bhyve - Improvements to Virtual Machine State Save and Restore

Darius Mihai
University POLITEHNICA of Bucharest
Splaiul Independenței 313, Bucharest, Romania, 060042
Email: dariusmihaim@gmail.com

Mihai Carabaș
University POLITEHNICA of Bucharest
Splaiul Independenței 313, Bucharest, Romania, 060042
Email: mihai.carabas@cs.pub.ro

***Abstract*—As more complex tasks are delegated to distributed servers, virtual machine hypervisors need to adapt and provide features that allow redundancy and load balancing. One such mechanism is the virtual machine save and restore through system snapshots. A snapshot should allow the complete restoration of the state that the virtual machine was in when the snapshot was created. Since the snapshot should encapsulate the entire state of the virtualized system, the guest system should not be able to differentiate between the moment a snapshot was created and the moment when the system was restored, regardless of how much real time has passed between the two events. This paper will present how the time management and block devices are saved and restored for *bhyve*, FreeBSD's virtual machine hypervisor.**

I. Introduction

Virtual machines can be used by extremely powerful systems (e.g., server farms) to more efficiently split resources between users, or run a variety of compatible operating systems without specifically installing one directly on the hardware system. These mechanisms are employed to reduce administrative complexity and better automate processes. Since a virtualized operating system is expected to still run as any other, tasks that depend on timer functionality and clock measurements are still expected to run with enough precision so results are not skewed over time.

This article refers to any system that allows running virtual machines as **host**, and the operating system running on virtualized hardware as **guest**. Since the host is responsible with managing the system resources, and therefore must not have its functionality hijacked by a badly behaving guest, a hypervisor (also known as virtual machine manager) is required to allow the guest systems access to hardware resources without impacting the host.

In some circumstances (e.g., the host will have to be stopped or if the host becomes over encumbered), users may want to stop the virtual machine, and potentially even move it to another system to continue work. This is achieved by creating a snapshot of the virtual machine and then restoring the virtual machine from the checkpoint.

A. Timers and Clocks

In order to perform periodic tasks, an operating system has to measure time in some manner, and, if the hardware permits it, request that an interrupt is sent when an interval has passed. Regardless of their exact function, a periodic task is a routine that will have to be called at (or as close as possible) a set interval.

For example, the basic Unix `sleep` command can be used to perform an operation every *N* seconds if `sleep $N` is called in a script loop. Since it is safe to assume that all modern processors have hardware timekeeping components implemented, sleep will request from the operating system that a software timer (i.e., one that is implemented by the operating system as an abstraction [1]) to be set for `N` seconds in the future and will yield the processor, without being rescheduled, until the system "wakes" it. The process "waking" is automatically done when the time has elapsed, and the process will be rescheduled to run by the operating system since it no longer waits for external events (in the case of sleep, it will usually simply end when the timer is done).

Considering that the virtual machine must not lose any functionality, time sensitive tasks should not be impacted either. More specifically, this implies that if a task ran in the virtual machine should end `N` seconds after it was created, it will expire `N-X` seconds after the virtual machine was restored from a checkpoint, if that checkpoint was created after `X` seconds.

Note that some tasks may be affected even when the virtual machine behaves as expected after a restore. For example, network communications may be timed out by the other end, even if from the guest's point of view the packets are expected to arrive well within the allotted time frame.

B. Block devices

A block device [2] is a permanent storage device that imposes access to data using a fixed width. A commonly used size is 512 bytes, equivalent to the length of a sector of a hard-disk drive (*HDD*). Reading or writing data to such devices requires sending requests of size that is a multiple of that size.

As with hardware systems, a virtual machine requires a virtualized block device as permanent storage to install the operating system, as well as other software. Similar to other virtual machine managers, *bhyve* [3] uses special files stored on a physical drive that act as storage medium for the virtual machine. The files are kept open by a part of bhyve and read/writes are performed in these files in stead of directly

on the block device (i.e., the file will ultimately be stored on a hardware device, using the abstraction layers of the FreeBSD kernel).

