Unikernels: Library Operating Systems for the Cloud

Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, Jon Crowcroft

Jay Sh
COMP 517
2020.10.12

Unikernels: The Rise of the Virtual Library Operating System

Anil Madhavapeddy, David J. Scott

Jay Sh
COMP 517
2020.10.12

Problem & Motivation

- VMs are used extensively in the cloud to achieve economy of scale
  - Xen, KVM
- VMs are often single purpose
  - Commodity OSs are not
- The hosting software stack has too many layers
  - Physical protocols, POSIX, processes and threads, language runtimes, application code
- VM images often need to be transferred for deployment
  - Commodity OSs are not optimized for size or boot time

Problem & Motivation

- libOS is good for applications that need performance but has two problems
  - Resource isolation is hard when running multiple applications
  - Device drivers need to be rewritten and catch up with commodity hardware
- “The fast-moving world of commodity PC hardware meant that, no matter how many graduate students were tasked to write drivers, any research libOS prototype was doomed to become obsolete in a few short years.”

Container-Based OSs?

- Relaxed isolation
- Cannot optimize system components to suit application needs
  - Networking stack

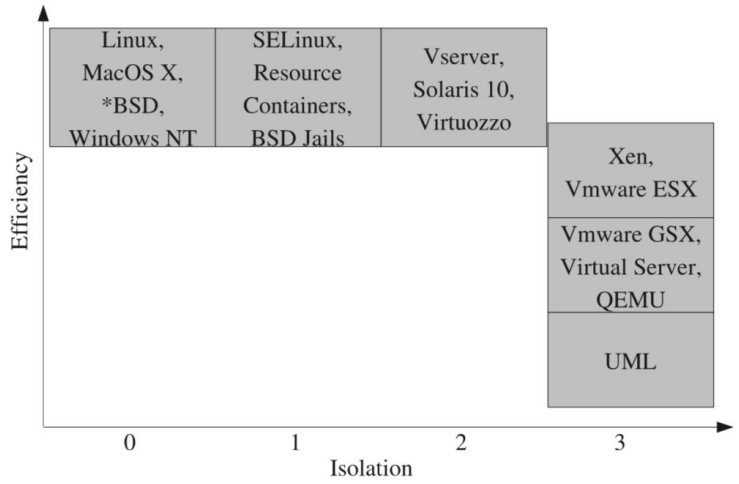


Figure 2: Summary of existing hypervisor- and COS-based technology

Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors

Unikernel

- Virtual machine running single language libOS and application
- OS components are implemented as libraries and applications link against them
- **Single process, single address space**
- Designed for the cloud

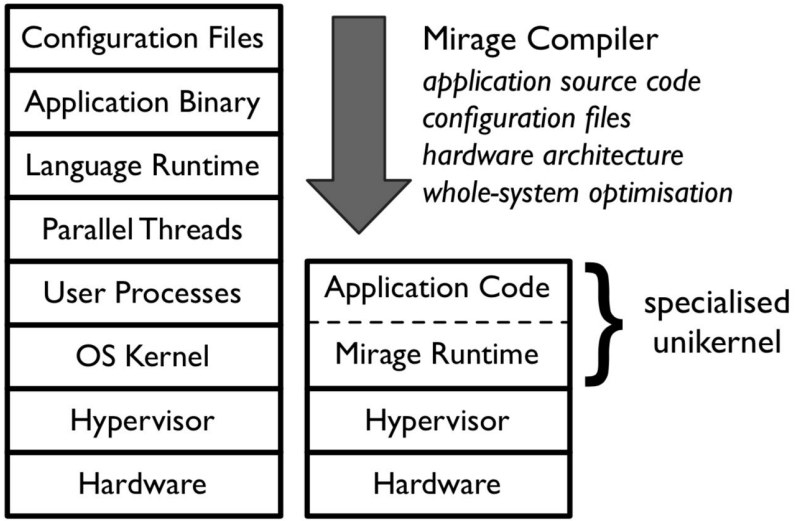


Figure 1: Contrasting software layers in existing VM appliances *vs.* unikernel's standalone kernel compilation approach.

MirageOS

- Functional prototype that runs on Xen
- libOS and application written in OCaml
- Uses PVBoot library to boot unikernels
  - One vCPU
  - One 64-bit address space
- Modified OCaml runtime
  - Occupies text and data region
  - GC uses two heaps
- Uses Lwt library for threading

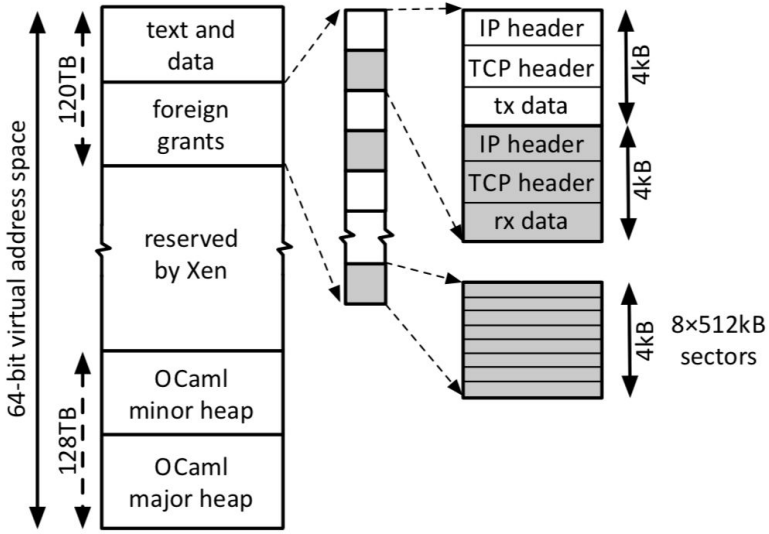


Figure 2: Specialised virtual memory layout of a 64-bit Mirage unikernel running on Xen.

Device I/O

- All device drivers are written in OCaml
- Xen devices are manipulated by writing to and reading from a shared memory page of slots
  - OCaml is used to safely access these pages
- Zero-copy I/O
  - Two VMs share a grant table that maps a page to an offset in the table that can be used to reference and access the page from another VM
  - OCaml is used to track and free contents of the grant table
    - High order functions that free resources when the wrapped function terminates or errors out

Protocol I/O

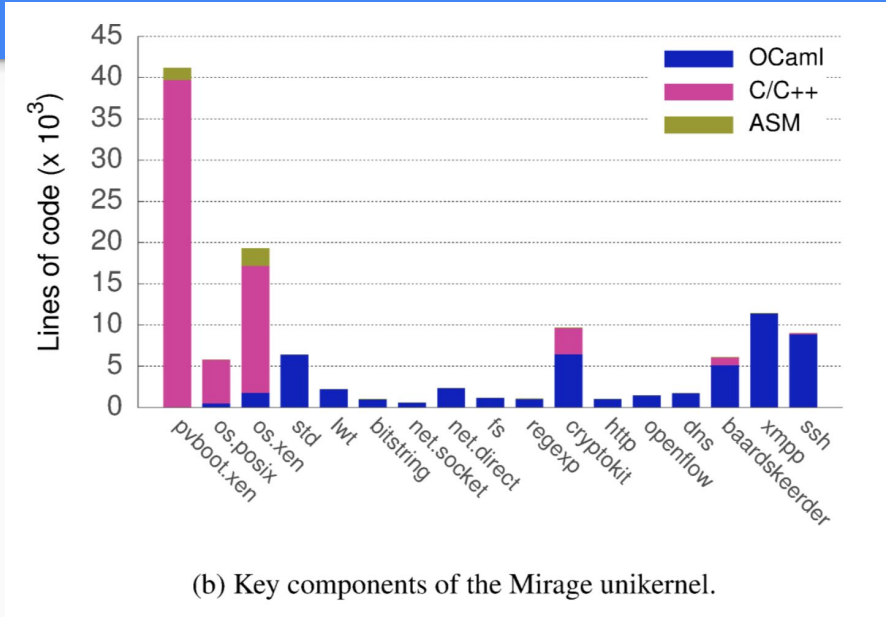
- Uses vchan for inter VM communication
  - Pages of shared buffer that other VMs can access
- Applications link to network protocol libraries that send and receive packets by reading and writing I/O pages as described before
- Same for block devices, also using I/O pages
  - Applications link to filesystem and caching libraries that can be customized

Security

- Thread model
  - Hypervisor is trusted
  - Applications trust external entities via protocols such as SSL
- Stack canaries
  - Special values placed after a buffer and monitored for modification
- Guard pages
  - Unallocated pages after allocated pages that cause segfaults when accessed
- Sealing
  - When booting, mark pages as either writable or executable but not both
  - Asks the hypervisor to disallow page table modification
- Compile time address space randomization
  - When compiling, randomly shuffle the positions of heap, stack, ...

Security

- Static type checking and garbage collected heap prevent memory errors
- Only system components needed are linked = smaller attack surface



(b) Key components of the Mirage unikernel.

Evaluation • Boot Time

- Unikernels boot fast

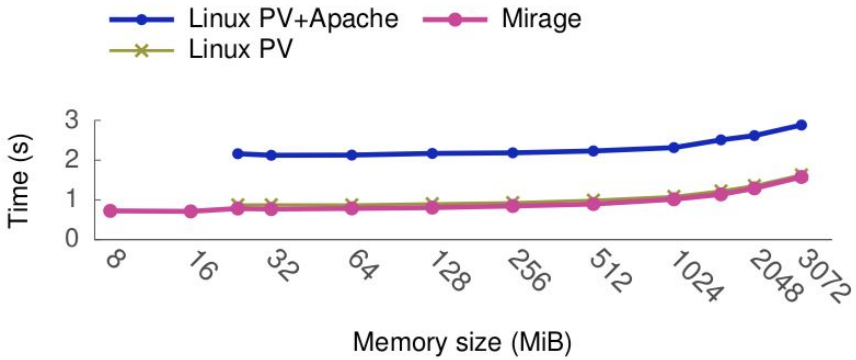


Figure 5: Domain boot time comparison.

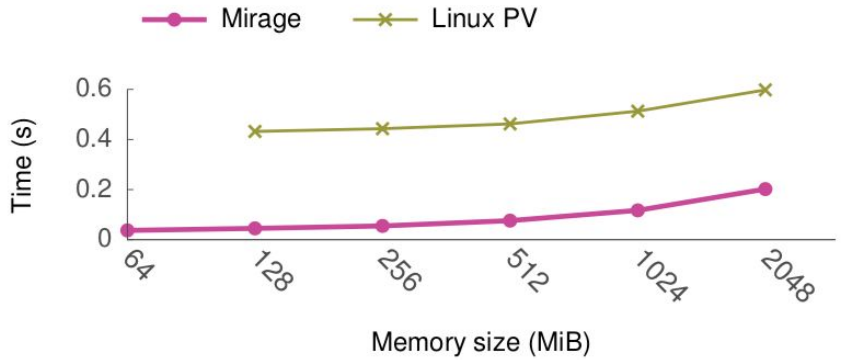
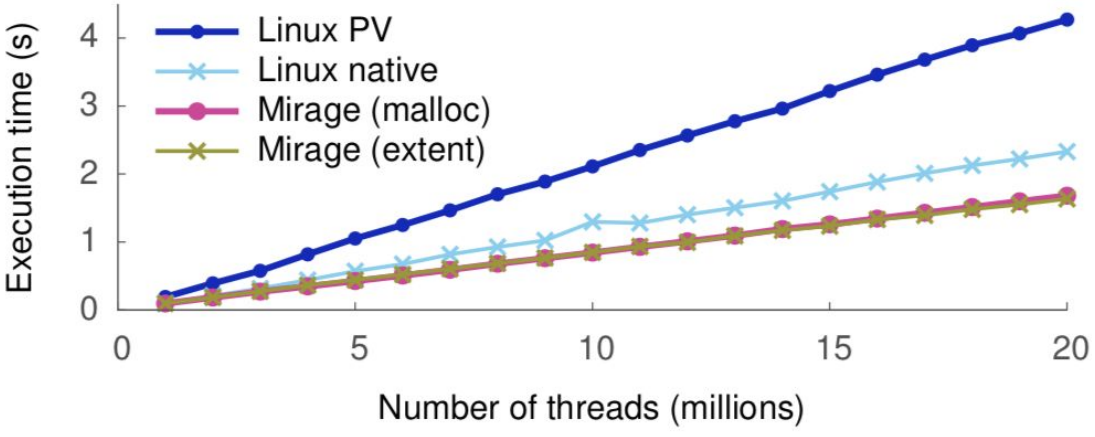


Figure 6: Boot time using an asynchronous Xen toolstack.

Evaluation • Threading

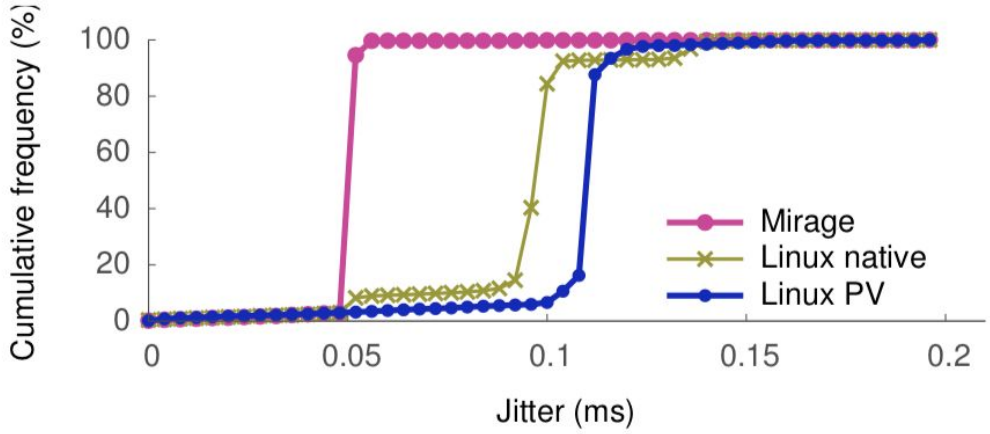
- Threads are faster to create due to garbage collected heap management



(a) Creation times.

Evaluation • Threading

- Thread timers are more accurate due to no kernel/userspace boundary



(b) Jitter for 10^6 parallel threads sleeping and waking after a fixed period.

Evaluation • Networking

- 10% higher latency due to type safety, higher receive throughput due to no userspace copying, lower transmit throughput due to higher CPU usage

Configuration	*Throughput [std. dev.] (Mbps)*			
	1 flow		10 flows	
Linux to Linux	1590	[9.1]	1534	[74.2]
Linux to Mirage	1742	[18.2]	1710	[15.1]
Mirage to Linux	975	[7.0]	952	[16.8]

Figure 8: Comparative TCP throughput performance with all hardware offload disabled.

Evaluation • Storage

- Comparable random block read throughput

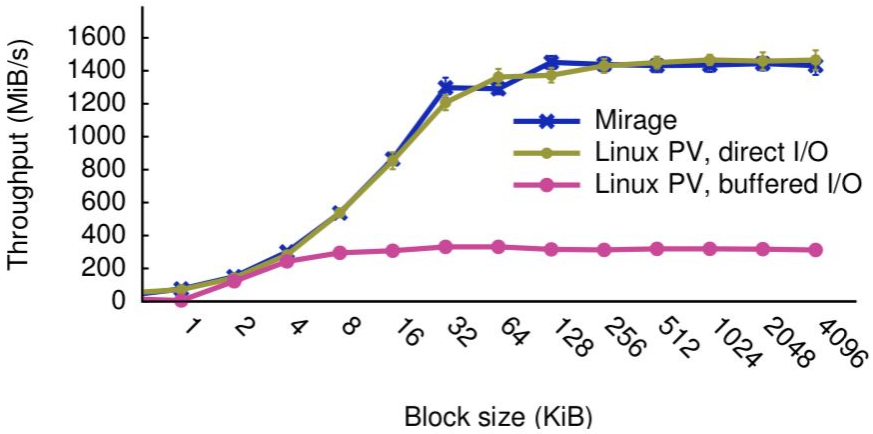


Figure 9: Random block read throughput, +/- 1 std. dev.

Evaluation • Web Application

- Mirage scales better

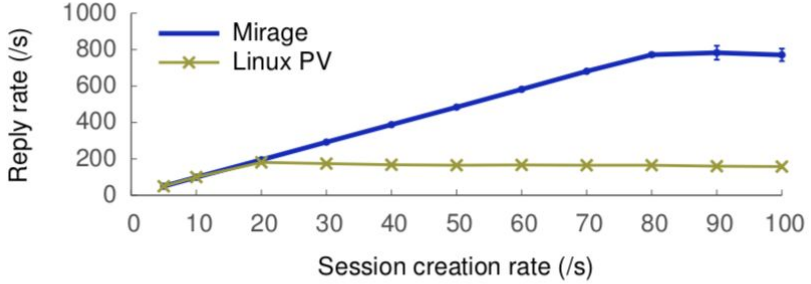


Figure 12: Simple dynamic web appliance performance.

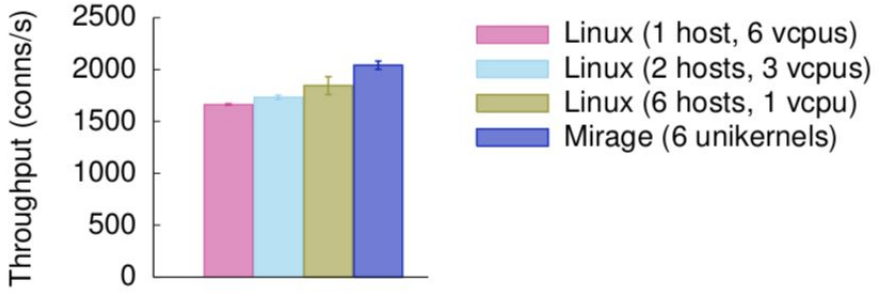
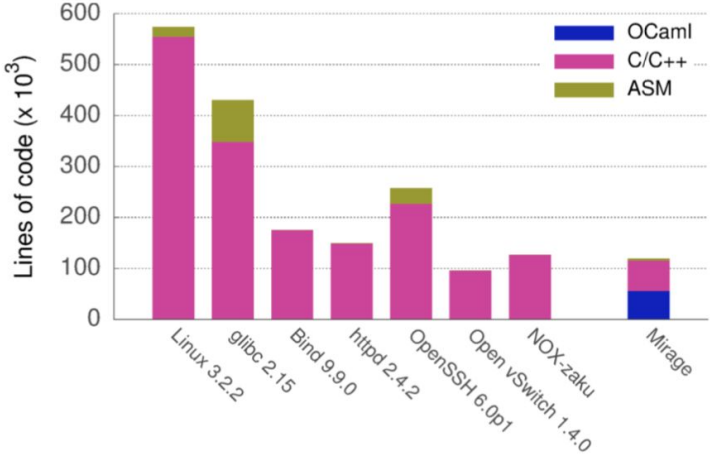


Figure 13: Static page serving performance, comparing Mirage and Apache2 running on Linux.

Evaluation • Size

- Fewer LoC, small binary size



(a) Key cloud components *vs*. the Mirage unikernel codebase.

Appliance	*Binary size (MB)*	
	Standard build	Dead code elimination
DNS	0.449	0.184
Web Server	0.673	0.172
OpenFlow switch	0.393	0.164
OpenFlow controller	0.392	0.168

Table 2: Sizes of Mirage unikernels, before and after dead-code elimination. Configuration and data are compiled directly into the unikernel.

Takeaway

- Single purpose virtual machines
- libOS and application written in one language *
- One process, one address space
- Light weight, secure, fast

Table 2. Other unikernel implementations.

Unikernel	Language	Targets
Mirage[13]	OCaml	Xen, kFreeBSD, POSIX, WWW/js
Drawbridge[17]	C	Windows “picoprocess”
HalVM[8]	Haskell	Xen
ErlangOnXen	Erlang	Xen
OSv[2]	C/Java	Xen, KVM
GUK	Java	Xen
NetBSD “rump”[9]	C	Xen, Linux kernel, POSIX
ClickOS[14]	C++	Xen

Discussion

- What are the tradeoffs between containers and unikernels? How would you choose between containers and unikernels?
- Are you satisfied with or convinced by the security that unikernels claim to provide?
- What are other potential use cases for unikernels?
- Do you agree with writing system libraries in high level language? Why is using type safe languages important for unikernels?
- MirageOS does not provide POSIX compatibility. Is it a good idea?