Safe Inspection of Live Virtual Machines

Sahil Suneja
IBM T.J. Watson Research
suneja@us.ibm.com

Ricardo Koller
IBM T.J. Watson Research
kollerr@us.ibm.com

Canturk Isci
IBM T.J. Watson Research
canturk@us.ibm.com

Eyal de Lara
University of Toronto
delara@cs.toronto.edu

Ali Hashemi
University of Toronto
ali.b.hashemi@gmail.com

Arnamoy Bhattacharyya
University of Toronto
arnamoyb@ece.utoronto.ca

Cristiana Amza
University of Toronto
amza@ece.utoronto.ca

Abstract

With DevOps automation and an everything-as-code approach to lifecycle management for cloud-native applications, challenges emerge from an operational visibility and control perspective. Once a VM is deployed in production it typically becomes a hands-off entity in terms of restrictions towards inspecting or tuning it, for the fear of negatively impacting its operation. We present CIVIC (**C**loning and **I**njection based **V**M **I**nspection for **C**loud), a new mechanism that enables safe inspection of unmodified production VMs on-the-fly. CIVIC restricts all impact and side-effects of inspection or analysis operations inside a live clone of the production VM. New functionality over the replicated VM state is introduced using code injection. In this paper, we describe the design and implementation of our solution over KVM/QEMU. We demonstrate four of its use-cases- (i) safe reuse of system monitoring agents, (ii) impact-heavy problem diagnostics and troubleshooting, (iii) attaching an intrusive anomaly detector to a live service, and (iv) live tuning of a webserver's configuration parameters. Our evaluation shows CIVIC is nimble and lightweight in terms of memory footprint as well as clone activation time ($6.5s$), and has a low impact on the original VM ($< 10\%$).

Categories and Subject Descriptors K.6.4 [*Management of Computing and Information Systems*]: System Management—Centralization/ decentralization; D.4.7 [*Operating Systems*]: Organization and Design—Distributed systems; C.5.0 [*Computer System Implementation*]: General; C.4 [*Performance of Systems*]: Design studies; D.2.8 [*Metrics*]: Performance measures; D.4.2 [*Storage Management*]: Virtual memory

Keywords Virtualization; Virtual Machine; Cloud; Data Center; DevOps; Monitoring; Inspection; Sandboxing; Live Cloning; Code Injection

VEE '17, April 08 - 09, 2017, Xi'an, China

ACM 978-1-4503-4948-2/17/04 . . . $15.00
DOI: http://dx.doi.org/10.1145/3050748.3050766

1. Introduction

Emerging DevOps methodologies are increasingly automating the entire development, deployment and operations lifecycle of cloud-native applications [88; 43]. However, with this end-to-end automation and an everything-as-code approach to lifecycle management, additional challenges emerge from an operational visibility and control perspective. Once a VM is deployed in production it typically becomes a hands-off entity in terms of restrictions towards inspecting it or adding new functionality to it. This stems from the risk associated with causing an intrusion or failure with an otherwise functional system that is running production workloads. If this risk can be mitigated, deep inspection of production VMs can lead to valuable insights that depend upon runtime state, and thus cannot be easily gathered during testing or staging of these systems.

Take the example of troubleshooting an application's runtime issues. In an ideal world, all bugs and issues would be ironed out in dev and staging environments. However, real-world application behavior is a function of both the underlying system and software, as well as runtime system state. Such state is impacted by incoming load, interaction with other running components, system configuration, and administrative policies, which introduces variability in live systems as opposed to pre-production environments. Despite best efforts, issues still arise in production VMs, where access is restricted. When that happens, the only resources available to fix the issue are logs, or, if the developer is really lucky, bug reports or stack traces. Operating in such constrained environment, without access to the actual systems containing the faulty runtime state, makes for an extremely

difficult and slow debugging and troubleshooting process. Similarly, there are other scenarios where operational visibility and control is desirable. For example, to catch anomalies proactively, to add on-the-fly new capabilities and monitoring hooks to a running system, or to simply understand how an application would perform under different configurations for a live workload.

Such live VM inspection presents significant challenges. The risk associated with making changes to live VMs is not unfounded, owing to unforeseen side-effects. On the other hand, recreating a given condition in a separate environment is both time consuming and challenging, due to it's significant dependence on runtime system state. These constraints thus necessitate an inspection environment that contains or operates on the live VM state, while safeguarding the VM from the impact and side-effects of its analysis operations.

In this paper, we present CIVIC (**C**loning and **I**njection based **V**M **I**nspection for **C**loud), a new mechanism that enables safe inspection of production VMs on-the-fly. CIVIC achieves this in two steps. First, it creates a live replica of the production VM (the *source*), including its runtime state, in a separate isolated sandbox environment (the *clone*). This liberates the source from the resource overheads, runtime interference and even installation of inspection utilities and their associated dependencies ('safe inspection'). Second, it uses runtime code injection to introduce new userspace-level functionality over the replicated VM state. This avoids enforcing any guest cooperation or modification ('on-the-fly' operation).

CIVIC differs with existing alternatives that can provide such execution-inspection decoupling in the following ways. It enables efficient reuse of the vast stock software codebase, overcoming the software incompatibility and functionality duplication effort with virtual machine introspection (VMI) based monitoring and inspection solutions [37; 42; 76]. Unlike most redirection-based solutions [73; 40; 89; 35; 34] that inject in-guest handler components (and are slow), CIVIC restricts all inspection/analysis operations to clones, and does not cause any guest intrusion and interference which would be unacceptable in a production VM. CIVIC enables introducing new post-hoc operations unlike other VM replication solutions, which do not support introducing new functionality [23; 49] or analysis at an OS or application-level semantics [30; 17] (more details in Section 6).

CIVIC employs code injection to achieve (clone) VM access, otherwise restricted in production environments. One alternative could be to enforce guest cooperation by sharing credentials, or granting login access. But this aggravates manageability concerns at cloud scale [35], as well as auditing [36] concerns by essentially giving direct control over even the source VM. Another access alternative could be via vendor-specific in-VM software components [85; 82; 84; 55] installed inside the guest. Even assuming that this would be acceptable for a production system, such guest specialization still defeats cloud portability [69; 48; 76]. Dependence on shell access may also complicate inspecting or troubleshooting dysfunctional systems. For example, in a recent Google outage [19], several production systems became unresponsive due to a dynamic loader misconfiguration. This prevented diagnosis using both: (i) the in-system agents now unable to emit data to backend, as well as (ii) login (SSH) not possible due to the inability to *exec* a shell. Such dysfunctional-userspace situations can potentially be tackled using CIVICs ability to inject code and troubleshoot from within the kernel itself.

CIVIC limits its inspection operations to within the clone, and does not target merging the clone's state back to the source. Propagating a validated procedure to the source is application-specific. One possibility is to push the analysis outcome (such as leak fix, validated patch, or tuned configuration settings) to the source using the same injection mechanism as in clone. Another alternative is making the clone the new source, or an interim source while the original gets updated with the validated procedure.

CIVIC's execution-inspection decoupling approach enables experimentation with possibly intrusive, heavyweight, or speculative post-hoc operations without the fear of negatively impacting the original guest system. We demonstrate four such use-cases in this paper. The first use-case is that of injecting and running monitoring agents inside the clone on behalf of the source, that are not desirable to install or run in the source VM itself. An example is the buggy agents in the Amazon Elastic Block Store (EBS) Service that caused severe EBS performance degradation [7]. The second example enables a risk-free exploratory diagnosis for a webserver memory leak. Instead of further degrading source webserver capacity by troubleshooting the leak within it, the problematic runtime cache state gets replicated, and debugging tools introduced, in a webserver clone. The third example attaches an anomaly detector to a live Cassandra [50] service on-the-fly, instead of baking it directly into the base service. The latter gets spared from the analysis' intrusion and interference, while the detector is allowed more aggressive analysis for improved accuracy. And finally, the fourth use-case shows how CIVIC clones *can* be employed to perform faster, risk-free and on-the-fly webserver configuration-parameter autotuning. Our evaluation shows CIVIC is nimble and lightweight in terms of clone activation time (6.5s) as well as memory footprint (30MB transfer size, 80MB resident set size (RSS)), and has a low impact on the source/guest VM ($<$10%).