At the time this paper is written, bhyve only supports raw disks (i.e., data is stored without being compressed, and no extra features, such as copy-on-write, are available).

As with timers and clocks, the state of block devices needs to be correctly saved to avoid the corruption of the file systems used by the guest. Failing to do so, either by losing some requests received from the guest, or failing to send notifications when the operations have finished may result in the guest having an inconsistent view of the data on the disk, or corrupted data. Both scenarios can lead to damage to data and systems with bad behavior.

The following sections are split as follows:

- **section II** talks about how the timers and time measuring circuits work, how they are virtualized, and the previous state of the save and restore mechanism.
- **section III** presents how AHCI compatible block devices are virtualized
- **section Iv** shows the state of the overall save and restore feature of bhyve
- **section V** focuses on the improvements to the snapshot mechanism our work has achieved.
- **section VI** shows the results obtained after successfully restoring time components and virtualized AHCI compatible block devices on systems with Intel processors.
- **section VII** draws some conclusions and talks about future work on the save and restore feature for bhyve [3].

II. Time Management Virtualization

Timers are hardware resources used by the operating system for synchronization. Usually, the timers have internal counters that are incremented or decremented to measure time passage. Depending on how often the counter value is changed timers have a measurable *resolution*. The resolution is a measure of how good a timer is, and more precise timers are commonly more used when available, and unless tasks only need a rough estimation of time.

Clocks are time measuring circuits that can be used by an operating system to precisely measure wall clock time (i.e., real time). A clock uses an internal clock counter that is incremented by a monotonic clock signal. Using this value, and knowing how often it is incremented (i.e., the clock frequency), software can measure how much time has passed since the clock was reset.

In bhyve [3], the following timers and clocks are of particular interest:

- Local Advanced Programmable Interface Controller [4] (known as **Local APIC**, or **LAPIC**).
- High Precision Event Timer [5] (known as **HPET**).
- Time Stamp Counter [4] (known as **TSC**).

A. Timer: LAPIC

The Local APIC described in the *Intel 64 and IA-32 Architectures Software Developer's Manual Vol. 3A* [4], Chapter 10, is a per-CPU programmable interrupt controller (i.e., aggregates multiple sources of interrupt and delivers them to a specific CPU core) that is equipped with an internal timer.

The LAPIC's timer is a software programmable 32-bit timer that can be used to time events and system operations. The timer settings are split among four registers:

- Divide Configuration Register.
- LVT Timer Register.
- Initial Count Register.
- Current Count Register.

The Divide Configuration Register is used to configure the rate at which the current counter value will be decremented. Possible divider values range between 1 and 128 (inclusively), in powers of two. The set value is used by a circuit to reduce the processor's bus clock by the set factor (e.g., the timer's frequency will be half that of the bus clock, when the divider is set to use a factor of 2).

The LVT Timer Register can be used to set the timer in either periodic, or one-shot mode. Periodic mode means that the timer will rearm itself after expiring. In one-shot mode the timer will remain stopped after firing once. The register also determines which interrupt will be delivered to the processor.

Current Count Register and Initial Count Register are used together. When the Initial Count Register is set, its value will be copied to the Current Count Register. At every clock cycle of the timer, the value in the Current Count Register is decremented. When the value reaches zero, an interrupt is sent to the processor. If the timer is set to run in periodic mode, the value of the Initial Count Register is copied again and the cycle restarts. Setting the Initial Count Register to 0 will stop the timer. Figure 1 shows the workflow of the LAPIC timer.

To virtualize the Local APIC, bhyve traps access to the memory mapping of the device and updates the internal device state if required. Whenever the timer Initial Count Register is programmed, or a periodic timer expires, a `callout` [6] is set by the virtual machine manager, based on its internal state (i.e., the values of the four aforementioned registers) to program a timer on the host. When the timer expires, an interrupt is sent to the guest to signal that the time is up. In the case of periodic timers, another timer is set, and the cycle repeats.

Timer deadline for `callout` is computed as a function of system uptime, as required by the interface. More specifically, the timer frequency is computed as the fraction between the predefined value of frequency and the value of the divider. Since the timer is programmed using relative time (i.e., how much time should pass since it is programmed until it expires), when the Initial Count Register is set, simply adding the current system uptime with the product of timer frequency and the set counter will give the time when the timer should expire.

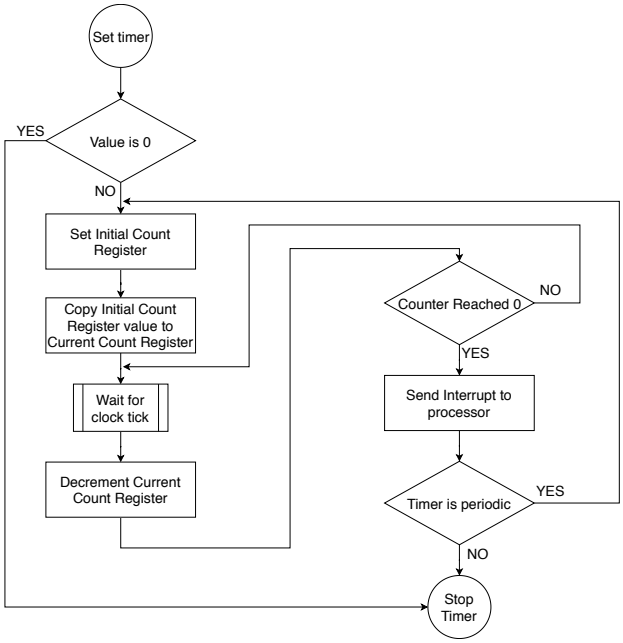

Fig. 1. Local APIC timer workflow.

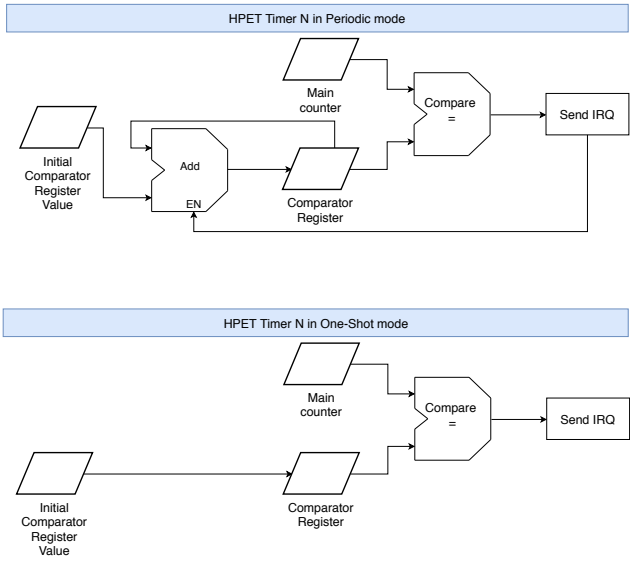

Fig. 2. HPET timer logic.

B. Timer: HPET

The High Precision Event Timers [5] are hardware timers developed by Intel for their processors. The timers use a 64-bit main counter that is incremented and an implementation-defined number of timers, with a minimum of 3.

Functionally, the timers use comparators to see when the main counter has reached a certain value. The value used for comparison is stored in a "match" register that can be either 32 or 64-bit wide. The value of the main counter is compared with the reference value using N-bit (with N being either 32 or 64, depending on the implementation) comparators and whenever the value compared matches the value of the value of the counter, an interrupt is generated.

The timers can function in both one-shot and periodic modes. In periodic mode, the comparator value is set to `value(base_counter) + value(comparator_register)`, so a new interrupt is sent every `value(comparator_register)` ticks.

