Serverless Computing: Design, Implementation, and Performance

Garrett McGrath and Paul R. Brenner

UNIVERSITY OF NOTRE DAME

Introduction

UNIVERSITY OF NOTRE DAME

Serverless Computing

- Explosion in popularity over the past 3 years
- Offerings from all leading cloud providers
- However, few performance comparisons of these platforms exist

This Presentation

- Explore serverless design through a new prototype platform
  - Focused on performant execution of functions
    - Serverless paradigms create long function chains, real-time pipelines; latency matters
- Develop cross-platform performance tests
- Measure performance of existing commercial platforms and prototype

Prototype Overview

UNIVERSITY OF NOTRE DAME

Serverless Prototype Platform

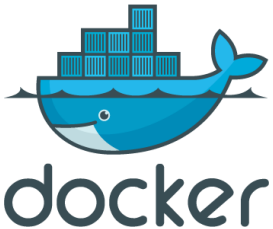
- Implemented in C#/.NET
- Utilizes Windows containers as function execution environments
- Docker provides container management functionality
- Deployed on a variety of services in Microsoft Azure
- Available: https://github.com/mgarrettm/serverless-prototype

Prototype Purpose

- Research prototype on which to explore serverless platform design
- Baseline to compare against existing platforms

Prototype Goals

- Efficient execution of functions
- Simplicity of implementation



Prototype Design

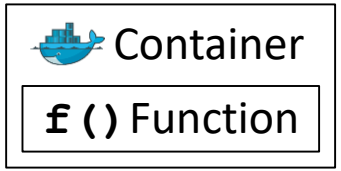
UNIVERSITY OF NOTRE DAME

`f()` Function

Functions!

- Functions are the unit of deployment and scaling
- Simple goals: support basic CRUD and synchronous execution of functions
- How to manage functions?
- Where to execute functions?
- How to discover those locations?

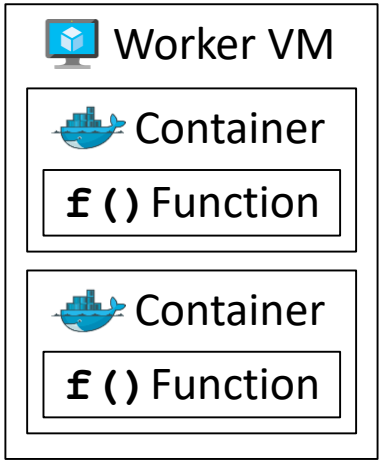
Prototype Design



Function Containers

- Function resides within container for security and resource isolation
- Containers are reused to offset unwieldy start-up times
- Windows Server Containers chosen as container technology
- Windows “Nano” Server image (801 MB) used
  - Alpine Linux is 18 MB
- Node.js v6.9.5 runtime supported

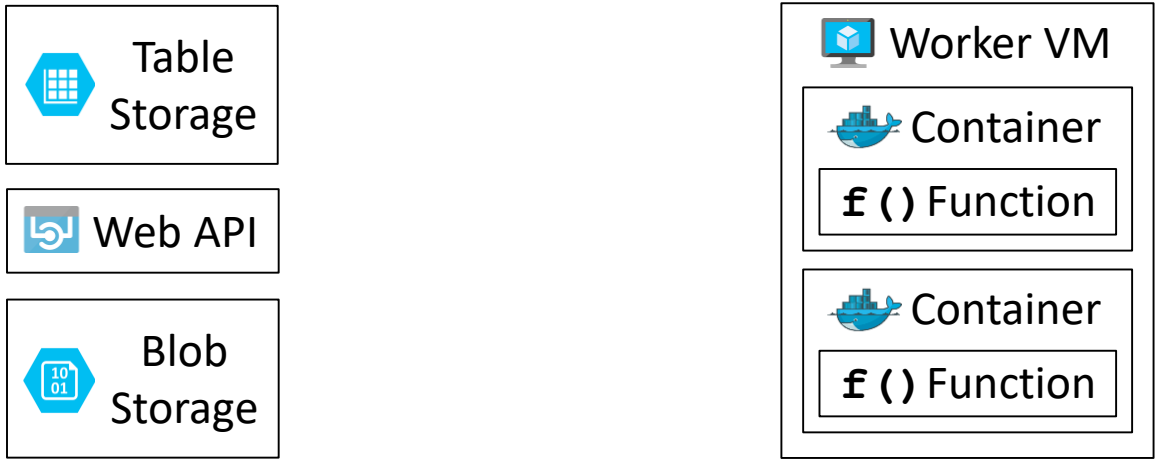
Prototype Design



Worker VMs

- Handles container lifecycle and accepts function execution requests
- Containers expire after 15 minutes without execution
- Many workers; many containers per worker
- Important choice between existing container management systems and custom solution