This paper makes four contributions. First, an approach to use clones as inspection-enabling proxy systems. Second, an end-to-end implementation of CIVIC on the KVM/QEMU hypervisor, that combines the principles of post-copy live migration [41], disk and memory copy-on-write (COW), peripheral hotplugging, and code injection. Third, a novel

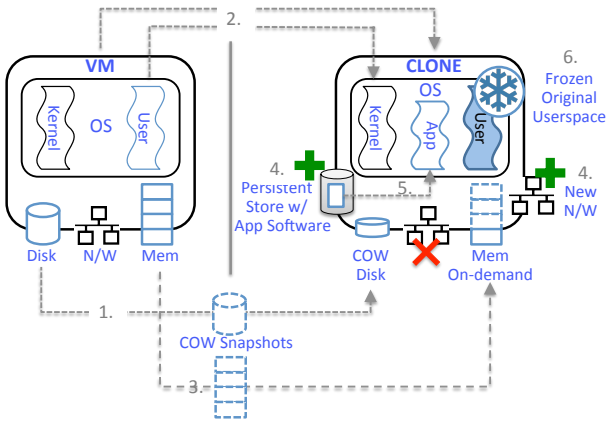


Figure 1: CIVIC's architecture; step-by-step description in Design Section.

mechanism for running userspace code inside the clone through kernel code injection from the hypervisor, which is useful for troubleshooting VMs with a dysfunctional userspace environment such as a non-responsive SSH. Fourth, porting stock software atop CIVIC. and demonstrating with four use-cases its ability to monitor, inspect, troubleshoot and tune unmodified VMs.

2. CIVIC's Design

CIVIC builds on top of whole-system replication and live cloning constructs [23; 49] to swiftly create a low-overhead clone of the guest VM, that acts as an inspection enabling proxy system containing the original guest's runtime state. The desired application functionality is then added to the clone through runtime code injection from the hypervisor into the clone's kernel. This hotplugged functionality can range from realtime operations such as periodic system health check by injecting monitoring agents in the clone (Section 5.1), to deep inspection such as diagnosing root cause of memory leaks by introducing debugging tools into the clone (Section 5.2).

CIVIC's emphasis on not interfering and impacting the guest system operation, not requiring any sort of guest modifications or cooperation to access the target systems, translates to the following set of challenges:

- **Consistency:** Since the clone is independent in its functioning and can deviate from the source, ensuring consistency requires: (i) preventing possible IP/MAC network conflicts, (ii) preventing clone's modification to the disk state from being reflected on the source and vice-versa, and (iii) (optionally) preserving process state on clone so that its inspected state refers to the source's runtime state at the time of clone creation.
- **Performance:** The performance requirements include: (i) minimal impact on the source's workload while forking a live clone, (ii) low clone initiation latency especially for realtime applications such as monitoring, and (iii) controlled memory consumption by the clone, limited to only what's required for a particular analysis application, as opposed to a full blown copy.
- **State Persistence:** A clone is a point-in-time image of the guest VM, and there may be several cloning rounds depending upon the inspection periodicity. In many cases, successive iterations may require intermediate state from previous ones, such as configuration settings, log files, and licenses, which presents the challenge of persisting injected application's state across cloning rounds.
- **Application Entry and Initiation:** Yet another challenge is introducing the desired application software into the clone. Depending upon the use-case, this could refer to a stock monitoring agent, some debugging tool, or simply a root shell capable of accessing files, listing processes and connections. However, we cannot expect or enforce all corresponding software components–binaries, libraries, helper packages, etc.–to be resident inside the guest. Even if new application software was to somehow find its way inside the clone, since we assume no guest cooperation as well as no artifacts in guest, this poses a challenge in initiating the application without the existence of any helper scripts inside the guest, or the login credentials for clone shell access in the first place.

To address these challenges, CIVIC employs the following sequence of operations, as illustrated in Figure 1:

1. Disk COW: A snapshot of the source's disk image is preserved at the time of cloning, and a COW slice is created for the clone, to ensure consistent disk access for both VMs.

2. Live Migration: The clone in CIVIC is created by live migrating a snapshot of the source VM. A postcopy migration technique [41] is employed which allows fetching memory on demand (pull-based approach) instead of transferring the full source VM memory at once (push-based approach). CIVIC does not enforce any restriction on the placement of clones– either locally on the same host as the guest-VM or on a separate host, the choice dependent on user preference, host resource constraints, as well as the target use-case.

3. Memory COW: Since in CIVIC's adoption of postcopy migration the source remains active, we also need COW memory on source. This is to ensure that the clone's on-demand memory accesses remain consistent to the source's memory state at the time of cloning.

4. Disk and NIC Hotplugging: In CIVIC, clone's modification to its COW slice are not preserved on its exit. This also circumvents the consistency and management concerns of preserving, merging and updating a previous clone's disk state modifications to a future clone. This is because a clone's disk view is based upon the source's disk state at the time of cloning, which might be different from when a previous clone was created. To enable state persistence for inspection applications, all clones of a particular source are

hotplugged with the same additional disk referred to as a *persistent store*.

For network consistency in terms of preventing IP/MAC conflicts, the source configured networking is disabled on the clone. However, there are cases where the clone requires network access such as for communication exchange (e.g. monitored data) between an application and its backend, and when traffic is partially or fully duplicated to the clone for tasks such as network filtering, runtime diagnostics and performance tuning. For such cases, the clone is hotplugged with its own NIC. Section 2.1 discusses network management further.

5. Code Injection: With the clone VM set up, the next step is to introduce the inspection application into its runtime. To enable this, the hotplugged *persistent store* from the previous step also acts as a storehouse of all application software components- binaries, related libraries, helper packages, config files, and an *application loader script*. Then, code injection from the hypervisor into the clone's kernel transfers control to the loader (shell) script, which runs inside the clone's userspace, thereby enabling the usual OS-exported functionality to be leveraged by the application software. It sets up the clone OS' operational environment, and initiates the desired application. Section 3.6 details the script's operations such as optionally freezing the replicated userspace in the clone to preserve its state for inspection.

2.1 Discussion

Usage Scope: CIVIC's execution-inspection decoupling approach enables hotplugging analysis to a live system, but without the associated cooperation, intrusion and interference hassles. Isolating inspection impact in the clone allows for a wide range of potentially heavyweight, intrusive or speculative operations like: (i) *systems monitoring* such as compliance scans and virusscanning, (ii) *sandboxing:* experimenting with different procedures and applying the optimal configuration on to the original system. For example, patch validation [81], live diagnostics and remediation [71], (iii) *deep inspection* such as application debugging, analysis and optimization [17; 21], etc., (iv) *proactive analytics* such as network filtering [15], malware detection [40], and fault injection [51; 47].

Network Management: Production environments typically have frontend networking nodes, like load balancers, forwarding proxies or SDN controllers, that filter and redirect incoming packets to backend systems. This is the registration point of a CIVIC clone (hotplugged with its own NIC), so that incoming packets to the source VM can be mirrored to the clone. Network spoofing may also be employed to let the OS or applications inside the clone keep believing they're using the same network configuration as before.

Privacy: Cloning a user's runtime environment can raise privacy concerns, but it provides the same kind of visibility and access as do VMI-based [37; 76] and redirection-based solutions [34; 35], or others that operate inside the original guest context itself. We argue in favor of the same kind of trust with the added advantage of guest operations remaining free of any intrusion and interference from inspection and tuning workflows, similar to VMI. Also, trusting the hypervisor is a fundamental assumption common to all such hypervisor facilitated solutions. This trust can be established using a self-serving cloud model [11], as well as cloud auditing [36].

Generality: While we implement CIVIC for Linux guests, it is equally applicable to Windows as well as Mac OSes. The only in-guest (clone-side) component to CIVIC's design is code injection, which has been shown to work for other OSes as well [16; 9; 58; 74].

Although we use GDB to aid with code injection, it is not a necessary dependency for CIVIC. Employing GDB simplifies the (static) code injection procedure such as modifying clone's memory directly, setting runtime break-points and corresponding commands to execute automatically, as well as locating addresses of the relevant kernel functions (alternatively can be read from *System.map* symbol table file). An alternative to GDB is to inject code through QEMU's dynamic binary translation engine [1] together with its Tiny Code Generation API [96] . Other hypervisor-based injection techniques also exist as employed by redirection-based solutions [40; 89; 35].

Also, CIVIC's injection mechanism is independent of the VM's OS version, using only standard kernel functionality, like the `schedule()` function as the clone interception point.

Limitations: Since CIVIC depends on access to the clone's memory from the hypervisor, it is not applicable for encrypted VMs, or those protected from hypervisor access [54].

Even with the appropriate access, CIVIC may not be suitable for all types of analysis due to the clone's deviation from the source. For example, any compliance scans or healthchecks are only good at the point of cloning, and not thereafter. Complete coverage may not be guaranteed during application debugging since not all external inputs (e.g., interrupts) are replicated, unlike record-and-replay solutions. A hardware fault on the source VM's host may not be present at the clone's end, which may change the behaviour of the in-VM application under inspection.