The frequency of the HPET timer can be much lower than that of the LAPIC (i.e., the specification document [5] imposes that it should be at least 10MHz, while the LAPIC runs at the same frequency as the processor, unless divided), so it is less precise.

Since the HPET has the main counter, a monotonically incremented counter value, it can also be used as a rougher granularity clock source.

Figure 2 is a representation of the logic used to implement one of the timers in HPET.

HPET virtualization relies on device memory mapping to identify register access and update internal device state. HPET timers are emulated using `callout` [6] structures, one for each timer. The guest can set and get values for the main counter and timer counters. For virtualization purposes, the value of the main counter is set once, and reading it will simply add the value set by the guest and add it to the time elapsed since it was set divided by the frequency. Using this mechanism, the value of the counter can be precisely computed without using periodic timers to increment it.

Each timer is virtualized by programming a callback structure to the time when it should expire. However, since the timers programming relies on absolute values, rather than relative, the "current" value of the main counter will have to be determined and subtracted from the value of the comparator to obtain a relative time in the future when the timer must expire. Similar to the way LAPIC timers are set, the moment in the future the timer will expire is computed as the sum between the current time and the number of relative "ticks" of the HPET timer multiplied by the its frequency.

For periodic timers, after the timer interrupt is sent to the guest, another `callout` will be set to expire after the another time period has passed.

C. Clock: TSC

The Time Stamp Counter, as described in Section 17.15 of the *Intel 64 and IA-32 Architectures Software Developer's Manual Vol. 3A* [4] is a per-CPU internal counter that is incremented at the same rate, regardless of CPU frequency changes. The constant rate means that TSC can be used as a wall clock timer (i.e., measures real time, as opposed to how much a process has been running on the CPU). It is possible to adjust the value of the Time Stamp Counter using the `wrmsr` special instruction with appropriate offset.

The value of TSC can be read using the `rdtsc` instruction without requiring special privileges in the operating system (e.g., for Unix systems the command will not run as `root`).

Using this mechanism, software can determine how much time has passed from a reference point by computing the absolute difference of the two values.

An extension to TSC, called Invariant TSC, will guarantee that the value of the TSC counter will continue to increment while the system moves to power saving states. However, this behavior is not supported on older CPUs, so TSC may not be as stable as HPET on all systems.

TSC virtualization relies on the hardware extensions provided by modern Intel and AMD processors. This article refers to Intel specific extensions, but AMD offers very similar functionality.

Since the value of TSC is directly provided by the processor, its value is shared between host and guest. Because of this, the guest cannot be allowed to directly change its value; in stead, the virtualization extensions provide two special registers: TSC offset and TSC multiplier, as described in Section 25.3 of the *Intel 64 and IA-32 Architectures Software Developer's Manual Vol. 3A* [4].

Considering that bhyve does not use the TSC multiplier, the TSC offset is used when the guest attempts to access the TSC register as follows:

- **write** - the TSC offset is set to the difference between the value desired by the guest and the current host TSC value.
- **read** - the sum between the value of the TSC offset register and the current host TSC is returned.

III. Block Devices Virtualization

The Advanced Host Control Interface (AHCI) [7] is a PCI class device used to transfer data between system memory and SATA devices. Using the AHCI device, the system can enqueue multiple requests to a single device, with the possibility to aggregate requests to reduce disk wear and improve performance.

AHCI can connect multiple hardware devices to the system CPU, offloading the CPU workload through DMA transfers. By offloading the AHCI device, the system can asynchronously send transfer commands to I/O devices. At most 32 ports (i.e., physical connections with other devices; more than one device can be connected to a single port using port multipliers). Each port must function independently from one another.

By intercepting accesses to AHCI through memory mapping, bhyve [3] is able to identify when the guest is attempting to access the virtual disk. The virtual machine monitor can intercept guest memory accesses to detect when the guest attempts to program the AHCI controller for data transfers. Since requests are in a standard format the host can then decode the requests sent by the guests, by interpreting the commands sent.

