What is Serverless?

• Very attractive abstraction:
  • Pay for Use
  • Infinite elasticity from 0 (and back)
  • No worry about servers
    • Provisioning, Reserving, Configuring, patching, managing

• Most popular offering: Function-as-a-Service (FaaS)
  • Bounded-time functions with no persistent state among invocations
  • Upload code, get an endpoint, and go

For the rest of this talk, Serverless = Serverless FaaS
## What is Serverless?

<table>
<thead>
<tr>
<th></th>
<th>Bare Metal</th>
<th>VMs (IaaS)</th>
<th>Containers</th>
<th>Functions (FaaS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit of Scale</strong></td>
<td>Server</td>
<td>VM</td>
<td>Application/Pod</td>
<td>Function</td>
</tr>
<tr>
<td><strong>Provisioning</strong></td>
<td>Ops</td>
<td>DevOps</td>
<td>DevOps</td>
<td>Cloud Provider</td>
</tr>
<tr>
<td><strong>Init Time</strong></td>
<td>Days</td>
<td>~1 min</td>
<td>Few seconds</td>
<td>Few seconds</td>
</tr>
<tr>
<td><strong>Scaling</strong></td>
<td>Buy new hardware</td>
<td>Allocate new VMs</td>
<td>1 to many, auto</td>
<td>0 to many, auto</td>
</tr>
<tr>
<td><strong>Typical Lifetime</strong></td>
<td>Years</td>
<td>Hours</td>
<td>Minutes</td>
<td>O(100ms)</td>
</tr>
<tr>
<td><strong>Payment</strong></td>
<td>Per allocation</td>
<td>Per allocation</td>
<td>Per allocation</td>
<td>Per use</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>Anywhere</td>
<td>Anywhere</td>
<td>Anywhere</td>
<td>Elsewhere</td>
</tr>
</tbody>
</table>
Serverless

“...more than 20 percent of global enterprises will have deployed serverless computing technologies by 2020.”
Gartner, Dec 2018

Serverless — the future of software architecture?
The future is transitioning from 3-tiered architectures to thick-client apps connected to cloud-based microservice functions

Survey Shows More than 75% Use or Plan to Use Serverless in Next 18 Months
Serverless Computing: One Step Forward, Two Steps Back

Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov and Chenggang Wu
UC Berkeley
jhellerstein, jmfaleiro, jgonzal, ismith atcs.eecs.berkeley.edu

Peeking Behind the Curtains of Serverless Platforms

Liang Wang 1, Mengyuan Li 2, Yinqian Zhang 2, Thomas Ristenpart 3, Michael Swift 1

1UW-Madison, 2Ohio State University, 3Cornell Tech

Cloud Programming Simplified:
A Berkeley View on Serverless Computing

Eric Jonas
Andraag Khandelwal
Karl Krauth
Johann Schleier-Smith
Qifan Pu
Neeraj Yadavkar
Iou Sotico
Vikram Sreekanti
Vaishaal Shanikar
Joseph E. Gonzalez
David A. Patterson
Chia-Chie Tsai
Joao Carreira
Raluca Ada Popa

UC Berkeley
serverlessview@berkeley.edu

"... we predict that (...) serverless computing will grow to dominate the future of cloud computing."
So what are people doing with FaaS?

• Many simple things
  • ETL workloads
  • IoT data collection / processing
  • Stateless processing
    • Image / Video transcoding
    • Translation
    • Check processing
  • Serving APIs, Mobile/Web Backends

• Interesting Explorations
  • MapReduce (pywren)
  • Linear Algebra (numpywren)
  • ExCamera
  • gg “burst-parallel” functions apps
  • ML training

• Limitations
  • Communication
  • Latency
  • Locality (lack)
  • State management
What is Serverless?

• Very attractive abstraction:
  • Pay for Use
  • Infinite elasticity from 0 (and back)
  • No worry about servers
    • Provisioning, Reserving, Configuring, patching, managing
If you are a cloud provider...

• A big challenge
  • You do worry about servers!
    • Provisioning, scaling, allocating, securing, isolating
  • Illusion of infinite scalability
  • Optimize resource use
  • Fierce competition

• A bigger opportunity
  • Fine grained resource packing
  • Great space for innovating, and capturing new applications, new markets
Cold Starts

- Typically range between 0.2 to a few seconds\(^1,2\)

---

\(^1\) [https://levelup.gitconnected.com/1946d32a0244](https://levelup.gitconnected.com/1946d32a0244)

\(^2\) [https://mikhail.io/serverless/coldstarts/big3/](https://mikhail.io/serverless/coldstarts/big3/)
Cold Starts and Resource Wastage

Keeping functions in memory indefinitely.

