and the KVM kernel module is the external object whose state must be virtualized.

The run-time portion of the KVM plugin is primarily concerned with a function wrapper around the `ioctl` system call. This wrapper function captures system calls by QEMU to KVM. This is used to make a local copy of the parameters that QEMU used to initialize the new virtual machine. At the time of restart, those same parameters are used to reset the KVM parameters to correspond.

The remainder of the KVM plugin is concerned with saving state at checkpoint time, and restoring state at restart time. The KVM saved state includes the state of the virtual CPU (registers, etc.) and the state of the interrupt controllers. The KVM API provides explicit system calls that the plugin used to save and restore the above state.

Another example of KVM/QEMU state is the virtual memory tables. These tables are contained within the user-space memory of the QEMU process itself (here viewing QEMU as a process in the host operating system). At the time of restart, the original mapping between the guest physical pages and host physical pages has been lost. However, the DMTCP plugin does not need to create a new mapping. This is because the page fault causes the hypervisor to re-establish the mapping.

Figure 1 illustrates the generic architecture of a guest virtual machine. At the time of checkpoint, the DMTCP plugin discovers the parameters of the KVM hypervisor in supporting the current state of the QEMU virtual machine. DMTCP then writes to a checkpoint image the memory of the QEMU virtual machine, which consists of the user-space memory of the process of the host operating system that is running QEMU.

Figure 2 presents the launching of a fresh virtual

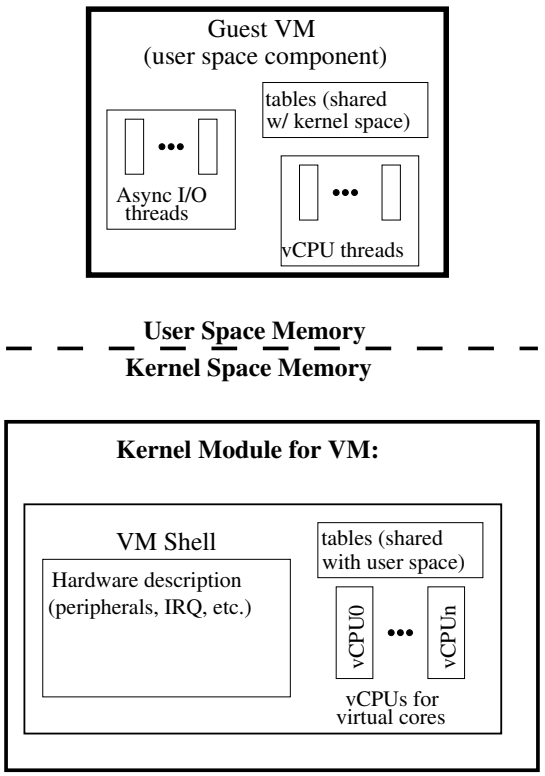

Figure 1: Generic VM Architecture. This sketch shows the VM components of interest for checkpoint-restart. The *VM shell* refers to the uninitialized data structures in the kernel driver that describes the virtual machine. A VM launcher initializes those data structures. A generic checkpoint-restart mechanism restores those data structures appropriately.

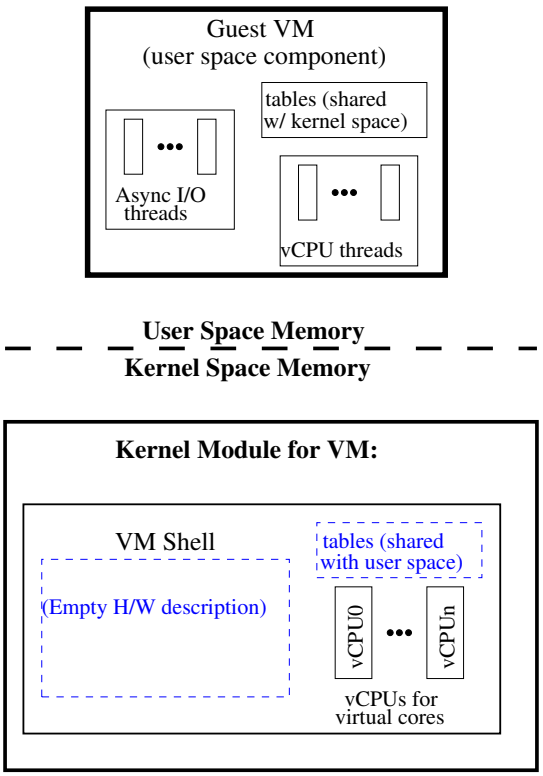

Figure 2: Re-Starting Virtual Machine from Checkpoint Image. DMTCP Plugin re-creates the original hardware description from the checkpoint image. In addition, the user-space memory of the guest VM is restored by DMTCP at the original addresses.

machine at restart time, which is then modified to correspond to the pre-checkpoint QEMU. At the time of restart, the DMTCP plugin requests KVM to create a fresh virtual machine (not specific to QEMU). Then, DMTCP replaces this fresh virtual machine (which exists as the user-space memory of a process in the host operating system) by the original user-space memory from the checkpoint image. Finally, the DMTCP plugin makes calls to the KVM kernel module to reset the KVM parameters so as to correspond to those of the pre-checkpoint QEMU virtual machine.

B. Checkpointing the TUN/TAP Network

A TUN/TAP plugin extends DMTCP similarly to the KVM plugin. Wrapper functions are implemented for `ioctl` to detect how the network was set up.

For background, we briefly review how DMTCP provides checkpointing over a TCP/IP network. At the time of checkpoint, "drains the network": (a) by stopping user threads of all processes in the computation; (b) receiving from each socket until all network data "in flight" has been collected; and (c) by then writing a checkpoint image. A "cookie" (unique set of data) is sent through each network connection so that the receiver can determine when no further data is in flight.

The TUN/TAP plugin employs a similar strategy, except that TUN/TAP does not provide an analog of a socket connection. It operates at a lower level in which network packets generated by the guest operating system are injected directly into the physical network. Only the guest operating system is aware of the socket connections being used by the applications within it.

Two alternative approaches to draining the network are: (a) to send a broadcast packet that plays the role of the DMTCP cookie; and (b) to wait for a specified time sufficient for all network packets to arrive. Mechanism (b) is used currently. For added reliability, at the end of writing the checkpoint image, the network is

checked to see if any late packets have arrived. If a late packet is detected, the user can be warned, or a second DMTCP checkpoint can be automatically initiated.

III. Generic Mechanism for Checkpointing a Single Virtual Machine

The techniques employed by the KVM plugin from Section II-A extend to other virtual machines. In particular, a DMTCP plugin was written for the Lguest virtual machine. In this case, Lguest provides a control mechanism by overloading the `read` and `write` system calls. Plugin wrapper functions were written for these calls. The Lguest kernel module also had to be modified with about 40 lines of code, in order to extend the Lguest API for `read`/`write`. This enables the Lguest plugin to discover and restore the virtual machine state. The plugin itself comprised 100 lines of code.

In the case of user-space QEMU (no KVM kernel module), the task of checkpointing is even simpler. The existing DMTCP package was found to correctly checkpoint and restart QEMU without any additional plugins. See Tables VII, VIII and X for timings across Lguest, KVM/QEMU and pure QEMU.

IV. Experimental Results

The experimental results are split into four subsections concerning: a network of virtual machines; the use of Btrfs for filesystem snapshots; DMTCP optimizations; and performance on a commodity computer.

Scalability is tested for two different architectures: distributed computing across a cluster of 12 nodes; and shared memory computing employing 16 CPU cores.

Configuration (cluster of 12 nodes): Each of the 12 computers is a 12-core Intel Xeon (1.6 GHz) server with 24 GB of RAM. The host operating system was a 64-bit version of CentOS-6.3 with Linux kernel 2.6.32. KVM/QEMU was chosen as the VMM. The guests were set up to run Ubuntu-12.04 Server version. DMTCP svn revision 1967 was used for these experiments.

Configuration (single node with 16 cores): These experiments were run on a 16-core AMD Opteron (1 GHz) server with 128 GB of RAM. The host operating system was a 64-bit version of Ubuntu-13.04 with Linux kernel 3.8. KVM/QEMU was chosen as the VMM. The guests were set up to run Ubuntu-12.04 Server version. DMTCP svn revision 1967 was used for these experiments.

A. Scalability of Checkpointing of Virtual Machines

Tables I, II, and III show that restart time increases slowly with the number of VMs, while checkpoint time is close to constant.

Further, Tables I and III show that two DMTCP options (further analyzed in Section IV-C) can enable checkpoint and restart in a fraction of a second. First, in *forked checkpointing*, a child process is forked in order to checkpoint while the parent continues running. Second, in *mmap-based fast restart*, mmap is used to map into RAM the memory saved within the checkpoint image. Hence, the process restarts faster, while remaining memory is paged into RAM on demand.