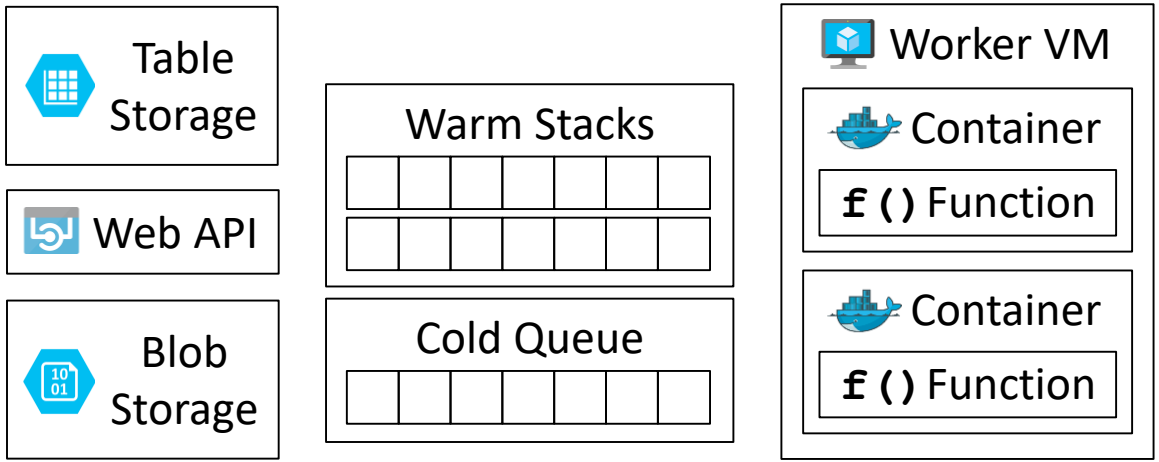
Prototype Design



Web Service

- External-facing component of platform
- Web API provides function CRUD and execution
- Function metadata stored in Azure Table Storage
- Function code artifacts stored in Azure Blob Storage and linked in metadata

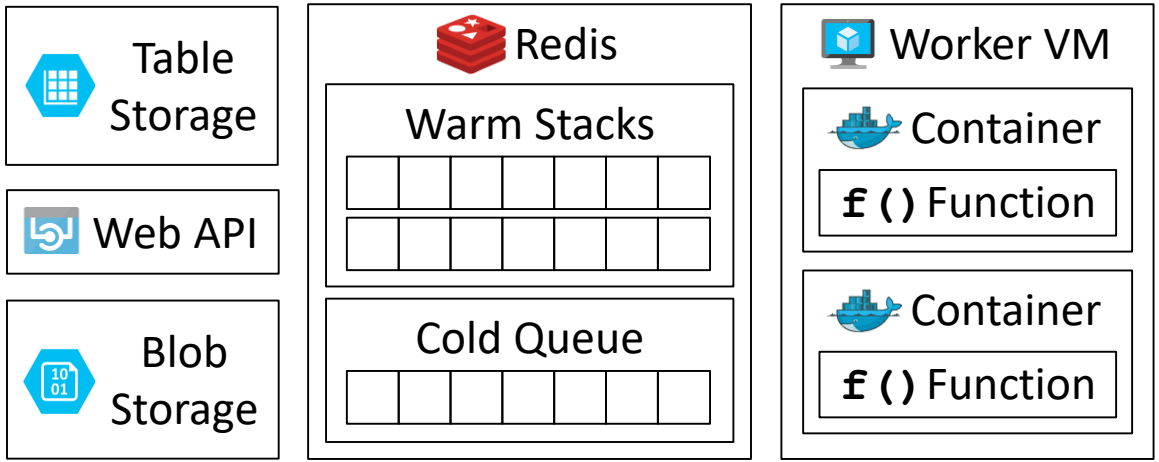
Prototype Design



Container Discovery

- Workers reserve memory space as allocations and store their locations in messaging layer
- Unassigned container locations reside in cold queue
- LIFO warm stack for each function to store assigned containers
- Workers are source of truth for container state (expiration, inconsistent data)