Another side-effect of the clone's independent existence could be when the source communicates with an external entity, say, a database backend. In this case, replicated requests from the clone may corrupt the backend. Depending upon the use-case these side-effects may be handled by: (i) dropping the clone's packets [29], (ii) freezing the corresponding process in the clone, or (iii) identifying the duplicates at the

[1] The injected code is static and does not *need* compilation at runtime; QEMU TCG serves only as another option to avoid any GDB-dependency. It might be possible to use QEMU directly to put breakpoints and redirect execution, without going through its gdb-stub.

backend by assigning a unique IP to the clone, or (iv) synchronously cloning all components of the system (e.g., the webserver and the database) together [45; 22; 70; 75], assuming the side-effect observer isn't the external world.

3. Implementation

CIVIC's prototype pieces together existing implementations (with slight modifications) of post-copy live migration, copy-on-write disk and memory, and device hotplugging. In addition to these building blocks, CIVIC incorporates a novel mechanism to run userspace code into the clone through kernel code injection from the hypervisor. The rationale behind these operations is already covered in the Design Section. These operations are orchestrated by a userspace bash script running on the host machine. Although our implementation is on the KVM/QEMU platform, these underlying constructs exist for the Xen hypervisor as well [93; 92; 61; 91; 94].

3.1 Disk COW

In order to ensure consistent disk access, on the clone side we use QEMU's redirect-on-write [2] feature to create a COW slice on top of the committed state so that clone's writes are redirected to a separate location than the original image file. On the source side, we use a simplistic approach of running the source VM in a QEMU `snapshot` mode, and enforce a write back (`commit`) of its disk state before cloning. A better alternative perhaps is to employ copy-on-first-write approach [31] on source wherein the source continues to write to the same disk image as before, but before a block gets overwritten, its original contents are saved to a separate location. Still, the explicit commit step above ensures that the proper impact on the source is incorporated in CIVIC's evaluation.

3.2 Live Migration

CIVIC achieves source VM cloning by employing post-copy migration [41], that fetches memory on demand similar to the VM Fork abstraction [49] in Xen. We use the latest QEMU post-copy implementation [24], which builds on top of Linux' `userfault` kernel feature [5] which enables handling memory page faults in userspace. QEMU uses this functionality to trap the clone's accesses to remote memory pages, fetch them from the source, and move them into the clone VM process' address space using the `remap_anon_pages` [2] system call [5].

We make two modifications to default implementation. First, while in a typical postcopy setting the original VM is paused while the migrated VM fetches memory from it, in CIVIC the source is allowed to resume operations as the primary/production VM (COW enabled for consistency, see next subsection), while the secondary migrated instance operates as the proxy clone. Second, the default implementation also has simultaneous pre-copy iterations that makes the clone's memory footprint similar to the source. Thus, to make CIVIC clones lightweight, we minimize these pre-copy transfers by only allowing the initial state fetch during clone initiation via pre-copy, and thereafter switching to pure post-copy i.e. memory-on-demand based fetches.

[2] `remap_anon_pages` syscall is now replaced by ioctls: `https://lkml.org/lkml/2015/3/5/576`.

3.3 COW Memory

The source memory COW implementation follows the common approach of saving the original memory pages to a holding area inside QEMU on source writes, and servicing clone's page-fetch requests based upon dirty flags, either from the source's memory directly for an unset flag or from the holding area otherwise. We use HotSnap's [22] approach to enable COW snapshotting in KVM/QEMU. This involves trapping different sources of writes to guest memory- write faults by guest, DMA accesses from QEMU, direct writes by KVM, and QEMU-internal writes. Depending upon the source of writes, we modify KVM to either inform QEMU to directly save the existing memory page contents before being dirtied, or save the original page contents locally and send them to QEMU via Linux' `copy_to_user()`.

3.4 Hotplugging

To enable IO communication on the clone with the outside world, as discussed in Section 2, the clone is hotplugged with its own NIC by using QEMU's `host_net_add` functionality, as well as a *persistent store* disk using QEMU's `drive_add` functionality.

3.5 Code Injection

We use code injection as a means to initiate inspection operations inside the clone, to avoid requiring (i) guest modifications in terms of installing helper scripts inside the guest, as well as (ii) guest cooperation in terms of login credentials to access clone's shell. The basic goal with code injection is to run an *application loader script* residing in the *persistent store*, inside the clone OS' userspace from the hypervisor. We achieve this with the following sequence of operations:

1. Attach to the clone's kernel using QEMUs GDB stub [64].
2. Inject machine code into an empty memory buffer. The code performs the following operations:
   (a) Save registers
   (b) `mount` hotplugged disk
   (c) `exec` application loader script
   (d) Restore registers
   (e) Return to caller
3. Break at clone kernel's `schedule()` function.
4. Redirect flow to injected code (replace instruction by `jmpq`)
5. Restore control flow (restore original instruction)
6. Detach GDB from clone.

We now detail the technical specifics required for some of the above-mentioned steps.

Memory buffer (Step 2): An empty buffer can be obtained by trapping or redirecting control to `kmalloc()` [16], or hotplugging extra memory to the clone [40; 95; 80]. For convenience, in our current implementation, we use an empty memory buffer at *16MB + 1 page* from the beginning of physical memory. Across different kernel versions, we found this empty space to be sufficient for the 290 bytes we need to inject code.

Mount and Exec (Steps 2b, 2c) : In order to use the filesystem hosted inside the *persistent store*, the hotplugged disk device must first be mounted inside the clone OS. Achieving this from kernelspace boils down to calling the `do_mount()` kernel function with the appropriate device, mountpoint and filesystem-type arguments set. Next, to run the *application loader* shell script from the mounted disk inside the clone OS' userspace, we invoke the kernel function `call_usermodehelper_setup()` with path to the loader script as an argument, followed by `call_usermodehelper_exec()`. The script can be run with real-time priority [53] for immediate execution.

Schedule() (Steps 3, 4, 5) : To preserve state consistency in the clone, we use the kernel's `schedule()` function as the control interception point, instead of hijacking the `system_call` handler as in [63]. Typical control redirection flow involves saving and replacing an instruction with a `jumpq` to the injected code, and restoring it thereafter. Although any instruction can be selected for replacement, we were simply able to leverage a `NOP` instruction (`xchg %ax,%ax`) in `schedule()`.

The injected code also includes instructions to disable/enable interrupts appropriately as needed by the different injected operations.

3.6 Application Loader Script

The previous stage results in an *application loader script* running inside the clone OS' userspace. Controlling the clone now, as an inspection enabling proxy for the source, is straightforward by utilizing OS-exported functionality. Specifically, the loader script performs the following tasks ('*once*' below refers to operations performed only for the first clone):

1. [*Optional*:] Pause userspace processes (`kill -STOP -1`).
2. Disable source configured networking (`ifdown`).
3. Enable networking on hotplugged NIC (`ifup`).
4. Setup clone runtime environment
   (a) [*Once*:] Mirror root partition (`'/'`) on hotplugged *persistent store* disk: automatically, via `chroot` (`yum --installroot`).
   (b) Update executable paths: prepend mirrored `/bin` to `$PATH`.
   (c) Update dynamic linker's run-time bindings: prepend mirrored `/lib` to `ldconfig`.
5. [*Once*:] Install application software and redirect paths in config files to the persistent store's `'/'` sub-tree.
6. Run application, either directly from the persistent store's `/bin` or via symbolic links in `'/'` pointing to persistent store's `'/'`.

The optional freezing of userspace processes enables preserving the source's runtime state for inspection inside clone. In Section 4 we show how we use this feature to track source's process-level resource utilization via clone as a proxy. The need for network reconfiguration is described in Section 2.

Mirroring the root partition inside the hotplugged disk enables state persistence, as well as dependency resolution along the lines of `chroot` jails [52]. Application software installation typically involves downloading dependency packages, and if the installation process is allowed to run as is, these helper binaries and/or libraries will be installed inside the standard '/' directory (`/bin, /lib`). Due to CIVIC's design, these changes will not persist across cloning rounds. One way of avoiding this is to require all dependency packages to exist (or be installed) in the source. But this enforces guest cooperation and modification which is against CIVIC's design principles. A better alternative is to instead redirect installation of all software components inside the clone to a mirrored root partition inside the hotplugged persistent store, and to update the linker/loader bindings and executable paths correspondingly. A positive side-effect with such `chroot`-like approach is that the source is spared from package pollution and potential dependency conflicts. The end product of this last CIVIC stage is a proxy clone completely set up for analysis and inspection on behalf of the source without any source intrusion and enforced cooperation.