1) Scalability for a Distributed Network of VMs: Table I shows checkpoint and restart timings of HPCC [8].

Number Nodes	None (sec)		F/C (sec)		F/R (sec)		F/C + F/R (sec)	
	Ckpt	Restart	Ckpt	Restart	Ckpt	Restart	Ckpt	Restart
1	9.45	2.83	0.29	3.10	3.78	0.38	0.31	0.34
2	10.11	3.17	0.34	3.22	3.56	0.36	0.33	0.38
4	10.63	3.45	0.36	3.73	3.85	0.42	0.38	0.50
8	11.38	4.59	0.38	4.23	4.17	0.51	0.41	0.52
12	11.53	5.01	0.42	4.90	4.18	0.59	0.48	0.55

Table I: Checkpoint-restart of HPCC [8] benchmark on a Gigabit Ethernet cluster, as influenced by DMTCP's optional optimizations: forked checkpoint (F/C) and fast restart (F/R). DMTCP's default gzip compression of checkpoint images is incompatible with DMTCP F/R, and so is not used in those cases. (Memory allocated in each case is 1024 MB.)

2) Scalability for a Network of Virtual Machines in Multi-Core Shared Memory: Table II shows the efficiency for a network of virtual machines under shared memory. Coverage over three types of parallel middleware is demonstrated: MPI (HPCC [8]), TCP/IP sockets (IPython [9]), and PVM (the SNOW parallel computing framework for the R statistical programming language [10]).

Number of VMs	HPCC		IPython		Parallel R	
	Ckpt (s)	Restart (s)	Ckpt (s)	Restart (s)	Ckpt (s)	Restart (s)
1	9.84	3.31	9.63	3.46	10.02	3.68
2	10.08	3.75	10.44	4.10	10.54	4.17
3	10.18	3.86	10.67	4.06	11.13	4.16

Table II: Checkpoint-restart times for virtual machines on a single multi-core computer. (The allocated memory in each case is 1024 MB.)

Table III shows that the two DMTCP optimizations, forked checkpoint and fast restart, greatly enhance checkpoint and restart times. See Section IV-C for descriptions of those optimizations.

B. Btrfs: Incremental Snapshots of Virtual Machines

A virtual machine snapshot mechanism includes the ability to save the current state of the VM filesystem. This is implemented through the Btrfs copy-on-write

DMTCP Optimizations	HPCC (sec)		IPython (sec)		Parallel R (sec)	
	Ckpt	Restart	Ckpt	Restart	Ckpt	Restart
None	10.18	3.86	10.67	4.06	11.13	4.16
F/C	0.37	3.17	0.41	3.92	0.38	3.91
F/R	3.25	0.36	3.48	0.34	4.01	0.27
F/C + F/R	0.38	0.35	0.43	0.34	0.41	0.37

Table III: Checkpoint-restart of three VMs on a 16-core computer, while running different applications. The DMTCP optimizations are forked checkpoint (F/C) and fast restart (F/R). DMTCP's default gzip compression of checkpoint images is incompatible with DMTCP F/R, and so is not used in those cases. (Memory allocated in each case is 1024 MB.)

filesystem for incremental snapshots of the guest virtual filesystem. Even though the host machines in our experimental facilities did not provide a Btrfs filesystem, we were able to support a Btrfs filesystem through nesting of one KVM/QEMU virtual machine inside another. The outer virtual machine provides a Btrfs virtual filesystem for the inner one. DMTCP runs as a process inside the outer virtual machine, and is used to checkpoint the inner virtual machine. Networking of the VMs is supported through TUN/TAP, as before. Table IV demonstrates the scalability for a distributed computation across four nodes of the cluster.

	1 node (sec)		2 nodes (sec)		4 nodes (sec)	
Optimizations	Ckpt	Restart	Ckpt	Restart	Ckpt	Restart
with Btrfs	2.36	1.20	2.45	1.65	3.68	2.35
without Btrfs	33.28	35.67	34.46	37.20	39.73	39.47

Table IV: Snapshotting up to four distributed VMs running HPCC [8] under KVM/QEMU. The Btrfs filesystem is used to snapshot the filesystem using nested VMs. (Memory allocated in each case is 384 MB. The size of the guest filesystem is 2 GB.)