Device virtualization implements asynchronous operation by offloading requests to additional worker threads with common work queues. Currently, bhyve uses eight workers in a generic block interface shared with the VirtIO [8] block device.

The device is emulated by translating the interpreted commands into I/O requests that can be executed on the host side. To emulate the AHCI's multiple command capabilities, reads and writes can be combined using FreeBSD's `readv` [9] and `writev` [10] system calls. Through the use of I/O vectors (as described by the manual pages of both `readv` and `writev`), numerous data transfers between the disk and guest memory can be executed using single system call, reducing device emulation complexity.

It is worth mentioning that using I/O vectors is only possible because the guest's memory is mapped directly into host memory space, such that data access to a specific memory address will be seen by the guest as if the device has completed the requests.

IV. Related Work

Bhyve [3] already featured a partially functional save and restore feature capable of resuming a guest running FreeBSD from a snapshot. The guest ran using VirtIO [8] network and block devices, and tests showed that it was able to connect to the internet, continue running timers, and read data on the disk after being restored. Obviously, testing advanced features like timers, network, and disks, would not have been possible if the guest CPU and memory were not properly restored before.

A regular test script to check if the internet connection and timers work correctly is shown in listing 1. If the script continued running properly after restoring the virtual machine, both the network connection and timers would have to work properly. Since `sleep` relies on a timer to finish its work (and so the contents of the `while` loop would execute), general timer misbehaviour would easily be spotted, since the command would not finish in time.

```
while sleep 1; do
  ping -c 1 8.8.8.8
done
```

Listing 1. Network and Timers testing.

To test disk functionality, simply being able to read a file from the virtual disk without any errors, and without visible data corruption was considered enough.

Since the tests performed on a FreeBSD guest were usually successful, the devices were considered functional. However, after changing test parameters, such as the guest operating system and attempting to restore after the host was restarted, a number of issues started to become apparent.

Linux and Windows, as opposed to FreeBSD do not communicate with the host directly through a serial console, but in stead use a frame buffer where they output a graphical interface, seen as either a classical CLI or GUI, and receive input from emulated xhci (i.e., USB) mouse and PS/2 keyboard. To properly communicate with the guest, these interfaces (frame buffer, xhci and PS/2) also had to be saved and restored, but their implementation is outside the scope of this paper and will be considered as de-facto functional.

The Linux guest had mostly the same functionality as the FreeBSD guest, but when running `dmesg` after a VM restore,

multiple filesystem operation errors on log files were displayed. This showed that despite being seemingly functional for files that were unchanged near the time a checkpoint was created, frequent background file I/O lead to data corruption. By using the same disk image multiple times to suspend and restore the virtual machine (as a special case of snapshot, that stops the guest after the snapshot is taken), after a few (around 4-5) iterations the disk usually became corrupted enough to render the kernel or core utils binaries unusable.

Improper functionality of the block devices was caused by how handling of incomplete requests was done. Before the snapshot of the virtual machine is created, the guest CPUs are frozen, and thus any interaction such as notifications that disk operations have finished were lost.

Windows guests running Windows Server 2016 (both with, or without, the full GUI) were completely frozen after restore. The issue with Windows, as we have assessed it, is that the user interface is directly linked to the system functionality, so if the interface is unable to properly update, applications with any form of interface would also stop working.

Moreover, a seemingly inconsequential difference, rebooting the host, would cause a kernel assertion to fail, and so the host system would crash whenever trying to restore a virtual machine. This was caused by the fact that the virtualization of some devices (e.g., HPET) uses the `callout` interface which relies on system uptime. Attempting to restore the value of the device state variables that kept track of system uptime usually meant that they would usually be reset to an "earlier" time (i.e., the host system at restore time had less uptime than it had when the snapshot was created), and an assertion meant to make sure the timer expiry date would not go backwards would fail, crashing the host.