4. Performance Evaluation

We evaluate CIVIC's performance by answering the following:

1. What is CIVIC's memory footprint in terms of clone's memory usage, memory transferred and COW-ed?
2. How much time does it take to get the clone ready, including time spent during migration, hotplugging, and code injection?
3. What is the impact on the source VM in terms of downtime and workload performance degradation?

Along with CIVIC clones we also create precopy clones, just to observe and quantify the savings that postcopy offers CIVIC in our setup.

Setup: The host is a 4 core Intel i5 @ 2.8GHz machine, with 16GB memory and Intel VT-x and EPT hardware virtualization support. The software stack includes Linux-3.19 host OS with KVM and `userfault` [8] support, QEMU 2.1.50

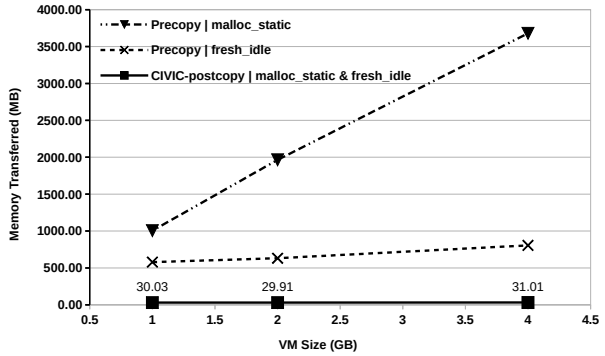


Figure 2: Measuring memory footprint of CIVIC's postcopy+COW clones and precopy clones, for different source VM sizes and memory use configurations.

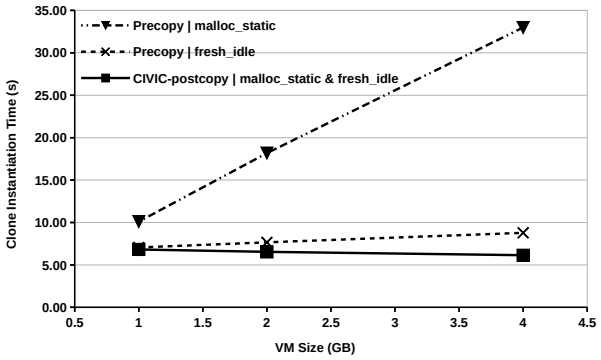


Figure 3: Measuring clone instantiation time for precopy and CIVIC's postcopy+COW clones, for different source VM sizes and memory use configs.

with postcopy migration support [25]. Guest VMs are configured with 1 CPU core, {1G, 2G, 4G} RAM, and Linux OS {3.2/Ubuntu, 3.4.4/Fedora}. The reported metrics in experiments below (and in Section 5) are averaged across at least 3 runs. The same host runs both source and clone VMs (full copy, no page sharing), with QEMU migration transfer rate set to 1Gbps. The host is assumed to have sufficient resources to run the clone VM. For high consolidation/contention scenarios, the clone can be run on a separate lightly-loaded host; the CIVIC orchestration script (Section 3) itself has negligible resource cost.

4.1 Memory Cost

We vary the memory load on the source while cloning it, and measure the clone's memory usage, memory transferred and memory COW-ed. The measurements are made across three different source memory load configurations: *fresh_idle*, *malloc_static*, and *dirty_dynamic*. The first configuration refers to a freshly booted idle VM, while the static and dynamic memory-use configurations touch about 75% of VM's memory with the latter continuously redirtying it. We use the `stress` utility [1] to achieve target memory usage in source.

For the first two source configurations, CIVIC clone's memory consumption (measured as resident set size (RSS)) is <=80MB, amount of memory COWed <=2.7MB, and amount of memory transferred during migration is around 30MB, irrespective of the source VM size. Whereas for *dirty_dynamic* configuration, all metrics are equivalent to the working set size i.e. ∼75% of the source VM size. This is when the memory dirtying workload is run as-is inside the clone VM as well, whereas in an alternate scenario the process would optionally be frozen in the clone leading to lean clones as in the other two configurations. Further reduction in the memory footprint for host-local clones can be achieved by augmenting CIVIC with a page sharing optimization.

On the other hand, as is to be expected, precopy clone's RSS and transfer size increases linearly with the source VM size as shown in Figure 2. Note that for the *dirty_dynamic* configuration, precopy migration over network can only complete when source's memory dirty rate is less than the network bandwidth (completed only for working set size <=35MB, in experiments).

Summary: CIVIC's footprint depends upon source's working set size; for scenarios (such as monitoring) that aren't heavily dependent on full system state, the clones are light weight with 30MB transfer size and 80MB RSS.

4.2 Clone Instantiation Time

Figure 3 shows how the clone instantiation time varies with the source VM size for the different source's memory-use configurations as described in Section 4.1. Compared are the end-to-end times including disk snapshotting, migration, hotplugging, and code injection costs, up until the *application loader script* execution inside the clone.

CIVIC's instantiation time is independent of the source VM size as well as its memory-use configuration, whereas the VM size affects precopy cloning time linearly. The results remain the same when the memory load on the source VM is replaced with a CPU intensive workload (*sysbench* prime computation [6]). It takes about 6.5s for CIVIC clones to get up and running, with the stage-wise breakdown being about 0.2s for VM initialization, 4s for disk snapshotting and hotplug, and 2.3s for code injection.

While the clone instantiation time can be reduced by having the *persistent store* and NIC for clone operations hotplugged (but inactive) ahead of time inside the source, but it introduces source modification which is against CIVIC's design principles. A better alternative is to reuse the same clone across successive rounds by only fetching the delta from the latest source state. This would take away the hotplugging and GDB overheads and lead to sub-second clones with an inspection-ready environment. This serves as a possible future optimization to CIVIC.

Summary: CIVIC clones are ready in at most 6.5s, irrespective of the source VM's working set size.

4.3 Impact on Source VM

To measure CIVIC's impact on the source VM, we periodically clone it while running the following three workloads individually inside the source: (i) x264 video encoding CPU benchmark [59] (v1.7.0) with ∼350MB of memory footprint, (ii) bonnie++ disk benchmark [65] (v1.96) processing 4GB of data sequentially, and (iii) full system webserver benchmark- httperf [57] (v0.9.0) serving distinct 2KB random-data files 512MB in working set size, to clients running on separate machines. Additional measures were taken to ensure true benchmark measurements, such as using high performance virtio drivers in the guest, disabling disk caching at hypervisor (and QEMU `snapshot`'s writeback caching for bonnie++), ensuring network isn't a bottleneck, and pushing webserver to saturation.

Each cloning iteration performs periodic monitoring tasks of compliance scan, healthcheck and resource monitoring (Section 5.1); by tracking open files and connections, loaded modules, running applications, system logs and resource utilization metrics. CIVIC's clone instantiation time (Section 4.2) limits the monitoring frequency to 0.1Hz. The average degradation on the source's workload was observed to be 5.2% on x264's framerate, 1.2% on bonnie++'s disk throughputs, and 10% on the webserver's maximum sustainable capacity, attributable to work queue backlogging [77] due to minor VM stuns (see downtime below).

In the case of monitoring, as well as inspection that isn't heavily dependent on full system state, a majority of source's memory would not get transferred over to the clone (memory-on-demand). Thus, to additionally account for higher-level analysis tasks like anomaly detection and autotuning (Sections 5.3, 5.4), we also let a cloned instance of the webserver operate in parallel with the source. For the case of httperf, during the cloning process both the source and the clone see 5-6% degradation on maximum sustainable capacity. Thereafter, both are able to operate at peak capacity as recorded in source pre-cloning.

Finally, for measuring VM downtime, we use `fping` to fire ICMP ECHO_REQUEST packets to the source VM with a 100ms timeout, and count the failed i.e. timeout-expired requests. The VM downtime was recorded to be 0.4 seconds.

Although the source impact is low for the 0.1Hz maximum cloning frequency supported by our current implementation, this seems to translate to a heavy impact for higher frequencies. For such high frequency use-cases, the 0.4s source VM stuns per cloning iteration would need to be minimized. Also, a lower impact can be expected with the potential optimization of reusing the same clone instance across cloning rounds (Section 4.2).

Summary: CIVIC has a low impact on the source VM, reaching 10% degradation with continuous (0.1Hz) cloning.

