Operating Systems à la Carte

Applying Composible Image Configurations for Scalable OS Research

Valentijn Dymphnus van de Beek

Delft University of Technology
Electrical Engineering, Mathematics, and Computer Science
The Netherlands

September 22, 2022

Outline

Operating Systems

- Computers are complicated
  - Heterogenous system
  - Different suppliers
  - Incompatibilities
- Operating Systems
  - Abstract over devices
  - Consistent API
  - Security guarantees
- Therefore, OSes are complicated

Irreplaceability of the Operating Systems

- Changes in OS modify those, but is that even possible?

So whose turn is it again?

Assume that there is a system that runs a kernel that makes use of an arbitrary job scheduling algorithm I and that the system allows for the in-place replacement of kernel components. The user replaces the job scheduling algorithm with J and to do so it starts process P to do so. Halfway through the replacement, the kernel interrupts P to run another job. How does it then decide what the next job is? Does it use I or J?

Irreplaceability of the Operating Systems

Never take a stuffed canary to the mine

Assume that there is such a system that makes use of a stack canary to secure against stack overflow attacks. Similarly, assume that this system also allows for the in-place updating of security mechanisms. A user uploads a new version of the mechanism that always returns true and runs a program that triggers the mechanisms. Since the system has a stack canary so the program crashes and the stack canary always returns true so the program keeps running. We can therefore conclude that one of the assumptions cannot hold. Since it is given that the system has a stack canary, it must be the case that the system cannot be updated in place.

Irreplaceability of the Operating Systems

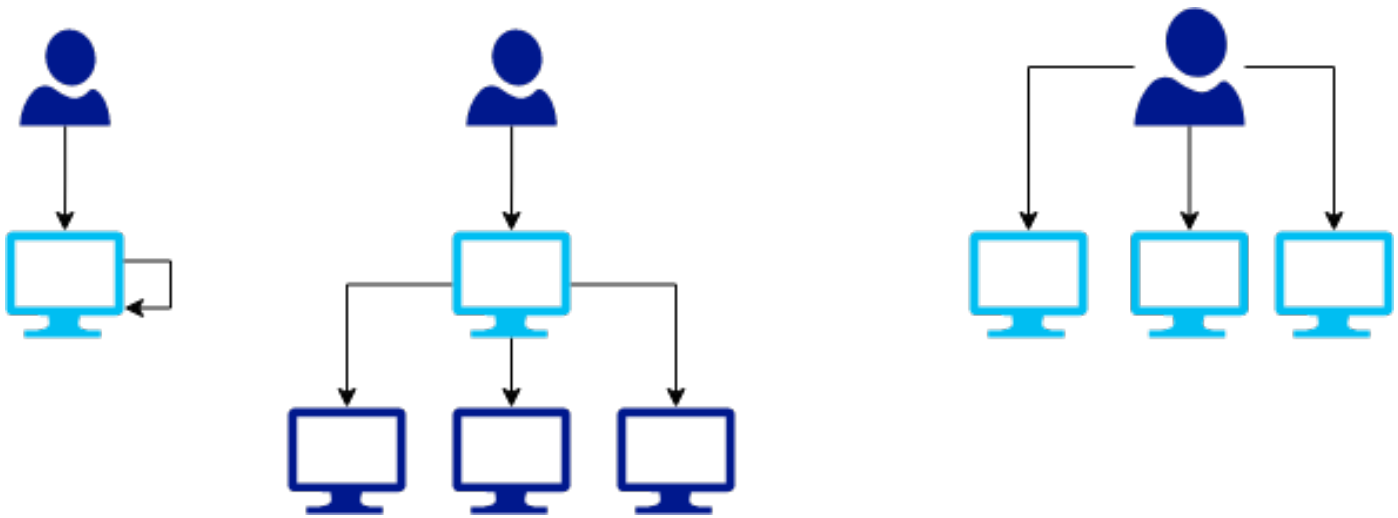
API (in)consistencies

Assume that there is a system S with a consistent API that abstracts over devices. Alice and Bob use instances S_a and S_b respectively of system S. Since the API is consistent any program that runs one system also should run on the other. Alice changes the system API such that it adds a new feature and writes a program using it. This program should run successfully on S_b since the API is consistent, but it should fail since the API is also changed.