The referenced behavior can be seen in listing 2, where the `KASSERT` instruction fails. `vhpet->countbase_sbt` is a variable set when the device emulation starts to the (then) current system uptime. Simply restoring it to its previous value can mean that its value is higher than the value of `now` (actual system uptime), thus resulting in a negative value of `delta`.

```
val = vhpet->countbase;
if (vhpet_counter_enabled(vhpet)) {
  now = sbinuptime();
  delta = now - vhpet->countbase_sbt;

  KASSERT(delta >= 0, ("vhpet_counter:"
    " uptime went backwards: "
    "%#lx to %#lx",
    vhpet->countbase_sbt, now));
  val += delta / vhpet->freq_sbt;
  if (nowptr != NULL)
      *nowptr = now;
}
```

Listing 2. HPET Virtualization Assert.

V. Save and Restore Debugging & Improvements

A. Time Management

Before being able to debug the more complicated issues with bhyve, the issue with the HPET virtualization that was shown in section IV had to be addressed.

Please note that issues described in this section refer to the manner a virtual machine behaved when restoring its state shortly after a host system reboot, when the system uptime is small. The virtual machine in all test cases is suspended when the snapshot is created, instead of having it continue running, since the disk currently does not support any copy-on-write mechanisms (e.g., qcow2, or similar).

Since the reference value used by HPET could not be restored as-is, if the timer was enabled prior to the virtual machine snapshot save, it would be reset using the current system uptime. While this did not solve any issues related to the guest's stability, it did stop the virtual machine from crashing the host, and allowed us to further guest behavior inspection.

While the host would no longer crash, the guest operating system did not work as expected: the `sleep 1` command did not finish after one second as expected, but rather after a seemingly random interval (the exact amount of time was not always the same). Moreover, trying to read the system time using `date` in this interval always returned the same value regardless of how much time had passed since the guest was restored. The reason for the inconsistent intervals after which the guest would resume functioning was caused by the fact that its timers were not restored.

As described in section II, the Local APIC virtualization relies on the `callout` [6] system to set a timer to expire Initial Count Register units (multiple of a base clock) into the future. At any point, a non-virtualized LAPIC keeps track of the amount of time until the timer must expire using the Current Count Register. Consequently, the snapshot mechanism saves the value computed for the CCR when virtual machine state is committed to disk (i.e., to not keep a separate counter that is decremented at a set frequency, the value is computed as a function of the value of the ICR and how much time has passed since the timer was set). When restoring the virtual machine, the timers that were set before the snapshot was created are reprogrammed using the value of the previous CCR.

Correctly saving and restoring the LAPIC timers resulted in a more stable, albeit incorrect, guest functionality. To be more precise, the system uptime still refused to update, but the intervals became more stable. Since the length of such intervals seemed close to the amount of time the host ran before the snapshot was created, the Time Stamp Counter and HPET were approached.

For the virtual machine, unless explicitly offset, the perceived value of the counter is the same as the one on the physical host. This means that if the virtual machine is restored after a system reboot, the value of the counter it reads may be smaller compared to a value it read before the snapshot

was created, because the reset value for TSC is 0. In turn, this implies that trying to determine how much time has passed by subtraction, using a value read before the snapshot, and one read after the restore would result in integer underflow (i.e., the value of TSC is a 64-bit unsigned integer).

TSC virtualization is done entirely by the virtualization extensions provided by the CPU manufacturer, and relies on their specific implementation. As such, the save and restore feature was added as a CPU-specific functionality, currently implemented for Intel CPUs only.

To keep track of both the offset set by the guest and the one required to compensate for system uptime differences between save and restore, two offset variables have been added for each virtual CPU: `guest_off` and `restore_off`.

The `guest_off` variable, which is used to keep track of the offset imposed by the guest is set when the `wrmsr` instruction with the proper offset is intercepted.

For the second offset variable, `restore_off`, the value of the counter is saved when the snapshot is created, and the offset is computed by subtracting the value of the counter at restore time from the value that was previously saved. If a new snapshot is created from a restored virtual machine (i.e., it runs as a result of the restore operation), the values of the restore offsets are added together.