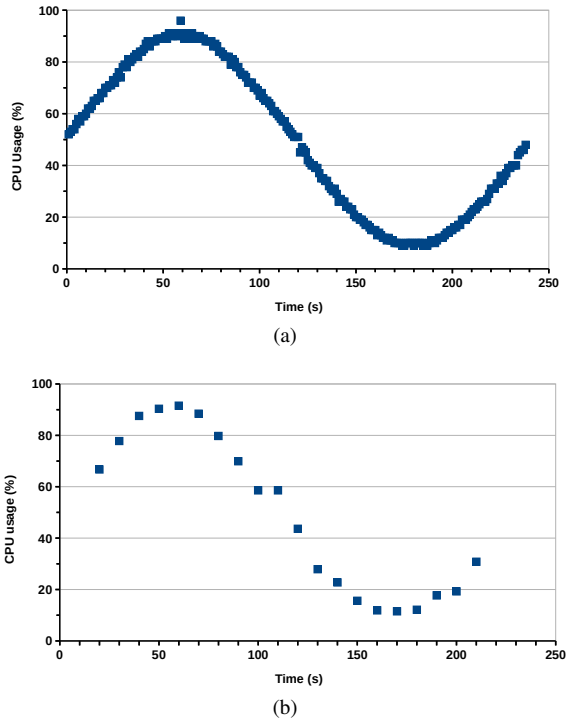


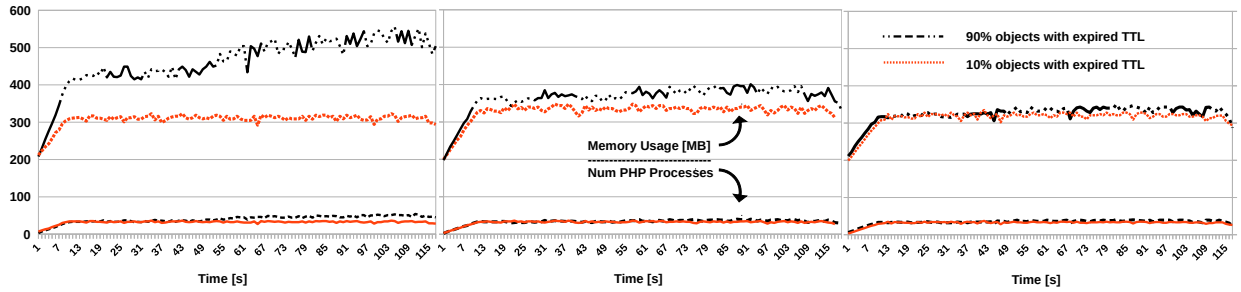
Figure 4: Measuring CPU usage with *collectd* in source (top), clone (bottom)

5. Applications

This section highlights CIVIC's versatility by describing how we have used CIVIC in different settings to facilitate a variety of inspection operations. These scenarios don't necessarily require CIVIC to operate, but in most cases CIVIC works better than other alternatives as described below (more details in Section 6). First, the VMI-based solutions [42; 76] would be incompatible with the stock software employed to address these scenarios. Second, the redirection-based solutions [73; 89; 35] would be very slow and cause guest intrusion and interference by running handler components inside the source VM, which is precisely what some of these scenarios attempt to avoid. Finally, most can be achieved by live cloning solutions [23; 49] by either directly accessing the clones via credentials/SSH, or through backdoors or hooks installed beforehand in the source VM [85; 84]. But the former approach hurts cloud manageability [35] and auditing [36], and wouldn't work when inspecting dysfunctional systems as in the Google outage example [19], while the latter approach defeats cloud portability [69; 48].

5.1 Safe Agent Reuse

Most companies require employees to run monitoring agents on their machines (VMs) to ensure compliance, check for vulnerabilities, or just monitor usage. Informal accounts from employees highlights their concerns against such heavyweight and/or intrusive but necessary agents.



(a) PHP 5.1.6 (b) PHP 5.3.20 (c) PHP 5.6.10

Figure 5: Count as well as memory usage of PHP processes in a webserver, for different proportions of cached data with expired TTL. Compared across 3 different PHP versions with memory leaks, fixed between v5.1.6 to v5.6.10.

There have also been reported cases, such as at Amazon [7], where bugs in agents caused severe performance degradation for the actual service being monitored. Although VM introspection (VMI) [37; 76; 42] can be used to monitor these systems non-intrusively, it requires extra functionality duplication effort (Section 6). CIVIC enables reusing the vast agent software codebase without the extra effort, while providing similar isolation benefits by virtue of restricting agents in the clone.

To provide simple visual evidence, we use process-level resource tracking as an example for agent-based monitoring. We have tested CIVIC successfully against three such agents- an internal custom agent, a closed-source enterprise-level agent- IBM Tivoli Endpoint Manager (BESClient [44]), and a popular open-source monitoring agent-*collectd* [32]. The agents were run as-is, with the config files updated to point to *persistent store*'s mirrored root partition for installing the agent software and associated dependencies.

In this experiment, we use *collectd* to track the resource use metrics for a custom source workload that varies its CPU and memory utilization sinusoidally. The workload gets frozen in the clone, with its runtime state analyzed by *collectd* injected on successive cloning iterations. Housing this agent (one of eventually many such) in the clone avoids installing up to 77 packages on the source. To illustrate the performance of a **CIVIC clone as a runtime monitoring proxy for the source**, Figure 4 compares the workload's CPU-usage tracking by *collectd* inside the clone, with the expected curve had the agent run inside the source (memory graphs similar; omitted for brevity). Section 4.2 discusses improving the clone's 0.1Hz monitoring frequency, in comparison to 1Hz as configured for in-source monitoring.

5.2 Problem Diagnostics and Troubleshooting

By replicating a troubled system's runtime state inside a clone, and introducing debugging tools in it, CIVIC enables risk-free exploratory diagnosis while absorbing the associated impact and side-effects. Since this use-case requires manual diagnosis, a root shell also gets injected for the dev, again useful when source system access is restricted.

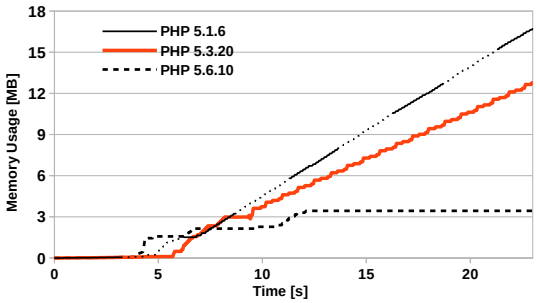


Figure 6: Measuring PHP process' memory usage via `strace`; Leaks detected in versions 5.1.6 and 5.3.20

We highlight this problem diagnostics use-case by capturing and fixing PHP memory leaks (by recreating bugs #45161 and #65458 [62]) in our custom webserver setting. Our *apache + php-fastcgi* webserver serves incoming user requests for data that it caches from a backend server (a database) based upon time-to-live (TTLs). When the TTLs expire, the webserver either fetches fresh data from backend, or otherwise renews their TTL until the next synchronization cycle. Figure 5(a) shows two different memory usage patterns for different proportion of data with expired TTLs- 10% and 90%, *without any fresh data being fetched from the backend*. Also, the difference in the count (Figure 5(a)'s bottom-most curves), and RSS for PHP processes in the two cases does not account for the memory usage explosion.

At this point, troubleshooting this apparent memory leak on the webserver VM itself could degrade its sustainable request rate by further polluting its memory cache and/or adding debugging/instrumentation load to the system (up to 20.5% capacity degradation with the diagnostics employed below). Furthermore, the production system might not have the instrumentation frameworks installed there and one wouldn't want to perturb the environment even more. To enable risk-free diagnosis, CIVIC replicates the problematic runtime cache state into a webserver clone, introduces diagnostics tools, and mirrors incoming requests to it.

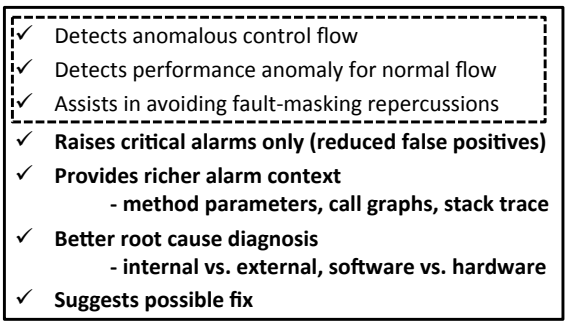


Figure 7: SAAD under CIVIC: enhancements on enabling debug mode in clones, in addition to stock SAAD capabilities (dashed box)

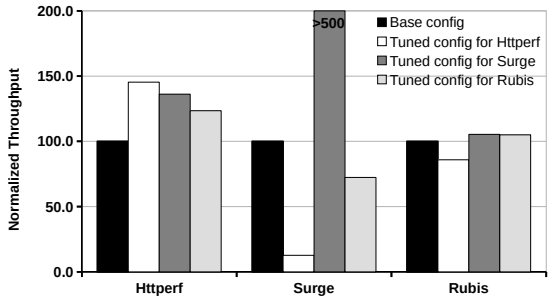


Figure 8: Webserver capacity variations with apache+kernel tuning

In this example, we attach `strace` to the apache/php processes in the webserver clone to measure their memory usage trends [79]. Figure 6 plots the memory usage across time across 3 different PHP versions- v5.1.6 used in the webserver experiment of Figure 5(a), v5.3.20 that fixed one leak, and v5.6.10 with no leaks. Although new data was not being fetched from the backend, simply communicating with it for TTL renewal was enough to activate the memory leaks (caused by cURL [4] reuse) Figure 5(b) and (c) plot the clone webserver's memory usage across time on changing the PHP versions. The latter setting gets reported to the source to fix the issue.