Prototype Design



Redis

- Can afford to compromise consistency and durability!
- Availability and load balancing may be problematic
- Consistent hashing service is viable alternative
- Azure Storage Queues do not provide LIFO functionality

Performance Framework

UNIVERSITY OF NOTRE DAME

Testing Framework

- Developed a basic cross-platform testing framework in Node.js using the Serverless Framework
  - Available: https://github.com/mgarrettm/serverless-performance
- Created a Serverless Framework provider plugin to deploy functions to the prototype
  - Available: https://github.com/mgarrettm/serverless-prototype-plugin
- Deploys a function that immediately returns with a unique id of its instance

Testing Methodology

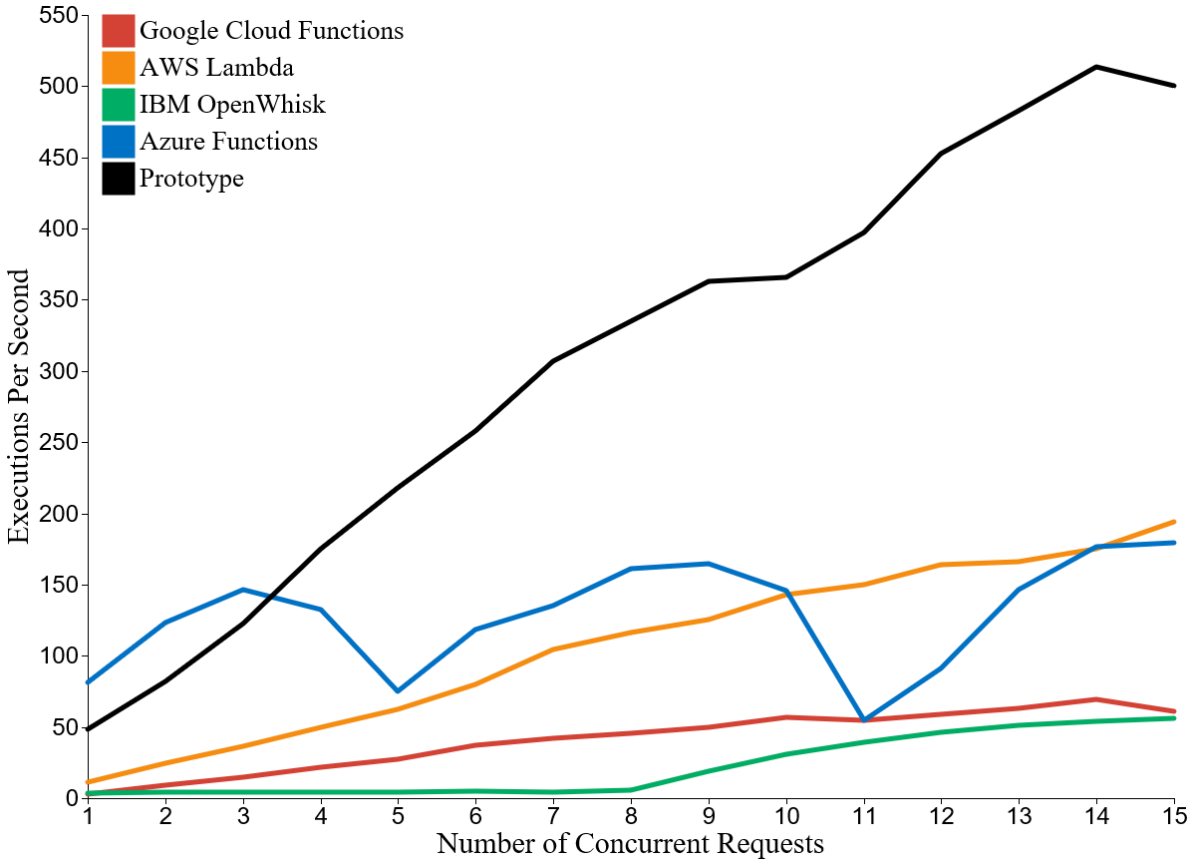
- Tests conducted from virtual machines in same datacenter as functions
  - Exception: IBM OpenWhisk tested from US-SouthCentral datacenter in Microsoft Azure (<10ms latency)
- Tests measure response time using test machine clock
  - Network latency unaccounted for (test machines placed as close as possible to function)
- Tests run in March 2017
  - Platforms change frequently

