Immutable Distributed Infrastructure for Unikernels
WG2.8, Greece

Anil Madhavapeddy (speaker)
with Benjamin Farinier and Thomas Gazagnaire
University of Cambridge Computer Laboratory

May 26, 2015
Background

- Unikernels
- Irmin, a large-scale, immutable, branch-consistent storage

Weakly consistent data structures

- Mergeable queues
- Mergeable ropes

Benchmarking Irmin

Use Cases
• Modern systems are built in **memory-safe programming languages**.
• We build elaborate libraries and applications to express complex logic.
• ...and watch it all come crashing down when it interfaces with the OS.
Mirage OS is a library operating system that constructs unikernels for secure, high-performance network applications across a variety of cloud computing and mobile platforms.
Mirage OS is a **library operating system** that constructs unikernels for secure, high-performance network applications across a variety of cloud computing and mobile platforms.
Mirage OS is a library operating system that constructs **unikernels** for secure, high-performance network applications across a variety of cloud computing and mobile platforms.
Unikernels

Unikernel compiler

application source code

configuration files

hardware architecture

whole-system optimisation

configuration files

hardware architecture

whole-system optimisation

specialised
unikernel
What should a storage interface to unikernels look like?

- Highly distributed systems
- Frequent failure is an option
- Debugging and tracing must be built into the fabric
Irmin, large-scale, immutable, branch-consistent storage

- Irmin is a library to **persist** and **synchronize distributed data structures** both on-disk and in-memory
- It enables a style of programming very similar to the **Git workflow**, where distributed **nodes fork, fetch, merge and push** data between each other
- The general idea is that you want every active node to get a **local** (partial) **copy of a global database** and always be very explicit about how and when data is shared and migrated
Irmin Store

Tag Store

Block Store
type t = ...
(/** User-defined contents. */)
type result = [ ‘Ok of t |
   ‘Conflict of string ]

val merge : old :t → t → t → result
(/** 3-way merge functions. */)
Weakly consistent data structures
Mergeable queues
module type IrminQueue.S = sig
    type t
    type elt

    val create : unit → t
    val length : t → int
    val is_empty : t → bool

    val push : t → elt → t
    val pop : t → (elt * t)
    val peek : t → (elt * t)

    val merge : IrminMerge.t
end
Current state

<table>
<thead>
<tr>
<th>Operation</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Pop</td>
<td>2 on average</td>
<td>1 on average</td>
</tr>
<tr>
<td>Merge</td>
<td>(n)</td>
<td>1</td>
</tr>
</tbody>
</table>
### Current state

<table>
<thead>
<tr>
<th>Operation</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Pop</td>
<td>2 on average</td>
<td>1 on average</td>
</tr>
<tr>
<td>Merge</td>
<td>$n$</td>
<td>1</td>
</tr>
</tbody>
</table>

### With a little more work

<table>
<thead>
<tr>
<th>Operation</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Pop</td>
<td>2 on average</td>
<td>1 on average</td>
</tr>
<tr>
<td>Merge</td>
<td>$\log n$</td>
<td>1</td>
</tr>
</tbody>
</table>
Mergeable ropes
module type IrminRope.S = sig
  type t
  type value (* e.g. char *)
  type cont (* e.g. string *)

  val create : unit → t
  val make : cont → t
  ...
  val set : t → int → value → t
  val get : t → int → value
  val insert : t → int → cont → t
  val delete : t → int → int → t
  val append : t → t → t
  val split : t → int → (t * t)

  val merge : IrminMerge.t
end
<table>
<thead>
<tr>
<th>Operation</th>
<th>Rope</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set/Get</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Split</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Concatenate</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Merge</td>
<td>$\log (f(n))$</td>
<td>$f(n)$</td>
</tr>
</tbody>
</table>
Weakly consistent data structures

Mergeable ropes

Anil Madhavapeddy (speaker) with Benjamin Farinier and Thomas Gazagnaire
University of Cambridge Computer Laboratory
Weakly consistent data structures

Mergeable ropes

Anil Madhavapeddy (speaker) with Benjamin Farinier and Thomas Gazagnaire

University of Cambridge Computer Laboratory

Immutable Distributed Infrastructure for Unikernels

19 / 28
Weakly consistent data structures

Mergeable ropes
Weakly consistent data structures

Mergeable ropes

Anil Madhavapeddy (speaker) with Benjamin Farinier and Thomas Gazagnaire
University of Cambridge Computer Laboratory

19 / 28
Weakly consistent data structures

Mergeable ropes

Anil Madhavapeddy (speaker) with Benjamin Farinier and Thomas Gazagnaire

University of Cambridge Computer Laboratory

Immutable Distributed Infrastructure for Unikernels

19 / 28
Benchmarking Irmin
Benchmarking Irmin

Anil Madhavapeddy (speaker) with Benjamin Farinier and Thomas Gazagnaire
University of Cambridge Computer Laboratory

Immutable Distributed Infrastructure for Unikernels
module ObjBackend ... = struct
  type t = unit
  type key = K.t
  type value = V.t

  let create () = return ()
  let clear () = return ()

  let add t value =
    return (Obj.magic (Obj.repr value))

  let read t key =
    return (Obj.obj (Obj.magic key))

  let mem t key = return true
  ...
end
Benchmarking Irmin

IrminMemory backend

IrminGit.Memory backend

Write time estimation (µs)

Read time estimation (µs)
Status

- HTTP REST APIs for remote clients.
- JavaScript compilation for pure browser operation.
- Bidirectional lenses (Git commits map to Irmin commits from any direction)
- Open source at https://github.com/mirage/irmin
- JFLA 2015 paper got great feedback. Now building more data structures, evaluating block scheduling.
• **Jitsu: Just-In-Time Summoning of Unikernels**
  [https://www.youtube.com/watch?v=DSzvFwIVm5s](https://www.youtube.com/watch?v=DSzvFwIVm5s)
• Ported complex Xen toolstack to use Irmin.
• Jitsu becomes the *inetd* of Xen:
  • Launch VMs in response to network requests in real-time.
  • Irmin coordinates toolstack RPCs with low latency.
  • Connection setup is proxied to eliminate packet loss.
Use Cases

DNS
TCP
HTTP
Jitsud
Synjitsu
Client
Jitsu
VM boot
domain builder
Netfront
Xen Unikernel
query
answer
SYN
ACK
Xenstore
libxl
boot
plug
stop
record
GET
HTTP response
1
2
3
4
5

Anil Madhavapeddy (speaker) with Benjamin Farinier and Thomas Gazagnaire
University of Cambridge Computer Laboratory
Immutable Distributed Infrastructure for Unikernels
27 / 28