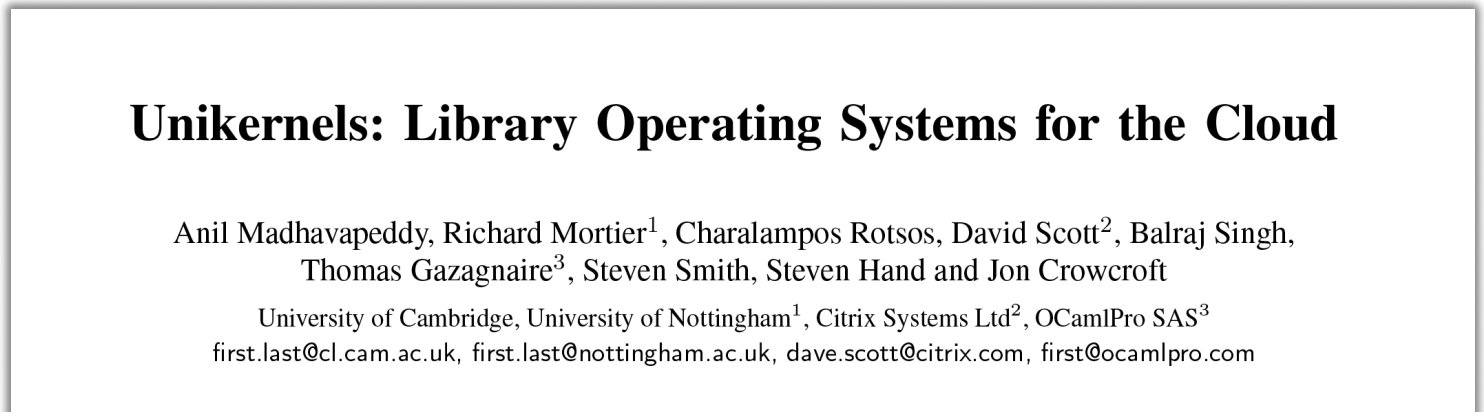


Unikernels: Library Operating Systems for the Cloud

Anil Madhavapeddy, Richard Mortier[1], Charalampos Rotsos, David Scott[2], Balraj Singh,
Thomas Gazagnaire[3], Steven Smith, Steven Hand and Jon Crowcroft
University of Cambridge, University of Nottingham[1], Citrix Systems Ltd[2], OCamlPro SAS[3]
first.last@cl.cam.ac.uk, first.last@nottingham.ac.uk, dave.scott@citrix.com, first@ocamlpro.com

Revisión de Trabajos Científicos

Unikernels: library operating systems for the cloud
ASPLOS 2013

Máster Universitario en Ingeniería Informática
M1689 - Sistemas, Virtualización y Seguridad

Jaime Iglesias Blanco

Universidad de Cantabria - Facultad de Ciencias

Noviembre 2023

Índice de contenidos

Índice de contenidos

Introducción

En servicios ofrecidos en *cloud*, cada máquina virtual (VM):

- Se presenta como un computador independiente (debido a la virtualización).
- Arranca un sistema operativo estándar y ejecuta aplicaciones sin modificar.
- Se especializa en un rol particular (servidor web, bases de datos...).
- El escalado implica la clonación a partir de un *template*.

Los **unikernels** son un enfoque para desplegar servicios en el *cloud* que:

- Tratan la imagen de la VM como una aplicación especializada en un solo propósito.
- Todas las librerías del sistema, el *runtime*, y aplicaciones se compilan en una única imagen de VM estática.
- Se puede arrancar y ejecutar en un hipervisor estándar que evita problemas de compatibilidad con el hardware.
- Ofrecen una reducción de tamaño de las imágenes y costes operativos, mayor rendimiento y seguridad.

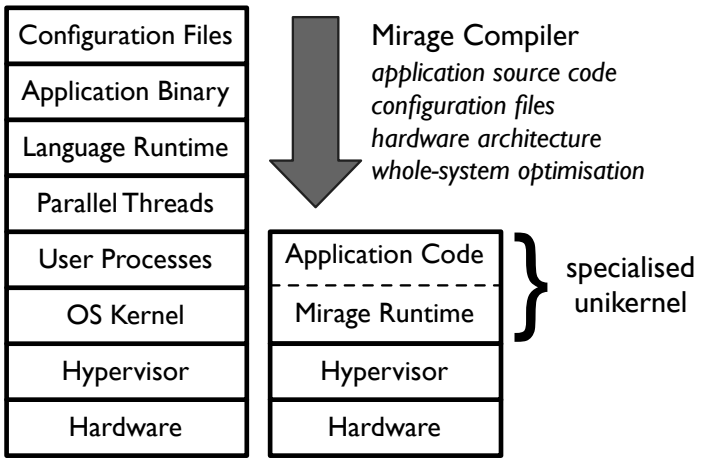


Figura 1: Capas de software en VM existentes vs. al enfoque de compilación de unikernel.

Índice de contenidos

Arquitectura

- La virtualización es la tecnología clave para el desarrollo del *cloud* y se implementa ampliamente a través de hipervisores.
- Proporcionan una abstracción de hardware virtual.
- Alto nivel de escalado dinámico.
- Un hipervisor como Xen nos permite utilizar los drivers de dispositivos virtuales.

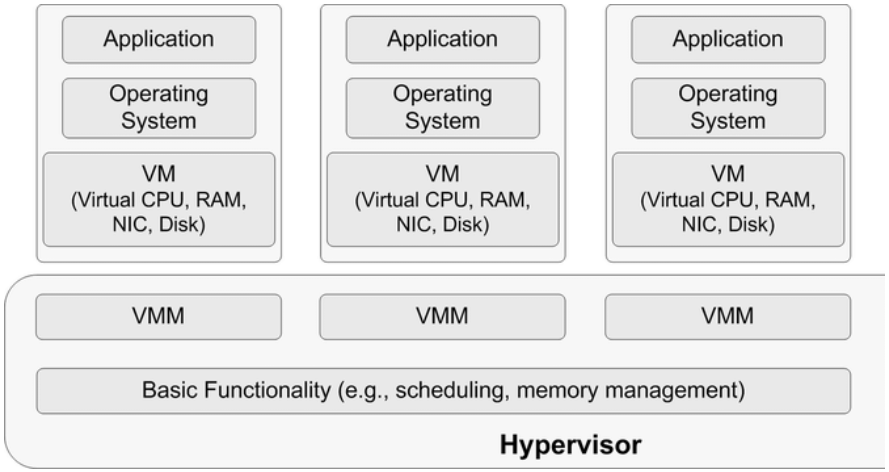


Figura 2: Arquitectura de virtualización basada en hipervisor.

Un **libOS** está estructurado de forma muy diferente a un SO convencional:

- Todos los servicios están implementados como librerías.
- Se enlazan directamente con la aplicación para su uso.

Configuración y despliegue

UC | Universidad de Cantabria

- Supone una sobrecarga considerable en la gestión del despliegue de un gran servicio alojado en *cloud*.
- Las distribuciones de Linux suelen recurrir a *shell scripting*.
- Los unikernels integran la configuración en el proceso de compilación.
- La configuración se trata como librerías dentro de una única aplicación.
- La configuración es explícita y programable en lugar de manipular muchos archivos.
- El resultado es una gran reducción del esfuerzo necesario para configurar máquinas virtuales de aplicaciones multi-servicio complejas.

Compactación y optimización

UC | Universidad de Cantabria

- Los recursos en el *cloud* se alquilan, y minimizar su uso reduce costes.
- Los unikernels enlazan bibliotecas que normalmente proporcionaría el host OS.
- Las funciones que no se utilizan en una compilación no se incluyen.
- Permite verificar el grafo de dependencias de manera sencilla.
- Producen binarios muy compactos, que pueden ser ejecutados directamente sobre el hipervisor.
- Todos los ficheros de configuración se evalúan estáticamente, lo que permite eliminar *dead-code*.
- Como contrapartida, es necesario recompilar para reconfigurar y no es posible la clonación (*copy-on-write*) de imágenes de VM, si hay configuración que debe ser única (direcciones IP estáticas).

Modelo de amenaza del Unikernel

UC | Universidad de Cantabria

- **Entorno compartido:** Se ejecuta software, accesible por red, en *datacenters* multiusuario.
- **Fiabilidad del proveedor:** Los clientes deben confiar en que el proveedor del *cloud* no es malicioso.
- **Exposición:** El software está bajo constante amenaza de ataque, desde otros servicios de clientes como usuarios conectados a Internet.
- **Confianza:** Los unikernels se ejecutan sobre un hipervisor que consideran que es completamente confiable.
- **Aislamiento:** Utilizan el hipervisor como única unidad de aislamiento y permite confiar en entidades externas siempre por medio de protocolos seguros con SSL o SSH.
- **Compatibilidad:** Los estándares para la compatibilidad de aplicaciones (POSIX API) requieren de millones de lineas de código, ejecutadas en cada arranque del sistema.
- **Código no utilizado:** Los servicios innecesarios en ejecución que pueden aumentar significativamente la superficie de ataque.

Índice de contenidos

Lenguaje OCaml

UC | Universidad de Cantabria

Ventajas:

- Lenguaje de programación funcional, imperativo y orientado a objetos.
- Sintaxis breve = Pocas lineas de código = Menor superficie de ataque.
- Fuerte tipado estático = mayor seguridad.
- La versión *open-source* de Xen Cloud Platform y los componentes críticos del sistema están implementados en OCaml, lo que facilita la integración.
- Rendimiento secuencial alto del *runtime*.
- Cada unikernel de Mirage se ejecuta en Xen utilizando una única CPU virtual, y soporta multicore comunicando múltiples unikernels en una única instancia de Xen.

Inconvenientes:

- Es necesario implementar componentes del sistema en este lenguaje (stack de red y almacenamiento).

```
string → t
let parse_static_data data =
  let open Yojson.Basic.Util in
  let static_data_as_string = Yojson.Basic.from_string data in
  let sources =
    let raw_sources = static_data_as_string ▷ member "sources" ▷ to_list in
    let par sig
    List.ma   val version : string
  in          exception Json_error of string
  let topic   val json_error : string → 'a
    let raw   type lexer_state = {                                  t in
    List.ma     buf : Bi_outbuf.t;
  in            mutable lnum : int;
  {             mutable bol : int;
    sources     mutable fname : string option;
    topics;   }
  }           module Lexer_state :
                sig
t option ref      type t =
let data: t         lexer_state = {
                    buf : Bi_outbuf.t;
t option ref → st   mutable lnum : int;
let to_stri         mutable bol : int;
  let open Yojson.Basic in
  let data_as_json d = `Assoc [
      ("sources",`List (List.map (fun source → `String source) d.sources));
      ("topics", `List (List.map (fun topic → topic ▷ Topic.to_json ) d.topics ))
    ] in
  let sdata = !data in
  match sdata with
  | Some d → d ▷ data_as_json ▷ pretty_to_string
  | None → ""

string → Topic.t
let find_topic_by_name (name: string) =
```

Librería PVBoot

- **Inicialización de VM:** PVBoot proporciona soporte para inicializar una máquina virtual (VM) con una sola CPU virtual y canales de eventos Xen.

  El objetivo es saltar a una función de entrada de la aplicación incluida en el unikernel.

- **Limitaciones en la concurrencia:** No está soportado la ejecución de varios procesos.

  Un espacio de direccionamiento de 64-bits único.

- Existen 2 *allocators* de páginas en memoria:
  - **Slab allocator:** se utiliza para spoprtar el código C en tiempo de ejecución.
  - **Extent allocator:** reserva memoria virtual contigua que manipula en trozos de 2 MB, lo que permite la asignación de superpáginas x86_64.

- **Roles de regiones de memoria:** Las regiones de memoria se asignan estáticamente a roles específicos, como el heap gestionado por el recolector de basura o las páginas de datos de E/S.

- PVBoot proporciona un soporte mínimo para una VM asíncrona y dirigida por eventos que duerme hasta que la E/S esté disponible o se agote el tiempo de espera.

Runtime del lenguaje

Gestión de la memoria:

- Los segmentos de texto y datos contienen el *runtime* de OCaml.
- El thread principal de la aplicación se lanza inmediatamente tras el arranque y la VM se apaga cuando retorna.
- El recolector de basura de OCaml divide el heap en dos regiones:
  - Minor heap: valores efímeros
    Extensión única de 2 MB (Trozos 4 KB)
  - Mayor heap: valores más duraderos
    Resto de la memoria (Trozos 2MB)

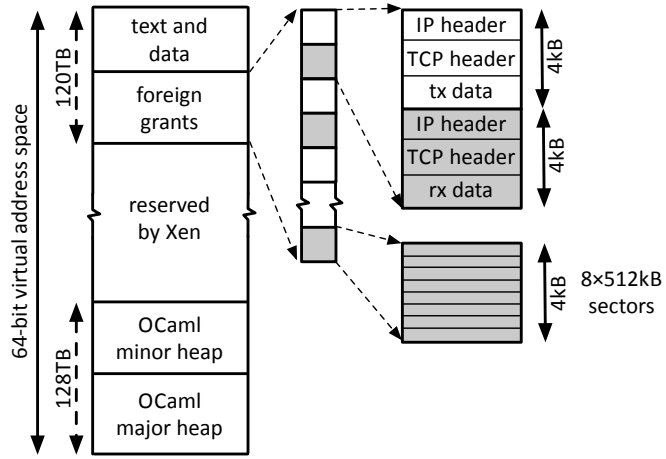


Figura 3: Disposición de memoria virtual de un unikernel Mirage de 64 bits en Xen.

Concurrencia:

- **Integración de Lwt:** Librería de threads cooperativos. Evalúa funciones bloqueantes internamente y las convierte en descriptores de eventos.
- **Implementación:** Escritos en OCaml puro, y son valores alocatados en el heap.
- Proporciona un *domainpoll* para escuchar eventos y despertar hilos ligeros.

Drivers de dispositivos

- **Interfaz con Xen:** Los drivers de Mirage se conectan a la abstracción de dispositivos proporcionada por Xen.

  Los dispositivos Xen constan de un driver frontend en la guest VM (unikernel) y un driver backend que multiplexa las peticiones frontend, normalmente a un dispositivo físico real.

- Se utiliza una única página de memoria dividida en ranuras de solicitud de tamaño fijo, que se rastrean mediante 2 punteros. Las respuestas se escriben en las mismas ranuras que las solicitudes + control de flujo.

- **Abstracción para I/O en Mirage:** La página compartida se asigna en OCaml utilizando el módulo *Bigarray*, que envuelve la memoria asignada externamente de manera segura en el heap de OCaml y la pone a disposición como un array.

- **Implementación en OCaml:** Los drivers Mirage se implementan como bibliotecas OCaml y manejan dicho array. Tienen capacidad asíncrona para enviar solicitudes y esperar respuestas.

- **Descubrimiento de errores:** La re-implementación de estos protocolos llevó a la identificación de varios errores en Linux/Xen.

Protocolo de seguridad de tipos en E/S

- **Protocolos Type-Safe:** Mirage implementa en OCaml librerías para garantizar que todas las operaciones de E/S externas sean seguras en cuanto a tipos.
- **Tratamiento de datos en paquetes:** Tanto en el stack de red como en el de almacenamiento, los datos llegan como un flujo de paquetes discretos.
- Mirage conecta paquetes y flujos utilizando iteradores de canal, que transforma streams de paquetes infinitos a streams tipados.

Conectividad por red:

- `vchan protocol` para comunicación interna: Permite que las VMs se comuniquen directamente a través de la memoria compartida (Compatible con Linux).
- Conectividad a Internet: La aplicación se enlaza con una librería de protocolo (HTTP), que a su vez se enlaza con un stack de red (TCP/IP), que a su vez se enlaza con el driver OCaml de dispositivos de red.

Almacenamiento:

- Los dispositivos de bloque comparten la misma abstracción de Ring, lo que les permite utilizar las mismas páginas de E/S para proporcionar un acceso eficiente a nivel de bloque.
- Los sistemas de archivos y la caché se implementan como librerías de OCaml, lo que proporciona un alto grado de flexibilidad y control a la aplicación.

Índice de contenidos

Microbenchmarks - Boot Time

UC | Universidad de Cantabria

Los Unikernels son suficientemente compactos para arrancar y responder al tráfico de red en tiempo real.

- **Arranque rápido de VMs:** Mirage genera VMs compactas que arrancan muy rápidamente, lo que es esencial para su eficiencia en entornos de *cloud*.
- Modificaciones en el toolstack de Xen para la construcción de dominios paralelos y aislar el tiempo de arranque de la VM.
- **Importancia de un reinicio rápido:** Abren la posibilidad de micro reinicios regulares. No penaliza tener que reiniciar para reconfigurar el unikernel.

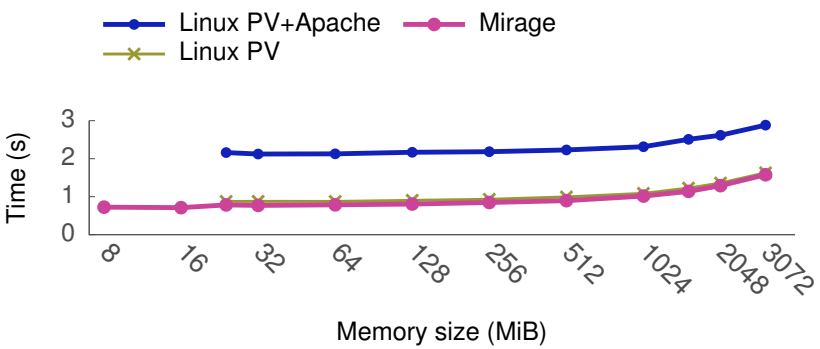


Figura 4: Comparación del tiempo de arranque.

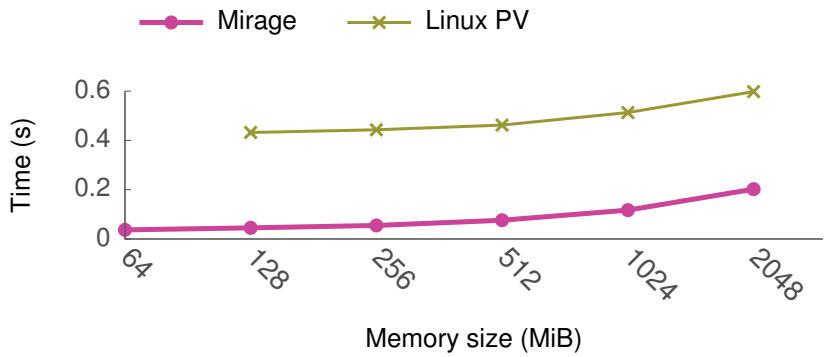


Figura 5: Tiempo de arranque utilizando una *toolstack* asíncrona de Xen.

`linux-pv`, Kernel Linux con paravirtualización en Xen.

Microbenchmarks - Threading

La recolección de basura del heap es más eficiente en un entorno de espacio de direcciones único.

- **Objetivo:** Evalúa el tiempo necesario para construir millones de threads en paralelo, donde cada subproceso duerme durante 0.5 a 1.5 segundos antes de terminar.
- **Resultados:** Ejecuciones con Mirage utilizan diferentes asignadores de memoria obtienen un mejor rendimiento debido a que la prueba está limitada por la velocidad de la recolección de basura. No hay un beneficio al utilizar superpáginas.

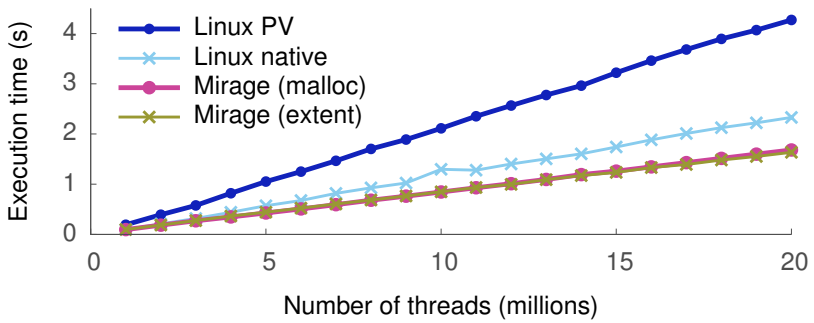


Figura 6: Tiempo de creación de threads.

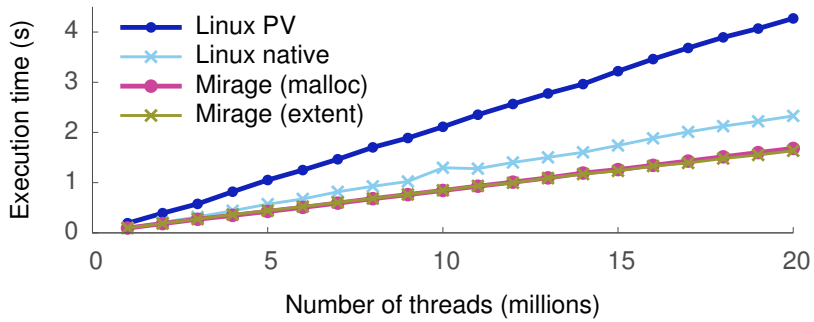


Figura 7: Jitter para 10^6 threads paralelos durmiendo y despertando tras un periodo fijo.

- **Resultados:** Ofrece una latencia más baja y más predecible al despertar a millones de threads en paralelo. Esto se debe a la falta de sobrecarga de las syscall de Linux entre espacio de usuario y kernel.

Microbenchmarks - Networking and Storage

Las redes compiten con los sistemas operativos convencionales, a pesar de ser totalmente de tipado seguro.

Prueba de latencia:

- Envio de 106 pings.
- Mirage experimentó un ligero aumento en la latencia (4-10 %) debido al overhead de la seguridad de tipos.

Prueba de Mirage TCPv4 stack:

- Utilizando la herramienta `iperf`.
- Mayor velocidad de recepción por la no copia de la página del espacio de kernel al de usuario, pero el rendimiento de envío es inferior.

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]

Figura 8: Rendimiento de TCP sin soporte hardware.

Las librerías para el manejo del almacenamiento por bloques se asemejan a el sistema por capas.

- Rendimiento de lectura aleatoria utilizando SSD PCIe.
- Tanto Linux en modo de E/S directa como Mirage tienen un rendimiento prácticamente idéntico.
- La falta de un caché de búfer no es significativa ya que las aplicaciones gestionan su propio almacenamiento.

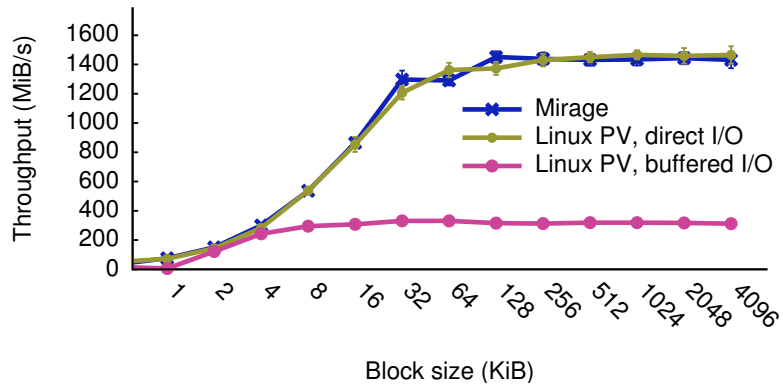


Figura 9: Rendimiento de lectura de bloques aleatorios.

Servidor DNS

- Se comparó el rendimiento del servidor DNS Mirage con dos DNS de alto rendimiento:
  - Bind (v9.9.0) logró 55,000 consultas por segundo.
  - NSD (v3.2.10) optimizada para rendimiento logró 70,000 consultas por segundo.

- El rendimiento del DNS Mirage fue malo, pero mejoró debido a la memoización de respuestas para evitar cálculos repetidos.
  - Aumentó el rendimiento de 40,000 a 75,000-80,000 consultas por segundo.

- Mejora de la compresión de etiquetas DNS.

  Proporcionó un aumento de velocidad del 20 % y protección contra ataques DoS.

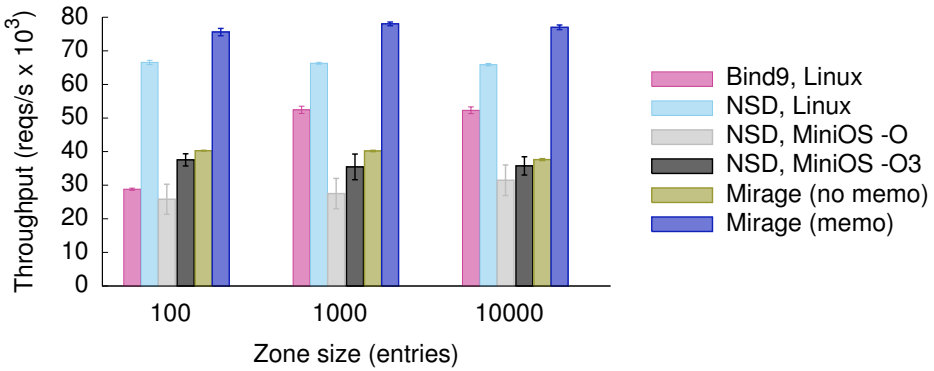


Figura 10: Rendimiento del DNS al aumentar el tamaño de las zonas.

- **Comparación de tamaño:** El servidor Mirage es más pequeño, con un tamaño de 183.5 kB en comparación con 462MB del sistema Linux en uso.

Controlador OpenFlow (SDN Ethernet Switch)

- **OpenFlow:** Es un estándar de redes definidas por software para switches Ethernet.
- **Implementación de Mirage:** Mirage proporciona librerías que implementan un analizador de protocolo OpenFlow, un controlador y un switch.
- **Aplicaciones posibles:** Las aplicaciones pueden ejercer un control directo sobre switches OpenFlow.
  - Útil cuando la aplicación ofrece funciones de capa de red (firewalls, proxies...)

- **Benchmark:** Se emularon 16 switches conectados simultáneamente al controlador.
- Se midió el rendimiento en solicitudes procesadas por segundo.
- Maestro: `Java`
- NOX: `C++`

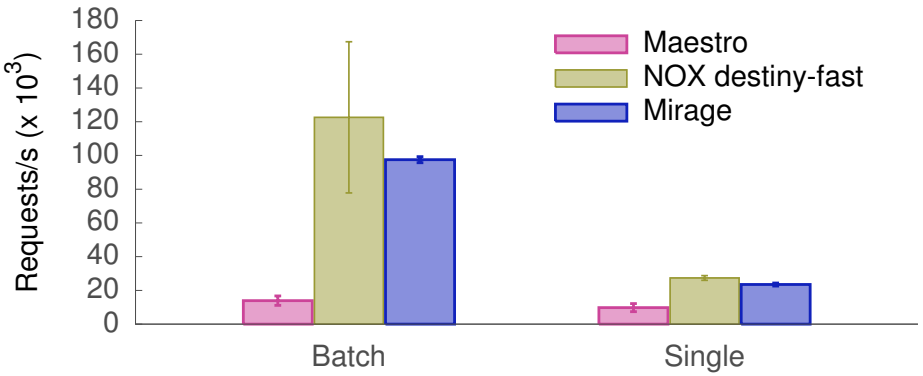


Figura 11: Rendimiento del controlador OpenFlow.

- Mirage logra la mayoría de los beneficios de rendimiento de `C++` optimizado mientras conserva las características de lenguaje de alto nivel como la seguridad de tipos.

Servidor web dinámico

UC | Universidad de Cantabria

- Implementación del servicio similar a Twitter en Mirage.
- Herramienta de evaluación de rendimiento `httperf` como cliente en la misma máquina física con núcleos dedicados separados (No overhead de Ethernet)
- Escalabilidad lineal de hasta alrededor de 80 sesiones antes de volverse limitada por la CPU.
- Huella de memoria más pequeña (32 MB frente a 256 MB), aislamiento y seguridad de tipos.

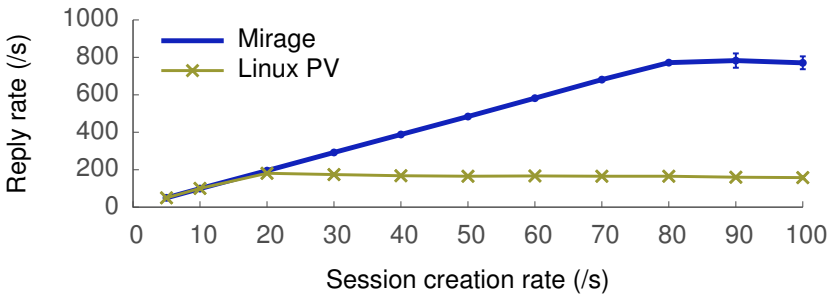


Figura 12: Rendimiento de la aplicación web dinámica simple.

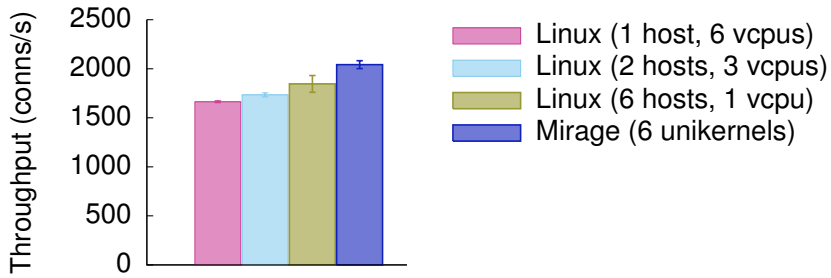


Figura 13: Rendimiento del servicio de páginas estáticas, comparando Mirage y Apache2 en Linux.

- **Comparación con Apache2 en Linux:** Linux estándar que ejecuta Apache2 con el backend Multi-Processing Module (MPM), adaptado al número de vCPUs.

Tamaño del código y binario

- Mirage demuestra una eficiencia notable al requerir menos líneas de código y producir binarios más compactos en comparación con Linux.
- La estructura de librerías de Mirage permite eliminar automáticamente dependencias no utilizadas en tiempo de compilación, lo que contribuye a su eficiencia y tamaño compacto.

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

Figura 14: Tamaños de los unikernels Mirage, antes y después de la eliminación de *dead-code*.

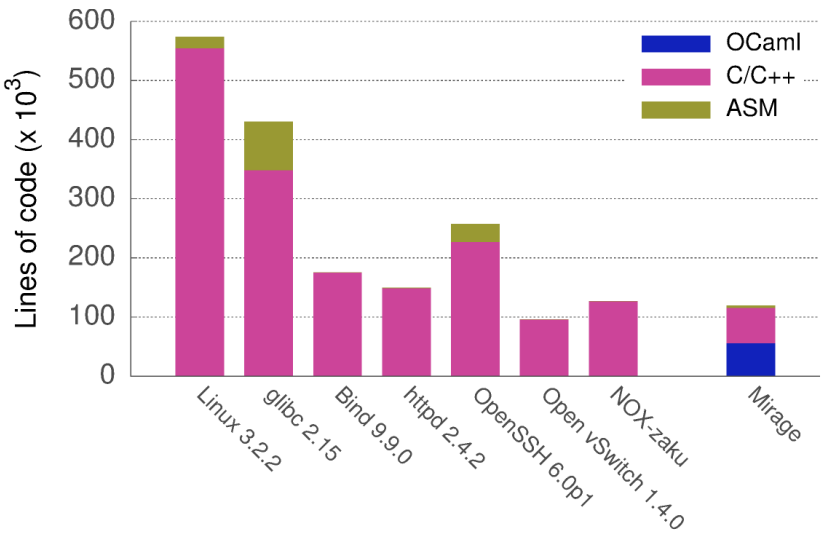


Figura 15: Servicios *cloud* vs. Código unikernel de Mirage.

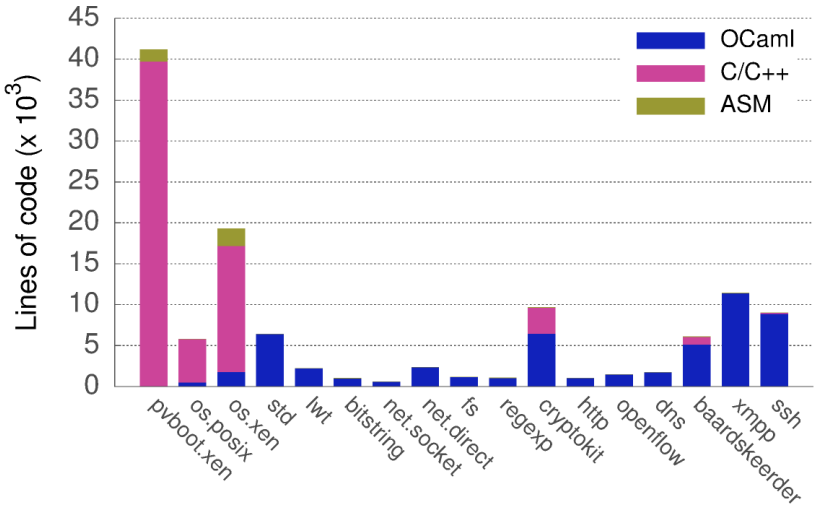


Figura 16: Componentes clave de Mirage unikernel.

Índice de contenidos

Puntos fuertes

UC | Universidad de Cantabria

- **Idea innovadora:** Primer unikernel desarrollado con capacidades competivas.
- **Practicidad:** Relevante para optimizar la eficiencia y seguridad de las aplicaciones en entornos *cloud*, muy demandados en la actualidad.
- **Enfoque en la eficiencia:** Los unikernels de Mirage se destacan por su eficiencia en términos de líneas de código y tamaño de binario. Importante en entonos *cloud* con políticas *pay-per-use*.
- **Comparaciones significativas:** El paper realiza comparaciones detalladas entre los unikernels de Mirage y sistemas operativos convencionales como Linux, destacando las ventajas de Mirage en términos de eficiencia y rendimiento.
- **Seguridad y aislamiento:** El paper resalta la seguridad y el aislamiento inherentes a los unikernels de Mirage. Importante en entornos de computación compatidos.
- **Rendimiento competitivo:** A pesar de su eficiencia en recursos, los unikernels de Mirage logran un rendimiento competitivo en comparación con sistemas operativos más tradicionales como Linux. Esto es importante para garantizar que las aplicaciones basadas en unikernels sean capaces de satisfacer las demandas de rendimiento de la nube.

Puntos débiles

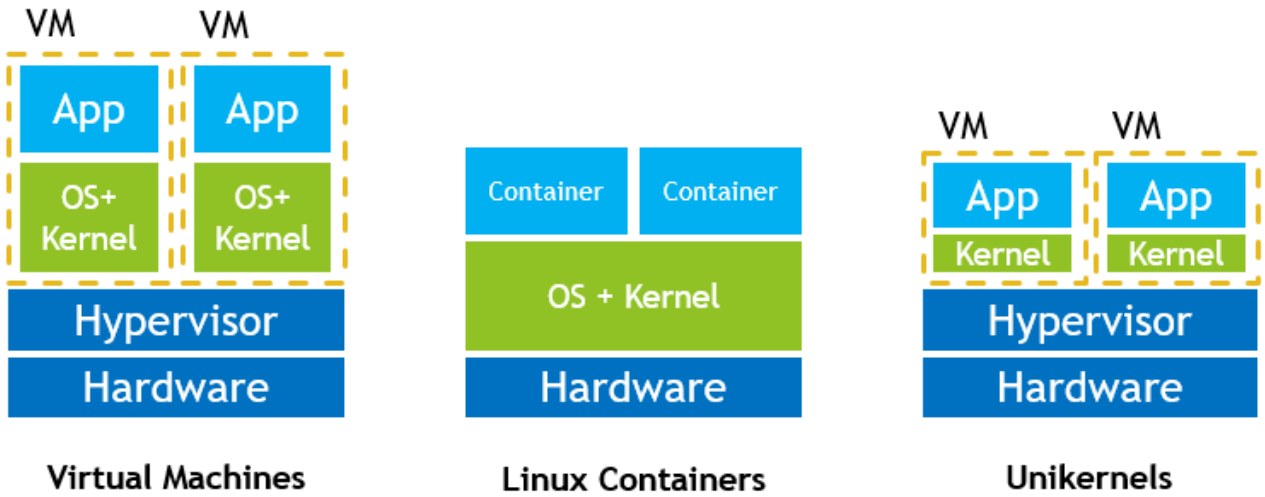
UC | Universidad de Cantabria

- **Limitación en la variedad de Aplicaciones:** Se centra en aplicaciones específicas, como servidores web y DNS. Puede ser útil explorar cómo se desempeñarían los unikernels en una variedad más amplia de aplicaciones típicas de un entorno *cloud*.
- **Limitaciones en aplicaciones interactivas:** Los unikernels están diseñados principalmente para aplicaciones sin estado y orientadas a servicios. El paper no aborda su uso con aplicaciones de interacción continua con el usuario.
- **Curva de Aprendizaje:** El paper no aborda la posible curva de aprendizaje o los desafíos asociados con la adopción de unikernels. Falta de orientación sobre cómo se pueden migrar o desarrollar aplicaciones en este nuevo paradigma.
- **Dificultad de migración:** La transición a un modelo de unikernel requiere mucho tiempo, al tener que adaptar parte de las aplicaciones y librerías ya existentes.
- **Falta de detalles sobre despliegue en producción:** El paper proporciona escasa información sobre la administración de los unikernels en entornos de producción a gran escala.
- **Necesidad de actualización:** El paper fue publicado en 2013, lo que significa que la tecnología y el panorama de el *cloud* han evolucionado desde entonces. Sería útil una actualización para abordar cómo los unikernels se han desarrollado y adoptado en el *cloud* en los años posteriores.

Perspectiva actual

UC | Universidad de Cantabria

- **Competencia con contenedores:** Con la proliferación de tecnologías de contenedores como Docker y Kubernetes, los unikernels podrían tener dificultades para competir en términos de facilidad de implementación y administración.
- **Nuevos lenguajes de programación:** `Haskell` o `Rust` ganan popularidad por su seguridad y rendimiento. Los desarrolladores pueden preferir estos lenguajes por su seguridad y facilidad de programación en lugar de `OCaml`.
- **Mayor variedad de hipervisores:** El paper se enfoca en Xen. En la actualidad hay una mayor variedad de hipervisores y tecnologías de virtualización, como KVM, VMware, Hyper-V, Virtual-Box entre otros. Los unikernels Mirage podrían necesitar adaptarse y ser compatibles con múltiples hipervisores para ser más ampliamente adoptados.



Virtual Machines | Linux Containers | Unikernels

Comentarios menores

UC | Universidad de Cantabria

"Even widely deployed codebases such as Samba and OpenSSL still contain remote code execution exploits published as recently as April 2012, and serious data leaks have become all too commonplace in modern Internet services."

- **Actualización de ejemplos:** Algunos de los ejemplos utilizados para exponer las ventajas que suponen los unikernels no son relevantes actualmente.
  - US-CERT/NIST. CVE-2012-1182, Samba before 3.6.3 (Lastest 4.19.0)
  - US-CERT/NIST. CVE-2012-2110, OpenSSL before 0.9.8v, 1.0.0 before 1.0.0i, and 1.0.1 before 1.0.1a (Lastest 3.1)

Índice de contenidos

Preguntas

Muchas gracias por vuestra atención