	Checkpoint (s)	Restart (s)
with Btrfs	1.52	0.7
Without Btrfs	10.23	12.48

Table V: Configuration is same as for Table IV, except that three VMs run on a single 16-core computer.

Tables IV and V show the advantage of using the copy-on-write feature of Btrfs to store the guest VM's filesystem. At checkpoint time a small additional DMTCP plugin rapidly copies the state of the entire filesystem (which appears as a single file on the outer guest's filesystem), using the `--reflink` option of the GNU binutils copy command. At restart time the state of the guest filesystem is similarly copied back. DMTCP's facilities for forked checkpointing and mmap-based fast restart were employed.

Tables IV and V show a performance penalty for restarting without Btrfs (using nested VMs), as compared to Table II (non-nested). DMTCP resides in the outer VM. Since the virtualization of I/O devices is never handled by KVM, the outer KVM then transfers control back to the outer QEMU. The outer QEMU resides in user space memory. The continual switching between kernel and user-space accounts for the inefficiency.

C. Optimizing: Forked Checkpointing and Fast Restart

DMTCP supports two further performance optimizations: forked-checkpointing and mmap-based fast-restart. Table VI demonstrates the much improved performance when using both of these optimizations. All experiments are run on the 16-core computer with just a single VM.

Allocated Memory (MB)	KVM/QEMU (F/C+F/R)		
	Checkpoint (s)	Restart (s)	Image Size
128	0.20	0.10	184 MB
256	0.19	0.09	310 MB
512	0.21	0.10	568 MB
768	0.22	0.10	822 MB
1024	0.21	0.10	1.1 GB

Table VI: Forked checkpoint (F/C) and fast restart (F/R) times for an idle VM under KVM/QEMU.

1) Forked checkpointing: Times for the forked checkpointing optimization are given for an idle virtual machine in Table VII. This uses the "`--enable-forked-checkpointing`" configure option of DMTCP. At checkpoint time, after "draining the network", a child process is forked. The child writes out the checkpoint image in parallel with the parent process continuing its execution. As expected, the parent completes its portion of the checkpoint largely independently of the size of the checkpoint image or allocated memory. Forked checkpointing typically requires 0.2 seconds.

The times for checkpoint and restart for KVM/QEMU are larger than the times for user-space QEMU. This is because the plugin for KVM/QEMU makes extra system calls at checkpoint and restart time. The times can be reduced by modifying the kernel driver to implement a new system call that coalesces all of the operations of the previous system calls.

2) Fast Restart: Times for the fast-restart optimization are given for an idle virtual machine in Table VIII. This uses the "`--enable-fast-restart`" option of DMTCP. This option uses mmap to map the checkpoint image from disk directly into virtual memory, instead of copying data from disk to virtual memory. In

Allocated	Lguest (F/C)			KVM/QEMU (F/C)			QEMU (user-space, F/C)		
Memory (MB)	Ckpt (s)	Restart (s)	Image Size	Ckpt (s)	Restart (s)	Image Size	Ckpt (s)	Restart (s)	Image Size
128	0.16	1.18	30 MB	0.18	1.28	44 MB	0.16	1.70	59 MB
256	0.17	1.43	32 MB	0.20	2.38	90 MB	0.17	2.99	111 MB
512	0.18	2.52	35 MB	0.23	3.06	122 MB	0.17	4.44	171 MB
768	0.17	2.45	36 MB	0.21	3.11	122 MB	0.18	4.97	191 MB
1024	0.18	2.82	37 MB	0.24	2.96	116 MB	0.19	5.63	213 MB

Table VII: Forked checkpointing (F/C) optimization for idle virtual machines.

Allocated	Lguest (F/R)			KVM/QEMU (F/R)			QEMU (user-space, F/R)		
Memory (MB)	Ckpt (s)	Restart (s)	Image Size	Ckpt (s)	Restart (s)	Image Size	Ckpt (s)	Restart (s)	Image Size
128	0.52	0.10	139 MB	0.69	0.10	182 MB	0.59	0.10	230 MB
256	0.83	0.10	267 MB	1.10	0.09	311 MB	1.33	0.10	408 MB
512	1.49	0.10	523 MB	1.84	0.10	566 MB	2.44	0.10	761 MB
768	2.50	0.10	779 MB	2.52	0.09	823 MB	3.54	0.10	1.1 GB
1024	3.02	0.10	1.1 GB	3.12	0.10	1.1 GB	4.48	0.10	1.5 GB

Table VIII: Fast restart (F/R) optimization for idle virtual machines.

this case, memory is demand-paged from the checkpoint image on an as-needed basis.