Irreplaceability of the Operating Systems

- OS replacement can thus not be done in place
- So by necessity the system must be reset.

OS installation strategies



Reusing machines risk data loss

- Simplest and cheapest system
- High risk of information loss
- High amount of labour-intensive manual steps
- Low throughput

Virtualization causes contamination

- Virtual machines
  - Virtualized hardware
  - All possible systems can be run
  - Experimental results may reflect the virtual hardware or the VM execution code

Virtualization causes contimination

- Containers
  - Makes use of the host kernel
  - Only homoousios systems, so systems of the same type

Only of the same type

On a Linux system, you can run Linux containers, but not FreeBSD containers and on a FreeBSD system you cannot run a Linux container. This is because in a container the same kernel is used.

- Results may depend on the characteristic of the host
- A change in the kernel cannot be tested

Resource clusters are hard to manage

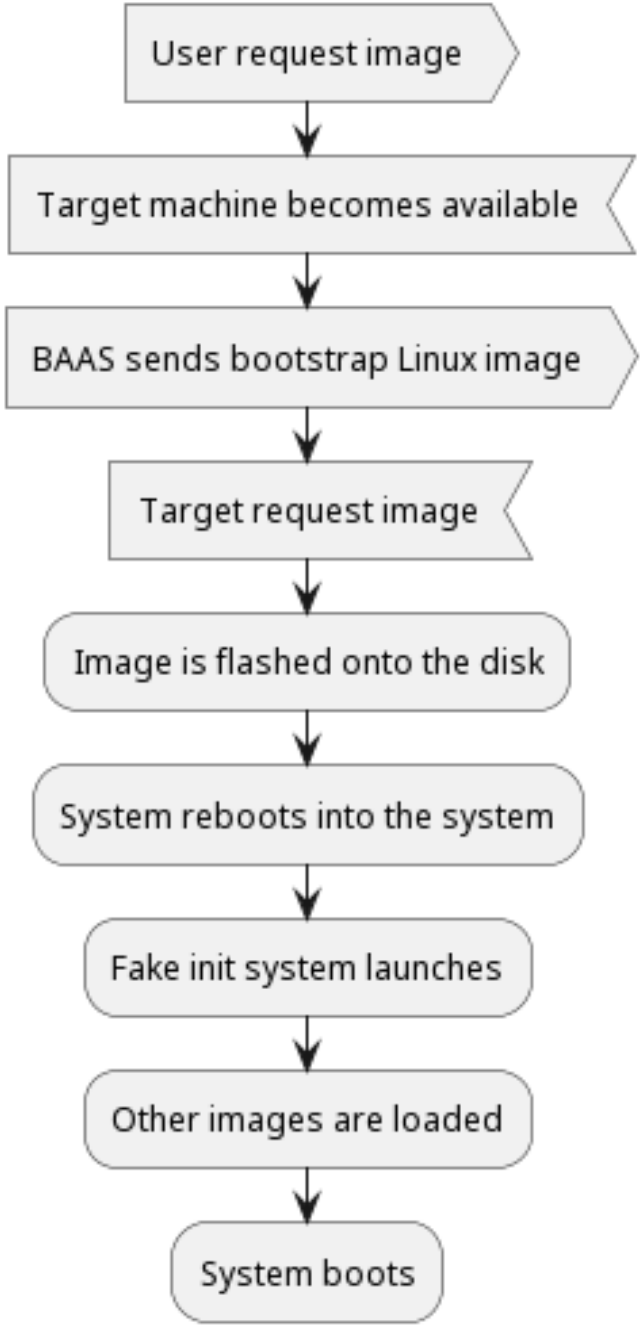
- Parallel instances on separate hardware
- Slight risk of information loss
- Difficulty ensuring hardware consistency
- Could be shared, but needs to be managed and scheduled.

Baremetal as a Service

- Bachelor student side project TU Delft students 2020-2021
- Based on the additional hardware scheme
- Server distributed images over the cluster
- See the GitHub page for the user manual with more information and diagrams

Control flow of BAAS

- Server that stores and schedules images
- A bootstrap OS that flashes images to disk
- Programmable REST API to manage server
- Web frontend


User request image
Target machine becomes available
BAAS sends bootstrap Linux image
Target request image
Image is flashed onto the disk
System reboots into the system
Fake init system launches
Other images are loaded
System boots

Benefits over prior schemes

- Direct access to hardware
- Eliminates dangerous operations
- Highly scalable
- Lower management overhead

Management Server

- Written in Go and SQL
- Stores images and versions
- Handles boot requests
- User management and access control
- Only exposes a REST API

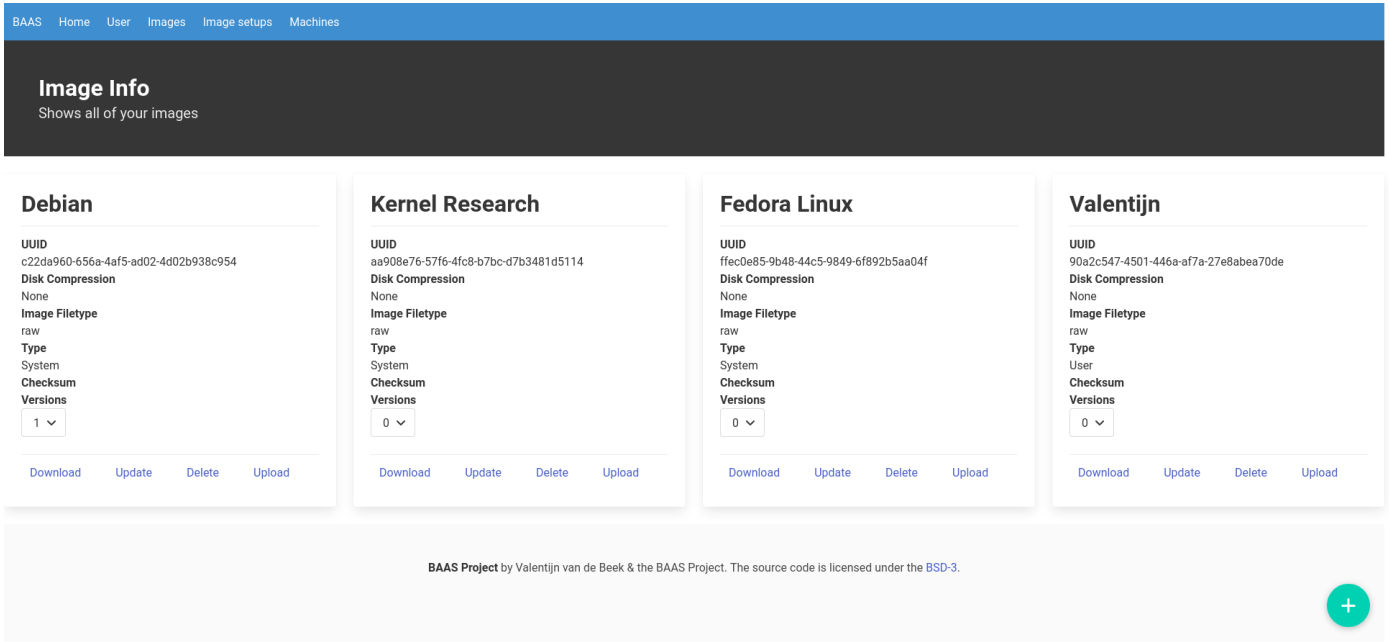
Deployment image

- Debian stable image generated from docker
- Go binary + UNIX tools + python scripts
- Downloads and deploys images from the server
- Uploads changes as a new version if requested
- Boots into the target image
- Makes use of an LRU cache to minimize disk writes

