LightVM

- **Goal**: make VMs light and as fast as containers, while providing stronger isolation
    - Fast instantiation
    - High Instance Density
    - Quick pause/unpause
    - Small memory and on-disk footprint
- Based on Xen
- Lightweight tool stack
- Modifies Xen's control plane
- LightVM can boot a VM in 2.3ms, comparable to fork/exec on Linux (1ms), and two orders of magnitude faster than Docker.

What is a Unikernel?

- A customized operating system for a particular application or set of applications. The operating system contains only the code required to run the specific application. Hence the name "uni-kernel": a kernel for a single purpose. Unikernels are lighter-weight than general-purpose kernels, but lose the ability to run other applications.

Using Unikernels

- General purpose OS makes virtual machines heavyweight
- First observation of authors: "Most containers and virtual machines run a single application"
- **Idea**: only include in VM what is necessary for that application (use a unikernel)
- How to achieve this idea: **Tinyx**, a tool that generates a unikernel for a given application
    - Prior work: Mirage and ClickOS
- Lower bound for how small a unikernel can be: an OS for an application which just returns the time
    - "The resulting VM image, which we will refer to as the daytime unikernel, is only 480KB (uncompressed), and can run in as little as 3.6MB of RAM"
    - Running an entire OS with so little memory is impressive!
    - To give you some context: just your Chrome browser takes 100+ MB of memory

Tinyx

- Tinyx is an automated build system that creates minimalistic Linux VM images targeted at running a single application (although the system supports having multiple ones)
- The Tinyx build system takes two inputs: an application to build the image for (e.g., nginx) and the platform the image will be running on (e.g., a Xen VM)
- To derive dependencies, Tinyx uses (1) objdump to generate a list of libraries and (2) the Debian package manager
    - For example, running objdump -x /bin/bash shows:
        - "...
        - Dynamic Section:
        - NEEDED libtinfo.so.5
        - NEEDED libdl.so.2
        - NEEDED libc.so.6
        - .."
- Tinyx first mounts an empty OverlayFS directory over a Debian minimal debootstrap system. In this mounted directory we install the minimal set of packages discovered earlier as would be normally done in Debian. Since this is done on an overlay mounted system, unmounting this overlay gives us all the les which are properly con gured as they would be on a Debian system

- Before unmounting, we remove all cache les, any dpkg/apt related les, and other unnecessary directories.
- Tinyx finally merges this directory with the VM image
- The kernel is built using “tinyconfig” configuration
    - If this fails, more modules are added heuristically (drop module, add if test fails)

Modifying Xen

- Dom0 hosts the XenStore, a proc-like central registry that keeps track of management information such as which VMs are running and information about their devices, along with the libxs library containing code to interact with it.
- The XenStore provides watches that can be associated with particular directories of the store and that will trigger callbacks whenever those directories are modified.
- For communication between Dom0 and the other guests, Xen implements a split-driver model: a virtual back-end driver running in Dom0 (e.g., the netback driver for networking) communicates over shared memory with a front-end driver running in the guests (the netfront driver)
- Instantiation time is the dominant cost for small guests

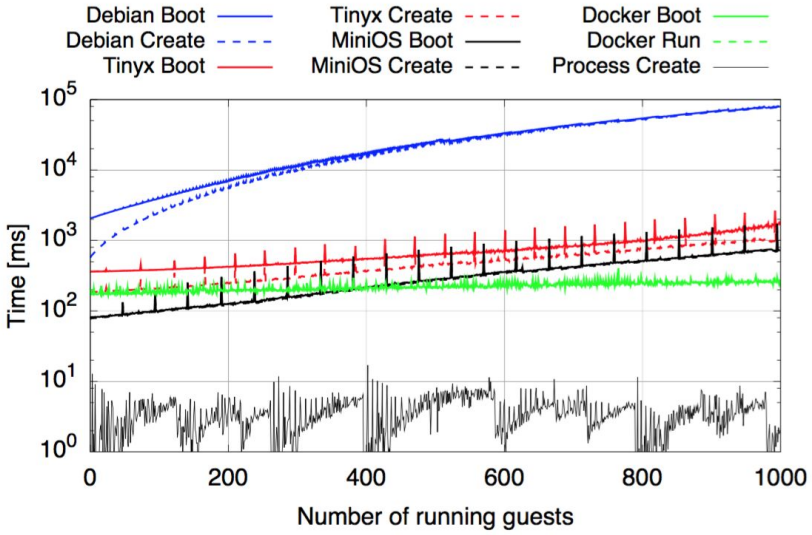

Figure 4: Comparison of domain instantiation and boot times for several guest types. With small guests, instantiation accounts for most of the delay when bringing up a new VM.