5.3 Anomaly Detection

CIVIC enables a model where secondary functionality is introduced as hotpluggable components, instead of baking it into the primary base service. Employing clones to provide the add-on functionality (such as a diagnostics framework) allows the latter to be as intrusive or destructive as need be, while also isolating possibly conflicting functionalities (or base service modifications) in their own separate environments (clones).

We highlight this capability by enabling the SAAD [38] anomaly detector to be hotplugged to a base Cassandra [50] service. SAAD serves as an example for a whole class of Java-based services that can be automatically ported over CIVIC by using JVM classloader level class-hotswapping.

Instead of users having to patch their Cassandra versions to get SAAD's functionality, CIVIC enables running an unmodified Cassandra service by adding SAAD-equipped clones to the Cassandra worker pool on-the-fly. To preserve a source's replicated runtime state in a clone (which will be lost on Cassandra re-instantiation with SAAD's version), we use JRebel tool [97] to automatically replace and reload running instances of stock Cassandra classes with their SAAD versions inside the clone. Requests to the source worker node are mirrored (either completely or partially), and redirected to the clone(s) under analysis.

Along with the source service being spared from analysis' intrusion and interference, CIVIC also improves SAAD's anomaly detection accuracy and quality. Figure 7 highlights SAAD's capability enhancement in a restriction-free operation mode, in this particular case by enabling debug-level logging on-the-fly in the clone. Such verbose logging is not recommended in a production system since it causes heavy performance degradation, such as a 25.8% impact on Cassandra's throughput in our experiments with the YCSB [20] benchmark. Similarly, CIVIC enables SAAD to incorporate potentially heavyweight syscall tracing in its analysis while limiting the associated overhead to the clone. This can improve SAAD's accuracy by additionally identifying anomalous methods, an example being a buggy infinite loop in Cassandra bug #5064 [13; 26].

5.4 Autotuning-as-a-Service

Webserver tuning is an error-prone, time consuming [67], computationally intensive, and highly workload-dependent task [18; 98; 100], with over 240 configurable apache parameters. As a final demonstration of CIVIC's usefulness, we propose to employ **CIVIC clones to perform faster, risk-free and on-the-fly webserver configuration-parameter autotuning.**

Using the context of Amazon cloud service, the users would simply need to deploy their webserver VMs on the cloud without having to predict how their traffic would look like, or worry about tuning their server themselves or via autotuners. With the webserver live, CIVIC would fork the clone(s), carrying over the webserver's runtime cache state onto the clone (otherwise costly to rebuild, see 'Instance Reproduction' in Section 6), followed by autotuner [27; 67; 100; 18] injection, and live input request replication and redirection to the clone. Furthermore, multiple clones can be employed in parallel for a faster, more aggressive and less time-constrained configuration space exploration.

To quantify the potential gains from such a model, we tune webserver configuration parameters for the following three workloads: (i) a streaming type Httperf [57] workload, (ii) a Surge [10] workload modeled to represent actual traffic serviced by commodity webservers, and (iii) a RUBiS [60] auction website workload modeled after eBay. In this experiment, the configuration space under tuning includes (i) apache (`httpd.conf`) parameters such as core multi-processing module type, `MaxClients, KeepAlive`

`Timeout, ListenBacklog`, etc., as well as (ii) kernel (`sysctl`) parameters such as `net.ipv4.tcp_rmem, vm.swappiness, fs.file-max` amongst others. For each workload, Figure 8 plots the webserver's capacity (normalized to the base system configuration) for 4 configurations-base, tuned configuration for the particular workload, as well the best performing configurations for the other two workloads. As can be seen, no single configuration works best for all workloads, and the tuned configuration for one may not be ideal for the other, leading to a performance range of 13% to over 500% of the base configuration in our experiments. Such extreme variations are not uncommon [3]. This experiment illustrates the need for tuning, and the potential benefits to be had with CIVIC-enabled autotuning.

6. Related Work

In this Section, we discuss other techniques that can potentially enable safe VM inspection.

Virtual machine introspection (VMI): VMI based monitoring and inspection solutions [37; 42; 76] operate outside the guest VM on a live runtime image of the VM's memory. However, the runtime view exposed in such a manner is of a raw byte-level form, requiring deep per-guest-OS kernel data structure information to reconstruct logical information. Additionally, it demands effort in terms of either exposing the entire OS like view (`/proc` etc.) for already existing software, or writing fresh tools using introspection directly. Tools for automatic generation of VMI-based utilities [28] are much slower than native execution, require in-guest training and expert intervention, and still remain incompatible with existing software. CIVIC, on the other hand, enables efficient reuse of the vast stock software codebase without requiring such effort.

Redirection-based solutions: These operate from a secondary context such a separate privileged VM or the hypervisor, and feed off of the guest VM's runtime state directly. This secondary context can be viewed as offering isolation, with access to the guest state being facilitated by employing techniques such as (i) process relocation outside guest [73], (ii) kernel data redirection to guest [33; 34; 66], and (iii) component implanting (syscall, function-call, module or process) inside guest [40; 89; 35; 16; 12; 86].

A major concern with most of these is that they inject handler components inside the guest VM itself. Such guest intrusion and interference would be unacceptable in a production VM. CIVIC, on the other hand, restricts all operations to clones. In many cases, only basic utilities (like *ps, lsmod*) are supported and that too with heavy performance slowdowns [33; 34; 89; 35], one reason being reliance on binary translation. In other cases, the solution is not transparent to the utility software [40], which needs to be made aware of a non-standard runtime environment (e.g., recompilation with static linking and hypercalls). CIVIC, on the other hand, supports complex stock software software with negligible runtime slowdown. Some solutions require separate OS fingerprinting to use the exact same version of the guest OS in the secondary VM [34; 66], while cloning gives that for free in CIVIC.

Instance Reproduction: These methods are capable of reproducing a problematic occurrence in another VM, thus providing greater diagnosis flexibility and isolation. These can be sub-classified based upon how the secondary VM is created from the primary- (i) a cold boot over base image copy, (ii) from point-in-time snapshot [83], (iii) live cloning with runtime state [23; 49; 78; 101; 56], or (iv) via record-and-replay [30; 72; 17; 46].

Copy booting suffers from full memory duplication, instantiation latency, as well state rebuilding which can be costly (e.g. up to 15 minutes to warmup caches or load data [14; 68]). VM snapshots, on the other hand, are useful for fast provisioning [39; 87; 102; 90]. But, for a secondary VM instantiated from a point-in-time snapshot, explicit state propagation, especially when the snapshot is not recent, costs additional developer effort, resource wastage (idle workers) and migration delays (consolidated idle VMs) [49]. Additionally, both of these categories may miss capturing exact point-in-time occurrences or anomalies. CIVIC circumvents all these concerns via on-demand replication of the source VM's live runtime state.

While other work has explored runtime VM replication (the third category), unlike CIVIC, the replicas do not perform new functionality and are typically employed for reliability and resource optimization such as high availability [23], fault tolerance [78], parallel worker forking [49; 56] and speeding up system testing [101]. Furthermore, they are based on the assumption of direct access to clones, which although valid when users themselves create replicas for self management, but problematic when access is restricted as in production environments. For the latter, it would either require enforcing guest cooperation for login access, or defeat portability by installing vendor-specific drivers and agents [85; 82; 84; 55], even if assuming this would be acceptable. CIVIC instead employs code injection to avoid imposing such guest cooperation and vendor locking. However, even without requring access to the clone VM, some types of analysis may still be performed, as done in JustRunIt [99]. It employes clone-based sandboxing to aid system administrators with tasks such as server consolidation and upgrade evaluation. For example, by experimenting with different resource allocations at the clone VM and observing its corresponding throughput / response time, it can estimate minimum resource needs for a service while satisfying its SLAs. CIVIC, on the other hand, can perform a wider range of analysis, by exercising control within the clone VM as well.