In addition to faster restart times, one observes faster checkpoint times. This is because fast restart disables the default gzip compression. The execution time of gzip normally dominates.

Note that on restart from a checkpoint image, the shadow page tables inside the kernel must be recreated, after which the pages will be faulted back into RAM. This impact is not captured in Tables VIII and VI, because most page faults occur after restart is complete.

D. Performance on a Commodity Host Computer

Configuration: The experiments of this section employed a MacBook laptop with an Intel Core i7 (2.3 GHz), a 256 GB SSD, and 8 GB of RAM. The host operating system was a 32-bit version of Ubuntu-12.10 with Linux kernel-3.5.7. The host was running natively in its own partition on the MacBook. The guest was set up to run Ubuntu-8.04 Desktop version. DMTCP svn revision 1967 was used for these experiments. Snapshots based on Btrfs (see Section IV-C) were used for all experiments.

Run-Time Overhead of DMTCP: The numbers in Table IX demonstrate the small overhead of executing with DMTCP. DMTCP incurs this overhead due to its use of lightweight wrapper functions around certain system calls. We used the nbench2 benchmark program [4] for these tests. The nbench2 benchmark program is a collection of applications that stress the cpu and the memory. Indexes for memory-intensive, integer-intensive, and floating-point-intensive computations are reported. Each *index* in Table IX is an nbench2 measure of performance, normalized to a value of one for the AMD K6/233. Higher numbers are better.

Table IX shows that DMTCP has little impact on performance for a VM running CPU-intensive or memory-intensive loads. As expected, the performance of KVM/QEMU is much higher than user-space QEMU, regardless of whether DMTCP is used.

	KVM/QEMU			QEMU (user-space)		
	Memory Index	Int. Index	Float-point Index	Memory Index	Int. Index	Float-point Index
with DMTCP	31.48	25.54	47.81	2.52	3.47	0.29
w/o DMTCP	31.38	25.52	48.38	2.44	3.34	0.27

Table IX: Nbench2 benchmark program on virtual machines. (Memory allocated in each case is 1024 MB. Higher index numbers represent higher performance.)

Influence of Memory Footprint: Table X analyzes the influence of the VM memory footprint on checkpoint-restart in the default mode of DMTCP (gzip compression) for an idle virtual machine. For larger sizes (guest VMs with 512 MB to 1024 MB), the checkpoint times grow proportionally to the size of the allocated memory for the larger sizes. Below these sizes, other factors dominate. Restart times do not change appreciably at the higher memory sizes.

V. Related Work

Virtual machines support snapshots, a form of checkpointing built into the virtual machine. Examples include Xen [11] and QEMU [3]. Xen has offered checkpointing (snapshots) at least since 2006 [12]. QEMU supports a "savevm" command to create a snapshot, both with and without KVM. Live checkpointing for KVM has been implemented using an additional checkpoint thread [13]. The CEVM system [14] uses a combination of KVM/QEMU's live migration and snapshotting facilities to provide a standalone high availability system. Similarly to CEVM, both Remus [15] and VM-μCheckpoint [16] offer high frequency checkpointing

Allocated	Free	Lguest			KVM/QEMU			QEMU (user-space)		
Mem. (MB)	Mem. (MB)	Ckpt (s)	Restart (s)	Image	Ckpt (s)	Restart (s)	Image	Ckpt (s)	Restart (s)	Image
128	2.5	2.29	1.26	30 MB	3.95	1.31	44 MB	4.34	1.69	59 MB
256	4.2	3.17	1.38	33 MB	6.42	2.35	89 MB	7.71	3.02	109 MB
512	184	5.39	2.42	35 MB	9.89	3.28	129 MB	11.87	4.43	170 MB
768	441	6.82	3.01	38 MB	9.21	3.31	130 MB	14.04	5.05	194 MB
1024	700	8.34	2.99	37 MB	10.03	3.13	122 MB	16.50	5.47	208 MB

Table X: Checkpoint-restart times for idle virtual machines. The checkpoint times include the times for compressing the memory image and writing the contents to the disk.

of guest VMs on Xen. They employ Xen's live migration and dirty page tracking facilities for incremental state snapshots. An earlier technical report provides additional details on the use of a DMTCP plugin in checkpointing a *single* virtual machine [17].

