Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider

Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini

July 15, 2020
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?

• 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

Cold Starts

Keeping functions in memory indefinitely.

Wasted Memory

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.
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.
Apps are highly heterogeneous.
What about memory?

If we wanted to keep all apps warm...

Fraction of Least Invoked Apps

Cumulative Fraction of Total 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 on 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

- **Cold Starts**: Keeping functions in memory indefinitely.
- **Wasted Memory**: Removing function instance from memory after invocation.

Graphs showing cumulative fraction of total memory allocated and physical memory used.

- CDF (Cumulative Distribution Function) of time:
  - Minimum
  - Average
  - Maximum
  - LogNormal Fit

Time: 1ms, 100ms, 1s, 10s, 1m, 10m, 1h

Fraction of Least Invoked Apps:
- Allocated Memory
- Physical Memory
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/
Fixed Keep-Alive Policy

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

8 mins  8 mins
Cold Start  10-minute Fixed Keep-alive

11 mins  11 mins
Warm Start
Fixed Keep-Alive Is Wasteful

Function image kept in memory but not used.

10-minute Fixed Keep-alive

Cold Start

Warm Start
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

8 mins

Frequency

Time

Cold Start

Warm Start

10-minute
Fixed
Keep-alive
A Histogram Policy To Learn Idle Times
A Histogram Policy To Learn Idle Times

- Pre-warm
- Keep-alive

- 5th percentile
- 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.

Time Series Forecast
The Hybrid Histogram Policy

ARIMA: Autoregressive Integrated Moving Average
More Optimal Pareto Frontier

![Graph showing normalized wasted memory time vs. 3rd quartile app cold start for different time intervals, indicating approximate 1.5X and 2.5X improvements for certain conditions.]

- Fixed
  - 120-min
  - 90-min
  - 60-min
  - 45-min
  - 30-min
  - 20-min
  - 10-min
  - 5-min

- Hybrid
  - 4-hr
  - 3-hr
  - 2-hr
  - 1-hr
• 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 (835.7μs)
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: