Making Serverless Computing More Serverless

Zaid Al-Ali, Sepideh Goodarzy, Ethan Hunter, Sangtae Ha, Richard Han, Eric Keller

*Views not representative of IBM policy, products, or strategies*
Serverless Background

• Serverless offerings are widespread today

Amazon Lambda

Google Cloud Functions

IBM Cloud Functions

Apache OpenWhisk

OpenFaaS

fission

Dispatch
Today’s serverless abstraction

Making Serverless Computing More Serverless

FaaS, Lamda, OpenWhisk, ...

Break limitations of single server

Zaid Al-Ali†, Sepideh Goodarzy†, Ethan Hunter†, Sangtae Ha†, Richard Han†, Eric Keller†, Eric Rozner*
†University of Colorado Boulder; *IBM Research

Abstract—In serverless computing, developers define a function to handle an event, and the serverless framework horizontally scales the application as needed. The downside of this function-based abstraction is it limits the type of application supported and places a bound on the function to be within the physical resource limitations of the server the function executes on. In this paper we propose a new abstraction through memory or storage. The challenge, of course, is realizing a process-based serverless framework which can map our serverless process abstraction to an underlying, physically distributed infrastructure. To that end, we propose a new architecture called ServerlessOS to enable our vision and argue three key components are necessary to make our

Low cost

Function instance bound to server

This talk: propose new serverless abstraction and overview its design
A new serverless abstraction

- Expand serverless beyond the bounds of FaaS
- Goals of our new abstraction:
  - Flexible enough to support general set of applications
  - Familiar to developers, operating systems, and admins
  - Easy to transition existing codebases to serverless
  - Same simplicity and scale-out as FaaS
A new serverless abstraction

- Flexible enough to support general set of applications
- Familiar to developers, operating systems, and admins
- Easy to transition existing codebases to serverless
- Same simplicity and scale-out as FaaS
Flexible enough to support general set of applications

- Multiple threads, I/O via sockets, persist state, ...

Familiar to developers, operating systems, and admins

- Abstraction already used today in non-serverless

Easy to transition existing codebases to serverless

- Pool of CPU, I/O, memory, storage: server → datacenter

Challenge: map serverless process abstraction to underlying physically distributed architecture
Outline of talk

• Goal: provide seamless, scale-out process abstraction

• This talk: high-level outline of our ServerlessOS vision

ServerlessOS

Fluid Multi-resource Disaggregation
Break coupling between process & underlying physical server’s resources

Fine-grained Live Orchestration
Monitor and allocate resources across infrastructure

Coordinated Isolation
Provide data privacy and resource isolation
Outline of talk

• Goal: provide seamless, scale-out process abstraction

• This talk: high-level outline of our ServerlessOS vision

- Fluid Multi-resource Disaggregation
  Break coupling between process & underlying physical server’s resources

- Fine-grained Live Orchestration
  Monitor and allocate resources across infrastructure

- Coordinated Isolation
  Provide data privacy and resource isolation
Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

Fluid multi-resource **disaggregation**

Disaggregation: decouple process’s resources from single server
Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

Fluid **multi-resource** disaggregation

Decouple memory, compute, I/O to increase flexibility
Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

**Fluid** multi-resource disaggregation

Aaccessing remote memory incurs much higher overhead than local memory
Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

**Fluid** multi-resource disaggregation

Fluidity: allow process to move to data when more efficient
Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

**Fluid** multi-resource disaggregation

Fluidity: enable process to exploit locality to improve performance
Fluidity over multiple resources

Already provided by prior works (RamCloud, DSM, InfiniSwap, …)

Move processing to data or other server with more compute resources (Initial results show 2-3x speedup over a DSM scheme)

Decouple device that captured I/O from device that will process I/O. Additionally, move I/O to more bandwidth. (CPU fluidity can move processing with socket)
Outline of talk

- Goal: provide seamless, scale-out process abstraction
- This talk: high-level outline of our ServerlessOS vision
Fine-grained live orchestration layer

- Monitor, allocate, and optimize run-time performance by automatically assigning, migrating, or scaling workloads

Global decision making

Determine subset of nodes (and resources) available to process $P$
Fine-grained live orchestration layer

- Monitor, allocate, and optimize run-time performance by automatically assigning, migrating, or scaling workloads.

Both decisions influenced by state of other local nodes:
- CPU
- Memory
- Network

Backpressure algorithm avoids poor decisions.

Local decision making

- When to expand (or contract)
- Where to expand (or contract)
Goal: provide seamless, scale-out process abstraction

This talk: high-level outline of our ServerlessOS vision

Outline of talk

- Fluid Multi-resource Disaggregation
  - Break coupling between process & underlying physical server resources
- Fine-grained Live Orchestration
  - Monitor and allocate resources across infrastructure
- Coordinated Isolation
  - Provide data privacy and resource isolation
Coordinated Isolation

Data Privacy

Ensure application cannot read or write state from another application

Resource Isolation

Bound CPU, memory, storage, and network usage of workloads

Linux Kernel

namespaces

control groups

ServerlessOS: extend isolation across multiple servers in coordinated fashion
Coordinated Isolation

**namespaces**
Extend process namespace across multiple servers

**control groups**
Centralize state in orchestration layer, but minimize overheads

Global store of CPU shares

From single server...

Orchestration layer

... to ServerlessOS
Outline of talk

• Goal: provide seamless, scale-out process abstraction

• This talk: high-level outline of our ServerlessOS vision

**ServerlessOS**

- **Fluid Multi-resource Disaggregation**
  Break coupling between process & underlying physical server’s resources

- **Fine-grained Live Orchestration**
  Monitor and allocate resources across infrastructure

- **Coordinated Isolation**
  Provide data privacy and resource isolation
Conclusions

- New abstraction for serverless: a seamless, scale-out process
- High-level overview of ServerlessOS architecture
  - Fluid multi-resource disaggregation
  - Fine-grained live orchestration layer
  - Coordinated isolation
- Complementary to current serverless techniques
- Next steps: refine design, build prototype, conquer the world!
- Thanks! mailto: erozner@us.ibm.com