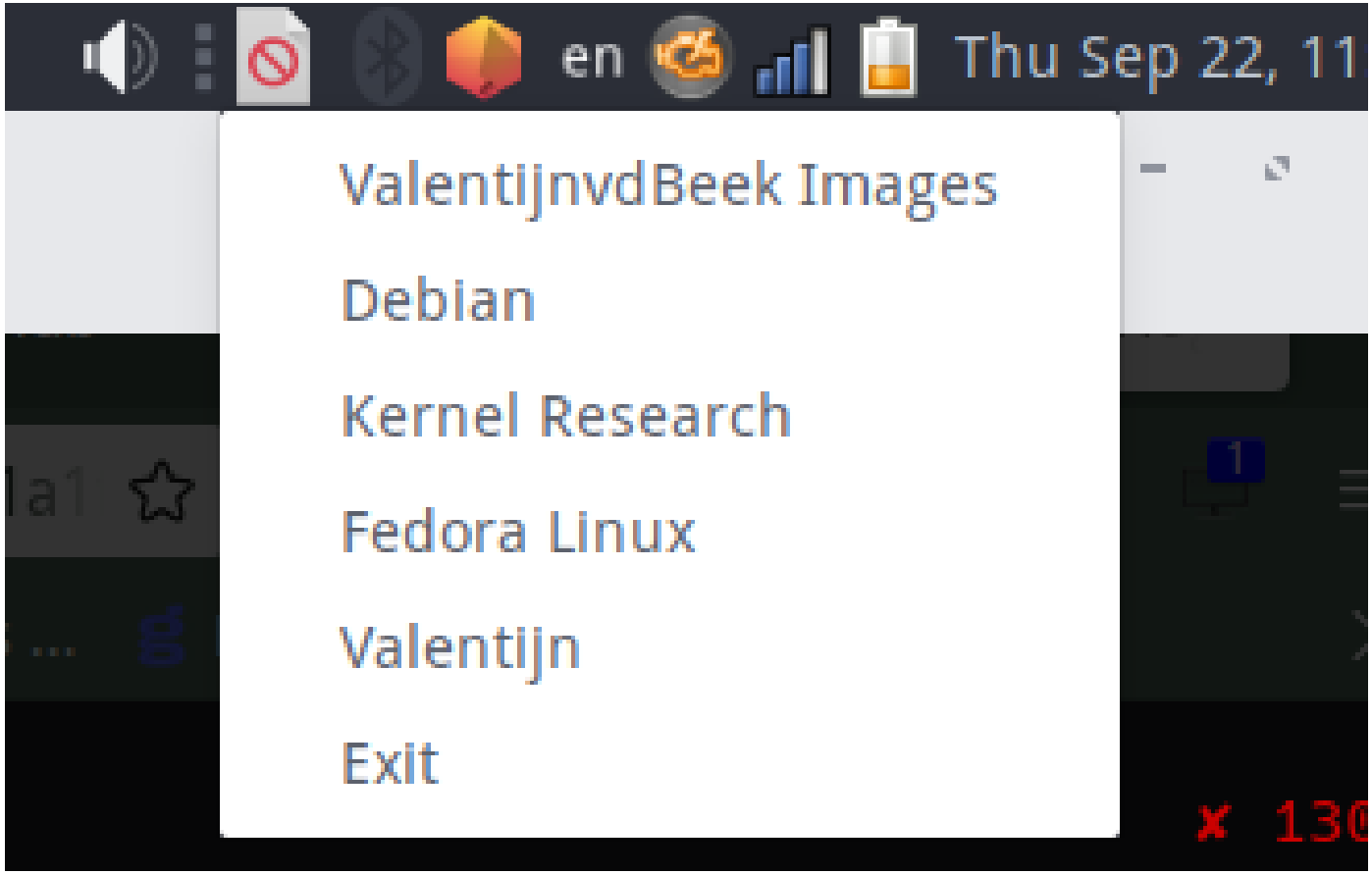
Web Frontend

- Written in HTML/CSS/JS with Bulma
- Minimal dependencies on libraries
- No privileged access to the system
- Only one of many possible versions, also possible Python & Emacs Lisp

Image web frontend



Python Applet for the titelbar



Shim init system

- Statically compiled C program.
- Replaces the init system on the target.
- Forks to download additional images in the background.
- Boots into the original init system.
- A working prototype for GNU/Linux + Systemd

Drawbacks of a central system

- High duplication of data

Huge Page Tables leading to Huge Data Duplication

Take a researcher who has written five algorithms to improve the performance of the Huge Page Table allocation on the Linux system and is particularly interested in the effect of her algorithms on a graphical system. For her to test her system she must make five images that contain the same version of her distro, an installation of X11, and a window manager. The the only thing that changes is the Linux kernel which is only 12M while the whole image is maybe 10GB, meaning 40GB is wasted on duplicate information.

Drawbacks of a central system

- Confused image size (disk or OS?)

