Serverless Computing Overview

Yiying Zhang
• Quiz 1 and project proposal graded
• Discussion lead sign up
• Topics for the rest of the quarter?
Outline

• Overview and major offerings
• Example applications
• Limitations and discussion
• Real-world serverless workload characterization

Acknowledgement: Some of the slides are from Ali Ghodsi and Ion Stoica’s Berkeley lecture notes in 2018 and Tyler Harter’s HotCloud’16 OpenLambda talk
What is Serverless Computing?

• Computing without servers?
• Running applications without the need to manage servers?
• Running functions instead of containers/VMs?
• Infinite scaling?

• The truth: no clear, agreed definition, i.e., no one really knows
One Perspective: How Cloud and Virtualization Evolved
Classic Web Stack

- Application
- Server
- OS
- Hardware

RPCs

weak virtualization
1st Generation: Virtual Machines
2nd Generation: Containers
3rd Generation: Serverless Computing
Evolution of serverless

- Bare Metal
- Virtual machines
- Containers
- Functions

Decreasing concern (and control) over stack implementation

Increasing focus on business logic
Serverless means ...

- No server or container management
- Flexible scaling
- High availability
- No idle capacity

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
What is the essence of “Serverless Computing”? Or, what do people really like about it?

• Management-free
  • No need to handle creation, failure, replication, etc.

• Autoscaling
  • Spin up/down functions quickly based on load

• Only pay for what you use

“I didn’t have to worry about building a platform and the concept of a server, capacity planning and all that “yak shaving” was far from my mind... However, these changes are not really the exciting parts. The killer, the gotcha is the billing by the function...

This is like manna from heaven for someone trying to build a business. Certainly I have the investment in developing the code but with application being a variable operational cost then I can make a money printing machine which grows with users...”

source: https://hackernoon.com/why-the-fuss-about-serverless-4370b1596da0
What is Today’s Serverless Computing Like?

- Largely offered as Function as a Service (FaaS)
  - Cloud users write functions (single-entry computing units) and ship them
  - Cloud provider runs and manages them
- Still runs on servers
- Have attractive features but also many limitation
First serverless app: BigQuery

Fully managed Data Warehouse
- “Arbitrarily” large data and queries
- Pay per bytes being processed
- No concept of cluster

Other similar systems
- AWS Athena
- Snowflake
- …
AWS Lambda

• An event-driven, serverless computing FaaS platform introduced in 2014
• Functions can be written in Node.js, Python, Java, Go, Ruby, C#, PowerShell
• Each function allowed to take 128MB - 10GB memory and up to 15min
• Max 50-3000 concurrent functions
• Connected with many other AWS services
Lambda Function Triggering and Billing Model

- Run user handlers in response to events
  - web requests (RPC handlers)
  - database updates (triggers)
  - scheduled events (cron jobs)
- Pay per function invocation
  - No charge when no functions run (no triggering event)
  - Billed by duration of function, configured memory size, and # of functions
    - charge \textit{actual\_time} * \textit{memory\_cap}
Serverless Applications

Event source

Lambda function

Services (anything)

Changes in data state

Requests to endpoints

Changes in resource state

Node.js
Python
Java
C# (.NET Core & Core 2.0)
Go
Ruby
Powershell
BYR – Bring your own Runtime

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Internal Execution Model

• Developers upload function code to a handler store (and associate it with a URL)

• Events trigger functions through RPC (to the URL)

• Load balancers handle RPC requests by starting handlers on workers
  • Calls to the same function are typically sent to the same worker(s)

• Handlers sandboxed in containers
  • AWS Lambda reuses the same container to execute multiple handlers when possible
Execution Model

load balancers

user → RPC → Load Balancer → ... → workers

handler store

H1, H2 → ... → H2 sandbox

Python → Server

Python → Server
Azure Functions

• Debuted in 2016

• Support Python, C#, Java, JavaScript and PowerShell

• Three plans:
  • Consumption, Premium, and Dedicated
  • Dedicated: functions run on dedicated VMs and scaled manually (unlimited duration)
  • Max duration for consumption and premium: 10min and 60min
  • Max memory: 1.5GB - 14GB (depending on plans)
Google Cloud Functions

- Support Node.js, Python, and Go
- Memory size 128MB - 2GB
- Max function duration 9min
- Max number of functions per project: 1000
- Bill CPU and memory separately
Open-Source Projects

• OpenWhisk
  • Originally developed by IBM and the core technology behind IBM Cloud Functions

• Cloudburst
  • Stateful serverless framework built on top of the Anna key-value store
Limitations of Today’s Serverless Offerings

- Difficult and slow to manage states
  - Have to use (slow) cloud storage or other VMs!
- No easy or fast way to communicate across functions
  - Have to go through cloud storage or other services
- Functions can only use limited resources, and resources have “fixed ratios”
- Still need manual effort to “split” applications and size functions
- No control over function placement or locality
  - e.g., starting functions on “cold” machines can be slow
- Billing model does not fit all needs
Characterizing Serverless Systems

Mohammad Shahrad

mshahrad@ece.ubc.ca
@MShahrad

The University of British Columbia

January 2022

UBC Cloud Infrastructure Research for Reliability, Usability, and Scalability
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

Microsoft Azure and Microsoft Research

• First characterization of production serverless workloads
  • Many new insights
  • Released production traces of Azure Functions:
    • https://github.com/Azure/AzurePublicDataset

• A new adaptive serverless management scheme
  • New angle: going after eliminating cold starts instead of reducing cold start overhead
  • Improves the user experience, reduces the underlying resource usage
  • Deployed in production
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.
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.
Who should you design/optimize the system for?

18% of applications that constitute 99.6% of invocations

- Can lead to immediate cost reductions for the provider.
- Keeps the big users happy and prevents them from migrating to competitors.

Or

82% of applications that constitute 0.4% of invocations

- Can incentivize more users to shift to serverless. Potentially higher long-term profit.
- Seems reasonable when user valuation of service/QoS is unknown.

Mixing both?
Average Invocations Do Not Tell the Entire Story
It is all about trade-offs!

**Performant Serverless**
- Short Execution
- Low Cold Start
- Overhead
- Fewer Cold Starts
- Minimum Wait Time

**Efficient Serverless**
- High Co-tenancy
- Minimal Resource Wastage
- High Cluster Utilization

**Conflicting Goals**
Cold Starts and Resource Wastage
Cold Starts and Resource Wastage

- Cold Starts
- Wasted Memory

Keeping functions in memory indefinitely.

Removing function instance from memory after invocation.
Removing function instance from memory after invocation.

\[ O(\text{execution time}) = O(\text{cold start overhead}) \]

Cold starts cannot be neglected.

Keeping functions in memory indefinitely.

Memory usage not negligible.

Can’t be kept in memory forever.
What do serverless providers do?

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.
A Histogram Policy To Learn Idle Times

Minute-long bins

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

Out of Bound (OOB)

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

Time Series Forecast
Decision Tree for the Hybrid Histogram Policy

ARIMA: Autoregressive Integrated Moving Average
More Optimal Pareto Frontier
Simulation

4-Hour Hybrid Histogram

Experimental

Container memory reduction: 15.6%
Average exec time reduction: 32.5%
99th–percentile exec time reduction: 82.4%
Latency overhead: < 1ms (~800 µs)

Production possible thanks to the dedication of Íñigo Goiri, Gohar Chaudhry, and the Azure Functions team.