The Emulab system has demonstrated checkpointing of distributed systems through the use of virtual machines [18]. They did so using Xen and a guest virtual machine that ran a modified Linux kernel. The modified Linux kernel logs packets and replays them on restart. In addition, Emulab uses "delay nodes" (additional virtual machines) sitting between the user's virtual machines, in order to throttle network bandwidth to an acceptable level. In contrast, the current approach does not incur the run-time overhead of delay nodes, and supports any guest operating system — not just a customized Linux kernel. Finally, Emulab operates over the Xen hypervisor, while the current approach employs hosted virtual machines.

Checkpointing of distributed computations is primarily handled by one of two mechanisms today: checkpoint-restart services for MPI; and transparent checkpoint of arbitrary distributed computations. MPI implementations of checkpoint-restart typically operate by first stopping all MPI messages [19], [20], [21]. When it can be detected that there are no MPI messages in transit, a single-host checkpointing package is then employed. Often that single-host package is the kernel-based BLCR [22] package. Open MPI supports the option of using either MTCP (the single-process component of DMTCP) or BLCR. In addition to BLCR, two other commonly used packages for single-host checkpointing are CryoPid2 [23] and OpenVZ [24] (based on CRIU [25]).

DMTCP [6] was the first transparent user-space checkpoint-restart for distributed computations, and remains the most widely used example of this. Further, unlike the MPI approach, DMTCP permits network messages to be in transit when the checkpoint occurs.

For the support of snapshots, one requires a copy-on-write filesystem. A common current choice is QCOW2 [26], which supports the creation of incremental snapshots. Another recent choice is BlobSeer [27], as used in [28, Section 3.3]. That choice has the advantage of exposing the raw checkpoint image file to the host operating system or hypervisor. The work described here uses Btrfs [5]. Like BlobSeer, Btrfs exposes the raw checkpoint image to the host, making it compatible with the use of DMTCP from outside both the VM and the VM kernel driver.

VI. Conclusion

A mechanism for checkpointing a network of virtual machines has been presented. This uses the plugin architecture of the DMTCP checkpoint-restart package, on top of the KVM/QEMU checkpoint-restart package. The implementation requires a 400-line KVM-specific plugin, as well as a 200-line plugin to adapt Linux's TUN/TAP to allow DMTCP to "drain the network" prior to checkpoint. The plugin mechanism has the potential to be easily adapted to other virtual machines. The integration of the Btrfs copy-on-write filesystem with nested copies of KVM/QEMU was used for fast, incremental snapshots of a network of virtual machines.

Acknowledgment

The authors acknowledge Kapil Arya for helpful comments on this paper, and also for advice on creating the DMTCP plugins. They also thank Larry Owen, Anthony Skjellum and the University of Alabama at Birmingham (under NSF grant CNS-1337747) for providing a cluster with KVM and TUN/TAP.

References

[1] KVM team, "KVM — QEMU," http://wiki.qemu.org/KVM, see also http://www.linux-kvm.org/page/Main_Page, Accessed Nov. 18, 2012.

[2] R. Russell, "Lguest: The simple x86 hypervisor," http://lguest.ozlabs.org/, Accessed Nov. 18, 2012.

[3] QEMU team, "QEMU," http://wiki.qemu.org/Main_Page, Accessed Nov. 18, 2012.

[4] U. F. Mayer, "Linux/Unix nbench," http://www.tux.org/~mayer/linux/bmark.html; retrieved Dec. 4, 2012.

[5] O. Rodeh, J. Bacik, and C. Mason, "BTRFS: The Linux B-tree filesystem," IBM Research Report, Tech. Rep., July 2012, rJ10501 (ALM1207-004); http://domino.watson.ibm.com/library/CyberDig.nsf/papers/6E1C5B6A1B6EDD9885257A38006B6130/$File/rj10501.pdf.

[6] J. Ansel, K. Arya, and G. Cooperman, "DMTCP: Transparent checkpointing for cluster computations and the desktop," in *23rd IEEE Int. Symp. on Parallel and Distributed Processing (IPDPS-09)*, 2009, pp. 1–12.

[7] DMTCP team, "DMTCP : Distributed multithreaded checkpointing," http://dmtcp.sourceforge.net, Accessed Nov. 18, 2012.