By adding the values of `guest_off` and `restore_off`, and setting the result as offset for the Time Stamp Counter, the guest is not affected by the snapshot operation.

Despite no longer being stuck reading the same value for the time / date, the guest did not run as expected. When the guest finished restoring, a message could be seen in `dmesg` saying that TSC was an deemed as unstable and replaced with HPET.

Linux uses different counters as clocksources, including the HPET main counter and TSC. Since TSC is considered a precise clocksource, it is usually preferred over the more coarse alternatives (e.g., HPET). However, since the behavior of TSC is not guaranteed to be the same on all systems, less precise, but stable clocks are used to check if TSC has deviated, or not. If intervals measured using the two clocksources differ by an amount greater than a threshold, the next best source is selected to replace TSC.

In this case, however, TSC was the clock running properly, while HPET was incorrectly restored. To snapshot HPET, the value of the main counter is saved and used as offset after the restore, and added to the value of the current counter.

B. Block Devices

To solve the issues with losing notifications by sending them to a frozen host, when a snapshot request is processed, the emulated devices are paused before freezing the guest CPUs. Pausing is handled by the same thread that handles the snapshot, so synchronization with the worker threads is done to ensure that the worker threads will not undertake any more work while the device is paused, and the pause functionality will not end until all workers have been paused.

By pausing the devices while the virtual machine's CPUs are unfrozen means that the guest will not miss notifications for completed requests.

Furthermore, because the workers are unable to complete any tasks while the device is paused, new requests sent by the guest are also saved and restored as part of the snapshot process.

VI. Results

The stability of guests has been greatly improved as a result of fixing the issues of time management and disk virtualization.

Since the virtual disk is only implemented to use raw disks, and no mechanism to take disk snapshots is in place, the virtual machine tests are only performed on suspend and restore scenarios. If the guest would instead be allowed to run after the snapshot is created, the state of the virtual machine disk would not be the same as when the snapshot was created.

A. Linux Guest

The Linux guest performs properly after a restore in the following scenarios, where the virtual machine was suspended and restored while the respective operation is underway:

- sleep for a predefined interval.
- copy large files in the guest.
- copy files from a guest NFS shared directory to the host.

The sleep test scenario is meant to prove that the errors with timekeeping have been solved. This is the simplest test, since it does not rely on any other device functionality to be done - the binaries do not need to be read from disk after the restored since (if the virtual machine has enough memory) they are mapped into memory. Also, other devices like network are also not required to work since all operations are local.

Creating a copy of a larger file in the guest tests the virtual disk save and restore. As an example, a 4GB file with random data is created using the command in listing 3.

```
dd if=/dev/urandom of=test.ref bs=1G \
   count=4
```

Listing 3. Creating a file with random data

The testing of the disk can only be currently done using AHCI virtualization, since the VirtIO block device does not have the pause and resume functionality implemented. If the virtual machine is suspended while the copy operation is underway and the disk loses any requests, the copy of the file differs from the original. An observed result is that if the requests are not restored, some chunks of the file will only contain bytes of value 0.

The last experiment tests mostly timers and the network. Since the NFS protocol accepts a large enough system down-time from any of the ends, the guest can be suspended while a file is copied from the guest to the host, and the operation must resume when the guest is restored. As is the case with the other file test, large files with random contents are used to assess performance because any inconsistencies are easily spotted.

As a side note, the log file errors seen in `dmesg` described in section IV no longer showed after safely restoring the disk.

B. Windows Guest

For the Windows guest the following test scenarios have been considered:

- interaction with the guest is possible after restore.
- sleep for a predefined interval.
- copy files from a guest samba (SMB) shared directory to the host.

The first point should, obviously, be a prerequisite for testing any other advanced functionality, but the graphical user interface was completely frozen after a restore, due to errors in the way time management was saved and restored. Since Windows did not provide much information about the reason it didn't work, all debugging was done on Linux, and thus, Windows was simply a "bonus" that confirmed that timers and clocks are working.