What are we wasting today?

The OS image is 32GB and the target machine has 512GB. If you choose the former, you store 16x of space actually on the server. While with the latter, you waste 16x of the space on the target machine. With the OS image, you get the additional complexity of what ought to happen if the image grows in size during execution. Compressing the data may not be an option either, since data is not stored linearly and the sizes of images may differ. Both of these factors add a degree of randomness to the "empty" space of each boot that is hard to compress and causes ever-increasing image sizes.

Drawbacks of a central system

- Data retrieval.
  - Single system → Data is on the system
  - Virtualization → Write to host system
  - Parallel system → Information is on the target
  - BaaS → Information is on some system somewhere

Drawbacks of a central system

- Persistent machine storage.

How big is a 1TB even?

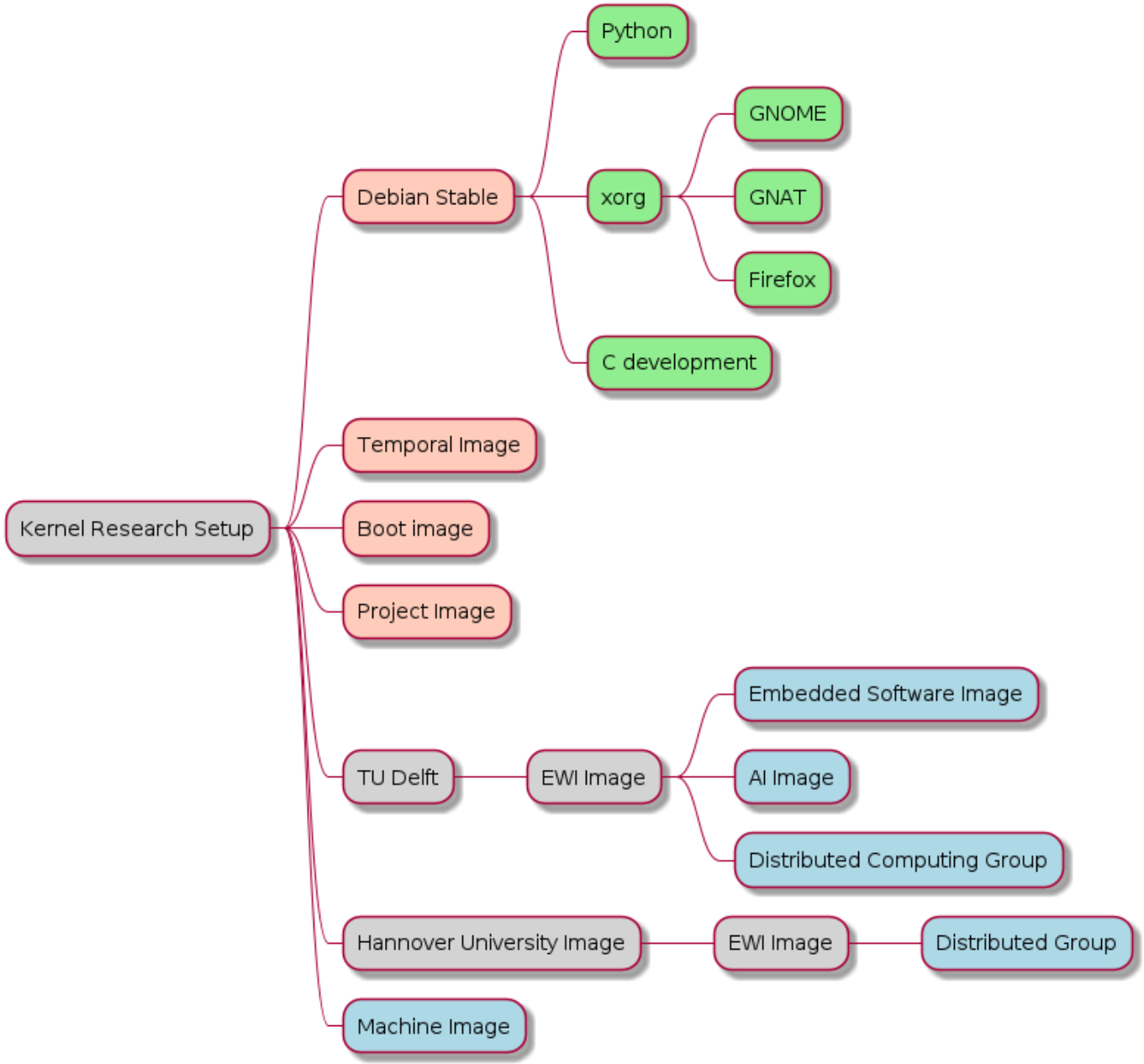
A researcher changes five parameters of the kernel and executes a benchmark measure of performance. She does ten iterations of the benchmark to ensure that the results are accurate. The image used has a total size of 20GB. Since the disk is overwritten for each flash to ensure a blank slate on the target image and netboot images have no persistent storage, caching is not possible. This causes a total of 5*10*20 = 1TB of reading and writing operations to take place.