[8] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The HPC challenge (HPCC) benchmark suite," in *Proc. of the 2006 ACM/IEEE Conf. on Supercomputing (SC-06)*. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188677

[9] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: http://ipython.org

[10] L. Tierney, A. J. Rossini, and N. Li, "Snow: a parallel computing framework for the r system," *Int. J. Parallel Program.*, vol. 37, no. 1, pp. 78–90, Feb. 2009. [Online]. Available: http://dx.doi.org/10.1007/s10766-008-0077-2

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. of 19th ACM symposium on Operating Systems Principles (SOSP-03)*. New York, NY, USA: ACM, 2003, pp. 164–177.

[12] G. Vallée, T. Naughton, H. Ong, and S. L. Scott, "Checkpoint/restart of virtual machines based on Xen," in *HAPCW'06: High Availability and Performance Computing Workshop*. Santa Fe, New Mexico, USA: Held in conjunction with LACSI 2006, Oct. 2006.

[13] V. Siripoonya and K. Chanchio, "Thread-based live checkpointing of virtual machines," in *10th IEEE Int. Symp. on Network Computing and Applications*, 2011.

[14] K. Chanchio, C. Leangsuksun, H. Ong, V. Ratanasamoot1, and A. Shafi, "An efficient virtual machine checkpointing mechanism for hypervisor-based HPC systems," in *High Availability and Performance Computing Workshop (HAPCW)*, 2008.

[15] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: high availability via asynchronous virtual machine replication," in *Proc. of the 5th USENIX Symp. on Networked Systems Design and Implementation (NSDI-08)*. Berkeley, CA, USA: USENIX Association, 2008, pp. 161–174. [Online]. Available: http://dl.acm.org/citation.cfm?id=1387589.1387601

[16] L. Wang, Z. Kalbarczyk, R. Iyer, and A. Iyengar, "Checkpointing virtual machines against transient errors," in *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International*, 2010, pp. 97–102.

[17] R. Garg, K. Sodha, and G. Cooperman, "A generic checkpoint-restart mechanism for virtual machines," Tech. Rep., 2012. [Online]. Available: http://arxiv.org/abs/1212.1787v1

[18] A. Burtsev, P. Radhakrishnan, M. Hibler, and J. Lepreau, "Transparent checkpoints of closed distributed systems in Emulab," in *Proc. of 4th ACM European Conf. on Computer Systems*. ACM, 2009, pp. 173–186.

[19] J. Hursey, J. M. Squyres, T. I. Mattox, and A. Lumsdain, "The design and implementation of checkpoint/restart process fault tolerance for Open MPI," in *Proceedings of the 21^{st} IEEE International Parallel and Distributed Processing Symposium (IPDPS) / 12^{th} IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems*. IEEE Computer Society, March 2007.

[20] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov, "MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes," in *ACM/IEEE 2002 Conference on Supercomputing*. IEEE Press, 2002.

[21] S. Sankaran, J. M. Squyres, B. Barrett, V. Sahay, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman, "The LAM/MPI checkpoint/restart framework: System-initiated checkpointing," *International Journal of High Performance Computing Applications*, vol. 19, no. 4, pp. 479–493, 2005.

[22] P. Hargrove and J. Duell, "Berkeley lab checkpoint/restart (BLCR) for Linux clusters," *J. of Physics Conference Series*, vol. 46, pp. 494–499, Sep. 2006.

[23] M. O'Neill, "Cryopid2," http://sourceforge.net/projects/cryopid2.

[24] OpenVZ team, "OpenVZ," http://wiki.openvz.org/.

[25] CRIU team, "CRIU," http://criu.org/.

[26] M. McLoughlin, "The QCOW2 image format," http://people.gnome.org/~markmc/qcow-image-format.html, 2008.

[27] B. Nicolae, G. Antoniu, L. Bougé, D. Moise, and A. Carpen-Amarie, "BlobSeer: Next generation data management for large scale infrastructures," *Journal of Parallel and Distributed Computing*, vol. 71, no. 2, pp. 168–184, Feb. 2011. [Online]. Available: http://hal.inria.fr/inria-00511414

[28] B. Nicolae and F. Cappello, "BlobCR: efficient checkpoint-restart for HPC applications on IaaS clouds using virtual disk image snapshots," in *Proc. of 2011 Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC-11)*. ACM, 2011, pp. 1–12.