Removing function instance from memory after invocation.
Stepping Back: Characterizing the Workload

- How are functions accessed
- What resources do they use
- How long do functions take

2 weeks of all invocations to Azure Functions in July 2019

First characterization of the workload of a large serverless provider

Subset of the traces available for research:
https://github.com/Azure/AzurePublicDataset
Invocations per Application*

This graph is from a representative subset of the workload. See paper for details.
This graph is from a representative subset of the workload. See paper for details.
Invocations per Application

This graph is from a representative subset of the workload. See paper for details.
Invocations per Application

This graph is from a representative subset of the workload. See paper for details.
Invocations per Application

This graph is from a representative subset of the workload. See paper for details.
This graph is from a representative subset of the workload. See paper for details.
Invocations per Application

- 18% >1/min
- 99.6% of invocations!
- 82% <1/min
- 0.4% of invocations

This graph is from a representative subset of the workload. See paper for details.
Apps are highly heterogeneous
What about memory?

If we wanted to keep all apps warm...

![Graph showing cumulative fraction of total memory versus fraction of least invoked apps with data points for allocated memory and physical memory.]
What about memory?

If we wanted to keep all apps warm...

- 82% of apps -> 0.4% of invocations -> 40% of all physical memory, 60% of virtual memory
- 90% of apps -> 1.05% of invocations -> 50% of all physical memory
Function Execution Duration

• Executions are short
  • 50% of apps on average run for <= 0.67s
  • 75% of apps run for <= 10s max

• Times at the same scale as cold start times\(^1,2\)

\(^1\)https://levelup.gitconnected.com/1946d32a0244
\(^2\)https://mikhail.io/serverless/coldstarts/big3/
Key Takeaways

• Highly concentrated accesses
  • 82% of the apps are accessed <1/min on average
  • Correspond to 0.4% of all accesses
  • But in aggregate would take 40% of the service memory if kept warm

• Arrival processes are highly variable

• Execution times are short
  • Same OOM as cold start times
Cold Starts and Resource Wastage

Keeping functions in memory indefinitely.

Removing function instance from memory after invocation.
What do serverless providers do?

Amazon Lambda

- Fixed 10-minute keep-alive.

Azure Functions

- Fixed 20-minute keep-alive.

Mikhail Shilkov, Cold Starts in Serverless Functions, [https://mikhail.io/serverless/coldstarts/](https://mikhail.io/serverless/coldstarts/)
Fixed Keep-Alive Policy

Results from simulation of the entire workload for a week.
Fixed Keep-Alive Won’t Fit All

- **Cold Start**: 8 mins
- **Warm Start**: 11 mins

10-minute Fixed Keep-alive

- **Cold Start**
- **Warm Start**
Fixed Keep-Alive Is Wasteful

Function image kept in memory but not used.
Hybrid Histogram Policy

Adapt to each application

Pre-warm in addition to keep-alive

Lightweight implementation
A Histogram Policy To Learn Idle Times

Idle Time (IT): 8 mins

Frequency

Time

10-minute
Fixed
Keep-alive

Cold Start

Warm Start
A Histogram Policy To Learn Idle Times

Pre-warm

Keep-alive

Frequency

Idle Time (IT)
A Histogram Policy To Learn Idle Times

Frequency

Pre-warm

5th percentile

Keep-alive

99th percentile

Minute-long bins

Limited number of bins (e.g., 240 bins for 4-hours)
The Hybrid Histogram Policy

We can afford to run complex predictors given the low arrival rate. A histogram might be too wasteful.
The Hybrid Histogram Policy

ARIMA: Autoregressive Integrated Moving Average
More Optimal Pareto Frontier

![Graph showing normalized wasted memory time versus 3rd quartile app cold start percentage, with markers for different time intervals and hybrid time settings.](image)
Implemented in OpenWhisk

- Open-sourced industry-grade (IBM Cloud Functions)
- Functions run in docker containers
- Uses 10-minute fixed keep-alive
- Built a distributed setup with 19 VMs
Simulation

4-Hour Hybrid Histogram

Experimental

Average exec time reduction: 32.5%

99\textsuperscript{th}–percentile exec time reduction: 82.4%

Container memory reduction: 15.6%

Latency overhead: < 1ms (835.7\mu s)
Closing the loop

- First serverless characterization from a provider’s point of view

- A dynamic policy to manage serverless workloads more efficiently
  (First elements now running in production.)

- Azure Functions traces available to download: