Virtualization at the Edge

Kyle C. Hale

Laboratory for High-Performance Experimental Systems and Architecture (HExSA)

Argonne/IIT Research Seminar - March 30, 2022
The Cloud is Decentralizing: Services and Infrastructure at the Edge

Kyle C. Hale
The Cloud is Decentralizing: Services and Infrastructure at the Edge

centralized cloud

edge nodes

user devices
Resources are becoming *disaggregated* in the datacenter

<table>
<thead>
<tr>
<th>CPUs</th>
<th>RAM</th>
<th>GPU</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPUs</th>
<th>RAM</th>
<th>GPU</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPUs</th>
<th>RAM</th>
<th>GPU</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*high-performance interconnect*
Resources are becoming **disaggregated** in the datacenter.

![Diagram showing disaggregation of resources in the datacenter.](Image)

- High-performance interconnect

Kyle C. Hale
Resources are becoming **disaggregated** in the datacenter

![Diagram showing disaggregated resources]

- CPU
- RAM
- GPU
- FPGA

High-performance interconnect

User 1

User 2
Composable Infrastructure

source: https://www.liqid.com/why-liqid/benefits


To date, there have been no compelling mass-market applications that require low latencies that cannot be achieved as a web service and that also are too computationally or energy intensive for modern smart phones to run locally. This might be a “chicken or the egg” problem: the lack of cyber foraging infrastructure could potentially be hindering the development of such applications. Later, we discuss one emerging class of applications that could prove to be the compelling application that cyber foraging needs.

What’s Changed?

• Virtualization technology has improved significantly
• Infrastructure provisioning has become more sophisticated (NB serverless research)
• Composable infrastructure
• Hardware design more democratic
• AR/VR/XR is here
...also, wireless latency continues to drop
Coalescent Computing
Ephemeral Single-System Image at the Edge
Coalescent Computing
Ephemeral Single-System Image at the Edge

User nears physical proximity of edge system
Coalescent Computing
Ephemeral Single-System Image at the Edge

Resources coalesced into one logical system
Coalescent Computing
Ephemeral Single-System Image at the Edge

User leaves environment, resources relinquished
Coalescent Computing
Ephemeral Single-System Image at the Edge

User approaches another edge system
Coalescent Computing
Ephemeral Single-System Image at the Edge

Resources coalesced again, subject to performance, policy constraints

[Hale, Coalescent Computing, APSys ‘21]
Coalescent System Software

Distributed VM

User monitor

CPUs
RAM

User device

Wireless network

Data plane

Control plane

Edge system

Resource monitor

Distributed VM

CPUs
RAM
GPU
FPGA

Applications

Control plane

Data plane

Wireless network

Applications

Control plane

Data plane

Wireless network
Current Work

- Adapting DSM-based approaches to the edge (e.g., GiantVM)
- Building a prototype co-designed hypervisor/OS for CC
- Applying PL techniques for coalescent offloading policies (collab. with Stefan Muller)
Virtines

Isolating Functions at the Hardware Limit (to appear in EuroSys ‘22)
Developer-Friendly, Fast, Function Isolation

• Function isolation
  • Serverless/FaaS – containers, vms
  • DB UDF - high level languages

• Low latency Startup, Short Lived Runtime
  • Spawn and manage many functions w/o significant impact

• Easy programming interface
What might function isolation look like?

- Ephemeral state by default (call stack)
- `return` destroys the context
Macro Goals

Function
Granularity

Low Latency
Startup

Easy to integrate
Virtines: Virtual Subroutines

- **Hardware-virtualized** isolated functions
- **Microsecond** level boot times
- General purpose
The lower bounds of virtualization
It’s not the hardware that’s expensive...

• What is the latency of a VM creation?
  • HW/SW state

SW VM Allocation is **expensive**

VM Interaction is **cheap**

(On AMD EPYC 7281)
Bootstrapping woes

- Real Mode: ~100 cycles
  - Setup stack, Setup segment registers

- Protected Mode: ~10,000 cycles
  - Load GDT, Mode switch, long jump

- Long Mode: ~50,000 cycles
  - Page Mapping, CR3 Write, Mode Switch, Long Jump
Traditional I/O is very expensive

- Try to make a VM feel like real hardware
  - Requires large device drivers
  - Lots of VM Exits for single ops (Expensive!)

- Paravirtualization
  - Codesigned, VM aware of the Hypervisor
  - I/O via hypercalls
HTTP server using hypercalls

- VM
  - 10k: recv()
  - 85k: send()
  - 175k:

- Hypervisor

- Linux
  - BLOCKED

Time (cycles)
Wasp: an implementation of Virtines
Wasp

• A micro-hypervisor library
• Abstracts hardware specific interfaces
• Lean

• Heavily optimized
Allocating virtines with Wasp

Virtine Creation

YES
Load Binary
Enter desired mode (x64)
Virtine Library Init
Take Snapshot

NO

First Execution?

Virtine Code Execution
Wasp is close to the hardware limit
C Language Extension

```c
void db_run_udf(db_udf_t *code) {
  // ...
}
```

- Default-deny access to host services
- Custom LLVM module pass to compile and manage virtines
Duktape JavaScript Engine

• **Duktape** JavaScript Engine: [https://duktape.org/](https://duktape.org/)
  • Embeddable, Portable
Do our latency optimizations work?

Time

Baseline

Duktape Init

Duktape Run

Duktape Free

419us
Effectiveness of language extensions

Baseline: **50us**, σ≈10us

Virtines: **80us**, σ≈30us
How easy is it to integrate?

• ~20 lines of code changed
  • Mostly glue logic
• Significantly slower... But we expect that
• 21kb

```c
virtine void do_vpaes_cbc_encrypt(
    struct virtine_aes_state *state,
    const AES_KEY key,
    int encrypt)
{
    // ...
}
```
Serverless Virtines

• OpenWhisk/AWS Lambda are good examples of modern serverless platforms
• Weak isolation between function instances!
• We developed our own using vitrines (based on OW)
Serverless Platform Interface
We can use virtines as drop-in replacement for containers!
Isolating Functions at the Hardware Limit with Virtines

Nicholas C. Wanninger*  
ncw@u.northwestern.edu  
Northwestern University  
Evanston, Illinois, USA

Joshua J. Bowden  
jbowden@hawk.iit.edu  
Illinois Institute of Technology  
Chicago, Illinois, USA

Kirtankumar Shetty  
kshetty11@hawk.iit.edu  
Illinois Institute of Technology  
Chicago, Illinois, USA

Ayush Garg  
agarg22@hawk.iit.edu  
Illinois Institute of Technology  
Chicago, Illinois, USA

Kyle C. Hale  
khale@cs.iit.edu  
Illinois Institute of Technology  
Chicago, Illinois, USA

Abstract
An important class of applications, including programs that leverage third-party libraries, programs that use user-defined functions in databases, and serverless applications, benefit from isolating the execution of untrusted code at the granularity of individual functions or function invocations. However, existing isolation mechanisms were not designed for this use case; rather, they have been adapted to it. We introduce

Keywords: virtines, virtualization, isolation

ACM Reference Format:
Thanks!

- HExSA Lab: hexsa.halek.co
- Thanks to MD Ali, Conghao Liu, Brian Tauro, Stefan Muller
- Virtines code available: github.com/virtines
- Thanks to Nick Wanninger (virtines lead author)
  - Email: ncw@u.northwestern.edu
  - Website: https://nickw.io
- Also Josh Bowden, Kirtan Shetty, Ayush Garg