-
- Docker containers start in around 200ms, and a process is created and launched (using fork/exec) in 3.5ms on average (9ms at the 90% percentile)

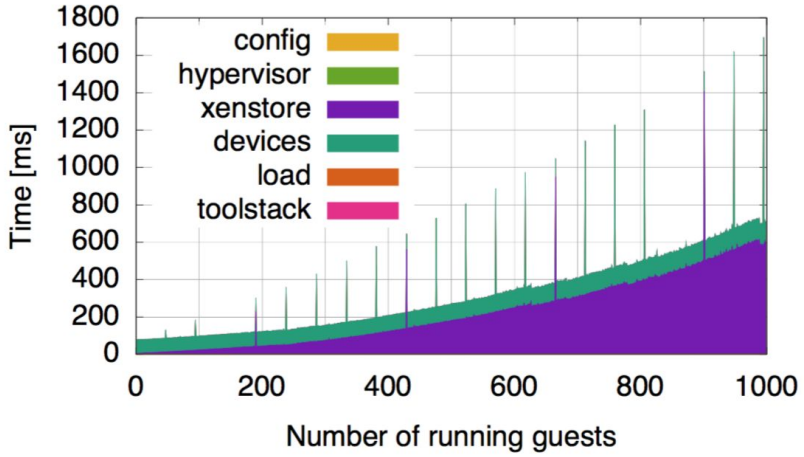

Figure 5: Breakdown of the VM creation overheads shows that the main contributors are interactions with the XenStore and the creation of virtual devices.

-
- The protocol used by the XenStore is quite expensive, where each operation requires sending a message and receiving an acknowledgment, each triggering a software interrupt: a single read or write thus triggers at least two, and most often four, software inter- rupts and multiple domain changes between the guest, hy- pervisor and Dom0 kernel and userspace; as we increase the number of VMs, so does the load on this protocol.
- Secondly, writing certain types of information, such as unique guest names, incurs overhead linear with the number of machines because the XenStore compares the new entry against the names of all other already-running guests before accepting the new guest's name.
- Finally, some information, such as device initialization, requires writing data in multiple Xen- Store records where atomicity is ensured via transactions.
- LightVM: a complete re-design of the basic Xen control plane optimized to provide light- weight virtualization
- LightVM does not use the XenStore for VM cre- ation or boot anymore, using instead a lean driver called noxs that addresses the scalability problems of the XenStore by enabling direct communication between the frontend and backend drivers via shared memory instead of relaying messages through the XenStore
- **Basic idea**: remove XenStore, use shared memory for communication instead of message passing
- LightVM provides a split toolstack that separates VM cre- ation functionality into a prepare and an execute phase, reducing the amount of work to be done on VM creation
- We have also implemented chaos/libchaos, a new virtualization toolstack that is much leaner than the standard xl/libxl, in addition to a small daemon called xendevd that quickly adds virtual interfaces to the software switch or handles block device images' setup
- The **insight** here is that the hypervisor already acts as a sort of centralized store, so we can extend its functionality to implement our noxs (no XenStore) mechanism.
- In addition, we modify Xen's hypervisor to create a new, special device memory page for each new VM that we use to keep track of a VM's information about any devices, such as block and networking, that it may have.
- We also have a hypercall to write to and read from this memory page, and make sure that, for security reasons, the page is shared read-only with guests, with only Dom0 allowed to request modifications.
- Steps in creating a VM:
    - Ioctl to NOXS to create device
    - Toolstack does hypercall to add device details to device page

    - When VM boots, maps in device page using hypercall, uses information in page to communicate with back end
    - Front and back-end of the networking device exchange information through shared memory on the device page
- Migration handled through a special sysctl device

The Xen ToolStack

- **Main idea:** pre-creation of VMs
- **Insight:** the same stuff happens for the creation of a large number of VMs
- VM creation in LightVM has a **prepare** phase (common functionality) which is pre-executed
- Execute phase customizes it for the actual VM creation
- Xendevd is a daemon that initialized user-space devices directly (faster than using bash scripts)

Evaluation

- LightVM able to boot 8000 virtual machines in 4-10 ms
- LightVM consumes ~16GB for 1000 Virtual machines: approx 16 MB per virtual machine
- LightVM memory usage on par with Docker containers
- Both lightVM and docker have low CPU utilization (slightly higher than Docker)

Use Cases

- Personal Firewalls
- Just-in-time Services
- TLS Service
- Lightweight Computer Service

Reading

http://cnp.neclab.eu/projects/lightvm/lightvm.pdf
Open source repositories to contribute to: http://cnp.neclab.eu/projects/lightvm/download/