Finally, in case of record-and-replay based replication, new analysis in the middle of a recording is typically limited to the granularity of registers, instructions, memory addresses, page bits, and disk blocks. Although this en-

ables interesting analysis like heap overflow detection and memory safety checking [17], it is tricky to realize analysis at an OS or application-level semantics as with CIVIC. Also unlike CIVIC, these systems do not support changes to the application state [81]. Further architectural requirements and constraints that CIVIC does not have to deal with include maintaining multi-threaded scheduling order, precise instruction/branch counting, and stricter constraints on target architecture than virtualization, amongst others [23].

7. Conclusion

In this work, we've presented our execution-inspection decoupling approach to enable safe inspection of production VMs. All inspection impact and side-effects are restricted to a runtime replica (the clone) of the original unmodified VM (the source). New functionality is introduced on-the-fly using runtime code injection into the clone. By using clones as inspection proxies, and without enforcing access via pre-installed credentials or special agents / hooks inside the original VMs, CIVIC allows developers to gain operational visibility and control in the DevOps automation world. We demonstrated CIVIC's versatility and benefits with four use-cases. Our evaluation showed our approach is nimble and lightweight and has a low impact on the target systems.

References

[1] Amos Waterland . Stress. `http://people.seas.harvard.edu/ apw/stress/`.

[2] Anthony Liguori and Stefan Hajnoczi . QEMU Snapshots. `http://wiki.qemu.org/Documentation/CreateSnapshot` and `http://wiki.qemu.org/Features/Snapshots2`.

[3] Caleb Gilbert. Scaling Drupal: HTTP pipelining and benchmarking revisited. `http://rocketmodule.com/blog/scaling-drupal-http-pipelining-and-benchmarking-revisited/`.

[4] Daniel Stenberg. PHP cURL Manual. `http://no1.php.net/manual/en/intro.curl.php`.

[5] Jonathan Corbet and Andrea Arcangeli. Page faults in user space. `http://lwn.net/Articles/615086/`.

[6] Alexey Kopytov. SysBench Manual. `http://sysbench.sourceforge.net/docs/#database_mode`.

[7] Amazon. Summary of the October 22,2012 AWS Service Event in the US-East Region. `https://aws.amazon.com/message/680342/`.

[8] Andrea Arcangeli. Linux Userfault. `https://kernel.googlesource.com/pub/scm/linux/kernel/git/andrea/aa/+/userfault`.

[9] Angelo Laub. Practical Mac OS X Insecurity. `https://events.ccc.de/congress/2004/fahrplan/files/95-macosx-insecurity-paper.pdf`.

[10] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '98/PERFORMANCE '98, pages 151–160, New York, NY, USA, 1998. ACM.

[11] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy. Self-service cloud computing. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 253–264, New York, NY, USA, 2012. ACM.

[12] M. Carbone, M. Conover, B. Montague, and W. Lee. Secure and robust monitoring of virtual machines through guest-assisted introspection. In *Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses*, RAID'12, pages 22–41, 2012.

[13] Cassandra. Bug 5064: Alter table when it includes collections makes cqlsh hang. `https://issues.apache.org/jira/browse/CASSANDRA-5064`.

[14] J. Chen, S. Ghanbari, F. Iorio, A. B. Hashemi, and C. Amza. Ensemble: A tool for performance modeling of applications in cloud data centers. In *IEEE TRANSACTIONS ON CLOUD COMPUTING, SPECIAL ISSUE ON SCIENTIFIC CLOUD COMPUTING*, 2015.

[15] P. M. Chen and B. D. Noble. When virtual is better than real. In *HotOS*, pages 133–138, 2001.

[16] T.-c. Chiueh, M. Conover, and B. Montague. Surreptitious deployment and execution of kernel agents in windows guests. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 507–514, Washington, DC, USA, 2012. IEEE Computer Society.

[17] J. Chow, T. Garfinkel, and P. M. Chen. Decoupling dynamic program analysis from execution in virtual environments. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 1–14, 2008.

[18] I.-H. Chung and J. K. Hollingsworth. Automated cluster-based web service performance tuning. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, HPDC '04, pages 36–44, Washington, DC, USA, 2004. IEEE Computer Society.

[19] C. Colohan. The Scariest Outage Ever. CMU SDI/ISTC Seminar Series. `http://www.pdl.cmu.edu/SDI/2012/083012b.html`, 2012.

[20] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[21] J. Criswell, A. Lenharth, D. Dhurjati, and V. Adve. Secure virtual architecture: A safe execution environment for commodity operating systems. *SIGOPS Oper. Syst. Rev.*, 41(6):351–366, Oct. 2007.

[22] L. Cui, B. Li, Y. Zhang, and J. Li. Hotsnap: A hot distributed snapshot system for virtual machine cluster. In *Proceedings of the 27th International Conference on Large Installation System Administration*, LISA'13, pages 59–73, Berkeley, CA, USA, 2013. USENIX Association.

[23] 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*, pages 161–174. San Francisco, 2008.

[24] Dave Gilbert. PostCopyLiveMigration. `http://wiki.qemu.org/Features/PostCopyLiveMigration`.

[25] Dave Gilbert. PostCopyLiveMigration. `https://github.com/orbitfp7/qemu/tree/wp3-postcopy`.

[26] D. J. Dean, H. Nguyen, X. Gu, H. Zhang, J. Rhee, N. Arora, and G. Jiang. Perfscope: Practical online server performance bug inference in production cloud computing infrastructures. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 8:1–8:13, New York, NY, USA, 2014. ACM.

[27] Y. Diao, J. L. Hellerstein, S. Parekh, and J. P. Bigus. Managing web server performance with autotune agents. *IBM Systems Journal*, 42(1):136–149, 2003.

[28] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee. Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection. In *IEEE Security and Privacy '11*, pages 297–312.

[29] Y. Dong, W. Ye, Y. Jiang, I. Pratt, S. Ma, J. Li, and H. Guan. Colo: Coarse-grained lock-stepping virtual machines for non-stop service. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 3:1–3:16, New York, NY, USA, 2013. ACM.

[30] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. *SIGOPS Oper. Syst. Rev.*, 36(SI):211–224, Dec. 2002.

[31] EMC. VNX Snapshots White Paper. `https://www.emc.com/collateral/software/white-papers/h10858-vnx-snapshots-wp.pdf`.

[32] Florian octo Forster. Collectd: The system statistics collection daemon. `https://collectd.org/`.

[33] Y. Fu and Z. Lin. Space Traveling across VM: Automatically Bridging the Semantic Gap in Virtual Machine Introspection via Online Kernel Data Redirection. In *IEEE Security&Privacy'12*.

[34] Y. Fu and Z. Lin. Exterior: Using a dual-vm based external shell for guest-os introspection, configuration, and recovery. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '13, pages 97–110, 2013.

[35] Y. Fu, J. Zeng, and Z. Lin. Hypershell: A practical hypervisor layer guest os shell for automated in-vm management. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 85–96, 2014.

[36] A. Ganjali and D. Lie. Auditing cloud management using information flow tracking. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, STC '12, pages 79–84, New York, NY, USA, 2012. ACM.

[37] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *NDSS*, pages 191–206, 2003.

[38] S. Ghanbari, A. B. Hashemi, and C. Amza. Stage-aware anomaly detection through tracking log points. In *Proceedings of the 15th International Middleware Conference*, Middleware '14, 2014.

[39] G. R. Goodson, S. Susarla, and K. Srinivasan. System and method for fast restart of a guest operating system in a virtual machine environment, Aug. 23 2011. US Patent 8,006,079.

[40] Z. Gu, Z. Deng, D. Xu, and X. Jiang. Process implanting: A new active introspection framework for virtualization. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, pages 147–156. IEEE, 2011.

[41] M. R. Hines and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 51–60, New York, NY, USA, 2009. ACM.

[42] J. Hizver and T.-c. Chiueh. Real-time deep virtual machine introspection and its applications. In *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '14, pages 3–14, New York, NY, USA, 2014. ACM.

[43] J. Humble and D. Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[44] IBM. BigFix / Endpoint Manager. `https://github.com/bigfix/platform-releases`.

[45] A. Kangarlou, P. Eugster, and D. Xu. Vnsnap: Taking snapshots of virtual networked environments with minimal downtime. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 524–533. IEEE, 2009.

[46] S. T. King, G. W. Dunlap, and P. M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, 2005.

[47] Konstantin Boudnik. Hadoop: Code Injection, Distributed Fault Injection. `http://www.boudnik.org/~cos/docs/Hadoop-injection.pdf`.

[48] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze. Cloud federation. In *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD COMPUTING 2011.

[49] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *EuroSys*, 2009.