Composing an image

- Images are just files that happen to contain other files
- Images encode a particular intent
- Semantic meanings can be encoded into concrete application-level guarantees.

Grundgesetz Principle

Technically the Grundgesetz is not a constitution, but rather a basic law meant to bootstrap the state institutions of the Bundesrepublik until the eventual reunification of Germany. In the intervening, it has taken the role of an official constitution within the German political system and has stayed in place even after the reunification in 1992. It serves as a good example of where the meaning of something is given by the meaning that humans have given to it, rather than the precise or initial meaning given to it.


Kernel Research Setup
Debian Stable
Python
xorg
GNOME
GNAT
Firefox
C development
Temporal Image
Boot image
Project Image
TU Delft
EWI Image
Embedded Software Image
AI Image
Distributed Computing Group
Hannover University Image
EWI Image
Distributed Group
Machine Image

Contractual Guarantees

- Contracts: possibly binding agreement between two parties
- Each type infers a contract onto the images
- External clauses are enforced by the server
  - Versioned (System) or Checksummed (User)
  - Life expectancy (Temporary)
  - Automatic Inclusion (Machine)
  - Sharing between users (Project)
- Internal Clauses are services optionally provided by the image
  - Lazy loading
  - Autostartup
  - Network access

BAAS Design principles

- Provide what you can, reject what you must
- It is better to do 100% for 90% than 50% for 100%
- Trust researchers to make the right decision for them.

Machine Image

- Automatically included by the system
- Allows for:
  - Persistent storage
  - Image caching
  - Online repair of remote systems
- Always updated

System Image

- User selected
- Base operating system
- Only updated on request
- Optionally loads and executes contracts

Image Generation

- How does a researcher make an image?
- Virtualization!
  - Use a docker to generate a container
  - Extract the container to an image
  - Add a kernel and bootloader to make it bootable
- Allows for:
  - Nested images
  - Automatic update
  - Group-specific files
- OverlayFS
  - Filesystem that combines multiple mounts into one
  - Only supported on Linux and possibly FreeBSD/NetBSD
  - Great use case for the contract mechanism where this is an option for those operating systems in particular.

Project Image (Not implemented)

- Always included in a project
- May be shared
- Includes experimental results
- May include a binary that executes the experiment

Future Work

- Implementing the derivative containers
- Exploring different virtualization options for image generations
- Shim init system on {Free,Net,Open}BSD
- Cleanup init system code base
- Finish Python & Emacs Lisp interfaces
- Run experiments on real hardware
- Implementations of more image types
- Online editing of files on images

How you can help

- Make use of the system
- Code contributions
- Letting me use your hardware for experimentation

Conclusions

- Operating system research is hard
- Wiping disks is dangerous and alternatives are hard to manage or expensive
- BaaS offers an attractive alternative
- Drawbacks to that model can be mitigated with composability
- By using semantic types features can be offered at will
- Provide what you can, reject what you must

Comparison Table

	Comparison Features				
Name/Feature	Reuse	VM	Container	Multiple machines	BaaS
Price	Free	Free	Free	High	Medium
Throughput	Low	High	High	High	High
Info loss risk	High	None	None	Low	Low
Scalablity	None	High	High	Low	High
Time Required	High	Low	Low	Very High	Low
Data reuse	High	Low	High	None	High
Lazy Loading	Yes	Yes	Yes	No	Yes
Management overhead	None	Medium	Medium	Very high	Low
OS support	All	All	Same kernel	All	All
Project sharing	No	Yes	Yes	No	Yes
Syncronize instances	No	No	No	No	Yes
Automatic update	No	No	Yes	No	Yes
User control	No	Yes	Yes	No	Yes

General Information

- Can the images be generated using virtualization techniques?

Yes, see the additional slide. I have also updated it with information about OverlayFS

- How does the project relate to Nix/Guix?

Nix and Guix have similar goals to this system concerning the composability of operating systems. However, these systems are packages manager that is added on top of an operating system, while this system focuses on the composability of the operating system itself. In other words: Guix and Nix aim to be general-use systems that complement your OS with agnostic packages, while BAAS is a server solution where packages are tied intimately with the OS that generated it.

Implementation Details

- Can users create personal derivative versions of system images?

At the moment no, system images are system-wide and shared with all users. This is a very interesting idea though and will be added at some point.

- What technologies are used to create the system?

Slides have been updated with the relevant information

Implementation Details

- How can reproducibility be ensured using such a system?

This is a very good question and something that I have not thought about before. Reproducibility within the group and university would be vastly improved since they can use the same setups that were used by the original research. The same ought to hold for people outside of academia since the system diagram gives a comprehensive and understandable view of what was used in the research. It is however not fully excluded that there may be some side-effects present due to the particulars of that specific image or the interaction between the composed images. It is therefore clear to me that being able to download an on-server composed image should be added so that these images can be made available to the community.

Performance questions

- What are some of the performance characteristics?

At the moment no formal experiments have been done on the system. However, this is not due lack of interest. The project was started as a development project with a focus on creating a working system rather than the novelty or research aspects of the project, so benchmarking was not in the initial goals of the project.
More importantly, the professor who started the project is no longer full-time at Delft which meant access to the hardware needed to test it has become increasingly hard. He did offer access to the clusters at his new group, but these are still in the process of being built.

Performance questions

However, roughly speaking it takes around 250ms to download, mount and use the machine image. 3 minutes and 14 seconds to download/flash an Arch Linux boot ISO and 250ms to unmount and upload the machine image. This yields around 4 minutes total from request to the start of the user-requested image on my development machine. Take these numbers with a grain of salt, they are generated using my 2013 water-damaged Lenovo IdeaPad with a VM as a target machine. So the results of a real-world system may differ

Composition here takes a minimal amount of time since setups are just labeled to images on disk and the total amount of space needed for such a setup is no different than putting together into one image.

Performance questions

Or more clearly, data is sent over in the different images and put together at the machine rather than being added together at the server. Doing it in this order allows for the lazy loading of images and avoids writing to a disk more than once.

- How many servers have been run using the system at once

Zero, as said above I do not have access to hardware to test with. If you do have access to such a system that I can loan for an afternoon, please let me know!

Performance questions

- I have a really interesting (idea for) a master thesis project and I would love to have you onboard. Can I contact you about that?
  Wow, thank you for the interest in someone who *definitely* is not me I promise. You can reach me at my university email at *v.d.vandebeek@student.tudelft.nl*. I am always happy to talk with you about it.

Author Information

University

@baas-project/baas

v.d.vandebeek@student.tudelft.nl

Personal

@ValentijnvdBeek

valentijn@posteo.net

Very special thanks to Jonathan Dönszelmann, Victor Roest, and Paulina Sobociska.

Operating Systems is a (relatively) new field for me, so feel free to send me any interesting papers, ideas or info!

Sources

N. Brown, Overlay filesystem, Overlay Filesystem - The Linux Kernel documentation, 14-Aug-2022. [Online]. Available: https://www.kernel.org/doc/html/latest/filesystems/overlayfs.html. [Accessed: 22-Sep-2022].

I. Velichko, From Docker container to bootable Linux disk image, Ivan on Containers, Kubernetes, and Server-Side, 16-Jul-2022. [Online]. Available: https://iximiuz.com/en/posts/from-docker-container-to-bootable-linux-disk-image/ [Accessed: 22-Sep-2022].

P. H. O'Neil, K. J. Fields, and D. Share, Cases in comparative politics. New York: W. W. Norton & Company, 2021.

T. Anderson and M. Dahlin, Operating systems: Principles and practice. United States: Recursive Books, 2014.