For the second and third scenarios, the implications and testing methodologies are similar to those presented for Linux, with the exception that SMB has a smaller window for system downtime. As a result, `rsync` may show an error when the transfer reaches 100%, but the copied file proves to be identical to the source.

VII. Conclusions and Future Work

The virtual machines are more stable, and all three operating systems of interest - FreeBSD, Linux and Windows are able to run. However, a number of issues will be addressed:

- separate snapshot and migration code - the current implementation has mixed code for virtual machine snapshots and warm migration (i.e., moving the saved data from one host to another through a socket).
- the offset for TSC is unnecessarily split between two variables - in case the guest sets the offset explicitly, the offset should be computed relative to the current value of TSC on the host, and discard the restore offset entirely.
- the offset used by HPET should be set to 0 when the guest explicitly sets the value of the counter - the set value should be used as the counter value, regardless of whether the guest runs as a result of a restore or not.
- refactor the snapshot code - the generic code currently iterates through all devices to check if it is used, so it will be snapshot; additionally, most of the device-specific code for save and restore is the same (i.e., fields are saved one-by-one and in the same order, so the difference lies in whether the snapshot buffer is written or read from).
- extend the block device pause and resume functionality to the VirtIO block device
- add suport for TSC on AMD CPUs

Acknowledgments

A special "thank you" to Sergiu Weisz and Elena Mihailescu for their continued support in implementing the code and debugging all issues that we have encountered.

We would like to thank Marcelo Araujo and Matthew Grooms for their insight while debugging and structuring the code.

We would also like to thank iXsystems and Matthew Grooms for their financial support.

References

[1] Daniel P. Bovet and Marco Cesati. "Timing Measurements," in *Understanding the Linux Kernel*, 3rd ed., O'reilly & Associates Inc., 2009

[2] Jonathan Corbet, Alessasndro Rubini and Greg Kroah-Hartman. "Block Drivers," in *Linux Device Drivers*, 3rd ed., O'reilly & Associates Inc., 2005

[3] FreeBSD Project. *Freebsd as a Host with bhyve*. [Online]. Available: https://www.freebsd.org/doc/handbook/virtualization-host-bhyve.html [Last accessed December 18th, 2018].

[4] Intel Corporation. *Intel 64 and IA-32 Architectures Developer's Manual: Vol. 3A*. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html [Last accessed November 25th, 2018].

[5] Intel Corporation. *Intel IA-PC HPET (High Precision Event Timers) Specification*. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/software-developers-hpet-spec-1-0a.pdf [Last accessed December 5th, 2018].

[6] FreeBSD Project. *FreeBSD Kernel Developer's Manual - callout*. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=callout&manpath=FreeBSD-12.0-RELEASE&arch=default&format=html [Last Accessed January 1st, 2019].

[7] Intel Corporation. *Serial ATA Advanced Host Controller Interface (AHCI) Rev. 1.3.1*. [Online]. Available: https://www.intel.com/content/www/us/en/io/serial-ata/serial-ata-ahci-spec-rev1-3-1.html [Last Accessed January 1st, 2019].

[8] OASIS Committee Specification 04. *Virtual I/O Device (VIRTIO) Version 1.0*. Edited by Rusty Russel, Michael S. Tsirkin, Cornelia Huck, and Pawel Moll. 03 March 2016. [Online]. Available: http://docs.oasis-open.org/virtio/virtio/v1.0/csprd01/virtio-v1.0-csprd01.html [Last accessed January 2nd, 2019].

[9] FreeBSD Project. *FreeBSD System Calls Manual - readv*. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=readv&manpath=FreeBSD-12.0-RELEASE&arch=amd64&format=html [Last accessed January 10th, 2019].

[10] FreeBSD Project. *FreeBSD System Calls Manual - writev*. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=writev&manpath=FreeBSD-12.0-RELEASE&arch=amd64&format=html [Last accessed January 10th, 2019].