[50] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.

[51] M. Le and Y. Tamir. Fault injection in virtualized systems-challenges and applications. *Dependable and Secure Computing, IEEE Transactions on*, 12(3):284–297, May 2015.

[52] Linux man page. Chroot. http://linux.die.net/man/1/chroot.

[53] Linux man page. chrt - manipulate real-time attributes of a process. http://linux.die.net/man/1/chrt.

[54] Matthew H. Intel SGX for Dummies (Intel SGX Design Objectives). https://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx.

[55] Microsoft Azure. VM Agent and Extensions . https://azure.microsoft.com/en-us/blog/vm-agent-and-extensions-part-2/.

[56] M. J. Mior and E. de Lara. Flurrydb: A dynamically scalable relational database with virtual machine cloning. In *4th Annual International Systems and Storage Conference*, Haifa, Israel, May 2011.

[57] D. Mosberger and T. Jin. httperf - a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, 1998.

[58] Nemo. Abusing Mach on Mac OS X . http://uninformed.org/index.cgi?v=4&a=3.

[59] OpenBenchmarking/Phoronix. x264 Test Profile. http://openbenchmarking.org/test/pts/x264-1.7.0.

[60] OW2 Consortium. RUBiS: Rice University Bidding System. http://rubis.ow2.org/.

[61] Patrick Colp. VM Snapshots. http://www-archive.xenproject.org/files/xensummit_oracle09/VMSnapshots.pdf.

[62] PHP. Bug 45161 and 65458. https://bugs.php.net/bug.php?id=45161 and https://bugs.php.net/bug.php?id=65458.

[63] B. Procházka, T. Vojnar, and M. Drahanskỳ. Hijacking the linux kernel. In *MEMICS*, pages 85–92, 2010.

[64] QEMU. Documentation/Debugging: Using gdb. http://wiki.qemu.org/Documentation/Debugging.

[65] Russell Coker. Bonnie++. http://www.coker.com.au/bonnie++/.

[66] A. Saberi, Y. Fu, and Z. Lin. Hybrid-bridge: Efficiently bridging the semantic-gap in vmi via decoupled execution and training memoization. In *NDSS*, 2014.

[67] A. Saboori, G. Jiang, and H. Chen. Autotuning configurations in distributed systems for performance improvements using evolutionary strategies. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, ICDCS '08, pages 769–776, Washington, DC, USA, 2008. IEEE Computer Society.

[68] T.-I. Salomie, G. Alonso, T. Roscoe, and K. Elphinstone. Application level ballooning for efficient server consolidation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 337–350, 2013.

[69] B. Satzger, W. Hummer, C. Inzinger, P. Leitner, and S. Dustdar. Winds of change: From vendor lock-in to the meta cloud. *IEEE Internet Computing*, 17(1):69–73, Jan. 2013.

[70] B. Shi, B. Li, L. Cui, J. Zhao, and J. Li. Syncsnap: Synchronized live memory snapshots of virtual machine networks. In *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, pages 490–497, Aug 2014.

[71] L. M. Silva, J. Alonso, P. Silva, J. Torres, and A. Andrzejak. Using virtualization to improve software rejuvenation. In *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*, pages 33–44. IEEE, 2007.

[72] D. Srinivasan and X. Jiang. Time-traveling forensic analysis of vm-based high-interaction honeypots. In *Security and Privacy in Communication Networks*, pages 209–226. 2012.

[73] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu. Process outgrafting: An efficient "out-of-vm" approach for fine-grained process execution monitoring. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 363–374, New York, NY, USA, 2011. ACM.

[74] Stanley Cen. Mac OS X Code Injection and Reverse Engineering . http://stanleycen.com/blog/mac-osx-code-injection/.

[75] R. Sun, J. Yang, Z. Gao, and Z. He. Lsovc: A framework for taking live snapshot of virtual cluster in the cloud. In *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 1727–1732, Nov 2013.

[76] S. Suneja, C. Isci, V. Bala, E. de Lara, and T. Mummert. Non-intrusive, out-of-band and out-of-the-box systems monitoring in the cloud. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '14, pages 249–261, New York, NY, USA, 2014. ACM.

[77] S. Suneja, C. Isci, E. de Lara, and V. Bala. Exploring vm introspection: Techniques and trade-offs. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '15, pages 133–146, New York, NY, USA, 2015. ACM.

[78] Y. Tamura. Kemari: Fault tolerant vm synchronization based on kvm. 2010.

[79] Tim Starling. Measuring memory usage with strace. http://tstarling.com/blog/2010/06/measuring-memory-usage-with-strace/.

[80] Vasilis Liaskovitis, Igor Mammedov, et. al. ACPI memory hotplug. https://lists.gnu.org/archive/html/qemu-devel/2014-04/msg00734.html.

[81] N. Viennot, S. Nair, and J. Nieh. Transparent mutable replay for multicore debugging and patch validation. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 127–138, New York, NY, USA, 2013. ACM.

[82] VMware. Guest Operating System Customization Requirements . https://pubs.vmware.com/vsphere-51/index.jsp#com.vmware.vsphere.vm_admin.doc/

GUID-E63B6FAA-8D35-428D-B40C-744769845906.html.

[83] VMware. Understanding Clones. https://www.vmware.com/support/ws5/doc/ws_clone_overview.html.

[84] VMware. VMCI Sockets Documentation. www.vmware.com/support/developer/vmci-sdk/.

[85] VMware. VMWare Tools. http://kb.vmware.com/kb/340.

[86] S. Vogl, F. Kilic, C. Schneider, and C. Eckert. X-tier: Kernel module injection. In J. Lopez, X. Huang, and R. Sandhu, editors, *Network and System Security*, volume 7873 of *Lecture Notes in Computer Science*, pages 192–205. Springer Berlin Heidelberg, 2013.

[87] E. Warszawski and M. Ben-Yehuda. Fast initiation of workloads using memory-resident post-boot snapshots, Nov. 3 2015. US Patent App. 14/930,674.

[88] J. Wettinger, U. Breitenbücher, and F. Leymann. Standards-based devops automation and integration using tosca. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, UCC '14, pages 59–68, Washington, DC, USA, 2014. IEEE Computer Society.

[89] R. Wu, P. Chen, P. Liu, and B. Mao. System call redirection: A practical approach to meeting real-world virtual machine introspection needs. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 574–585, 2014.

[90] X. Wu, Z. Shen, R. Wu, and Y. Lin. Jump-start cloud: efficient deployment framework for large-scale cloud applications. *Concurrency and Computation: Practice and Experience*, 24(17):2120–2137, 2012.

[91] Xen Project Blog. Debugging on xen. https://blog.xenproject.org/2009/10/21/debugging-on-xen/.

[92] Xen Project Wiki. Blktap. http://wiki.xenproject.org/wiki/Blktap.

[93] Xen Project Wiki. Migration. http://wiki.xenproject.org/wiki/Migration.

[94] Xen.org: Sean Dague, Daniel Stekloff, Reiner Sailer, and Stefan Berger. Xen Management User Interface. http://xenbits.xen.org/docs/4.3-testing/man/xm.1.html#block_devices.

[95] Yasuaki Ishimatsu. Memory Hotplug. http://events.linuxfoundation.org/sites/events/files/lcjp13_ishimatsu.pdf.

[96] J. Zeng, Y. Fu, and Z. Lin. Pemu: A pin highly compatible out-of-vm dynamic binary instrumentation framework. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '15, pages 147–160, New York, NY, USA, 2015. ACM.

[97] ZeroTurnaround. JRebel Java Plugin. http://zeroturnaround.com/software/jrebel/.

[98] F. Zhang, J. Cao, L. Liu, and C. Wu. Fast autotuning configurations of parameters in distributed computing systems using ordinal optimization. In *Proceedings of the 2009 International Conference on Parallel Processing Workshops*, ICPPW '09, pages 190–197, Washington, DC, USA, 2009. IEEE Computer Society.

[99] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner. Justrunit: Experiment-based management of virtualized data centers. In *Proc. USENIX Annual technical conference*, pages 18–18, 2009.

[100] W. Zheng, R. Bianchini, and T. D. Nguyen. Automatic configuration of internet services. *SIGOPS Oper. Syst. Rev.*, 41(3):219–229, Mar. 2007.

[101] J. Zhi, S. Suneja, and E. De Lara. The case for system testing with swift hierarchical vm fork. In *Proceedings of the 6th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'14, pages 19–19, 2014.

[102] J. Zhu, Z. Jiang, and Z. Xiao. Twinkle: A fast resource provisioning mechanism for internet services. In *INFOCOM, 2011 Proceedings IEEE*, pages 802–810, 2011.