Performance Results

UNIVERSITY OF NOTRE DAME

Concurrency Test

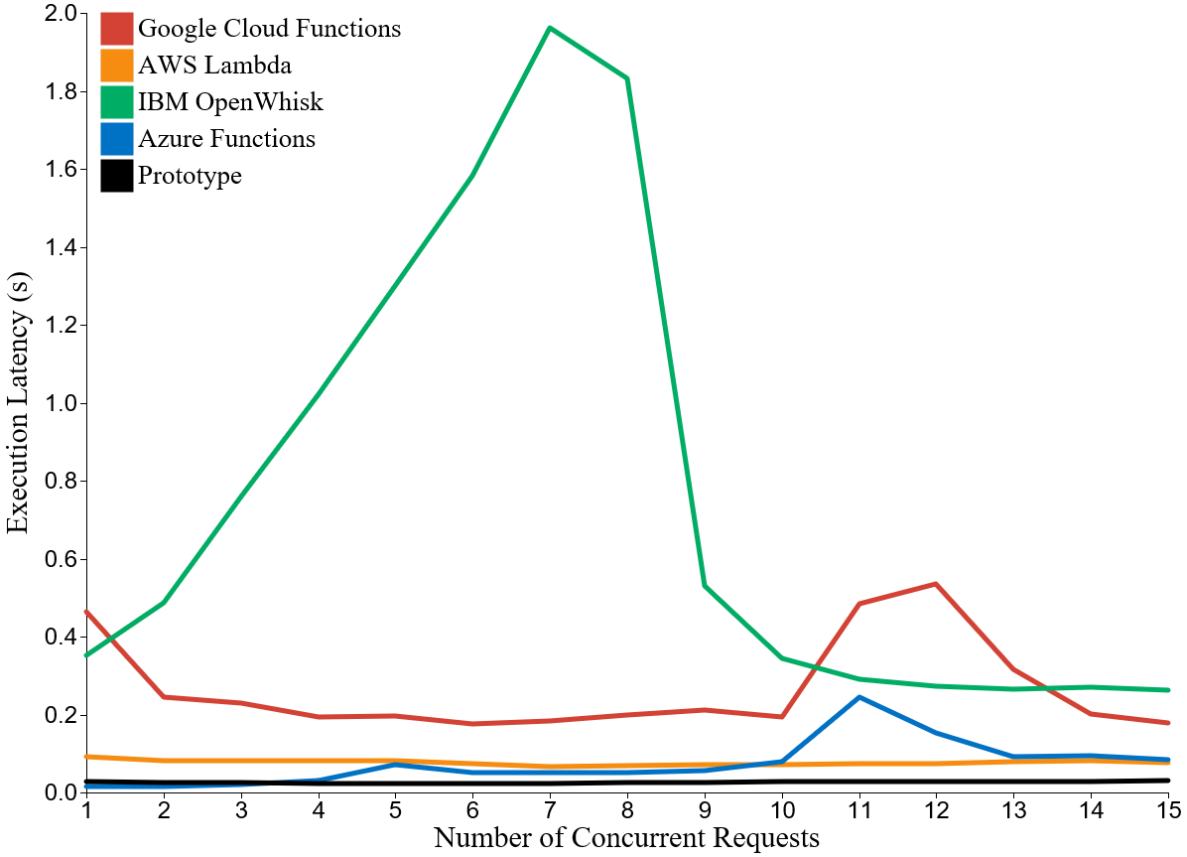
- Designed to measure throughput of serverless platforms
- Reissues each request immediately after receiving the response from the previous call
- Increase concurrent requests from 1 to 15



Performance Results

Concurrency Test

- Designed to measure throughput of serverless platforms
- Reissues each request immediately after receiving the response from the previous call
- Increase concurrent requests from 1 to 15

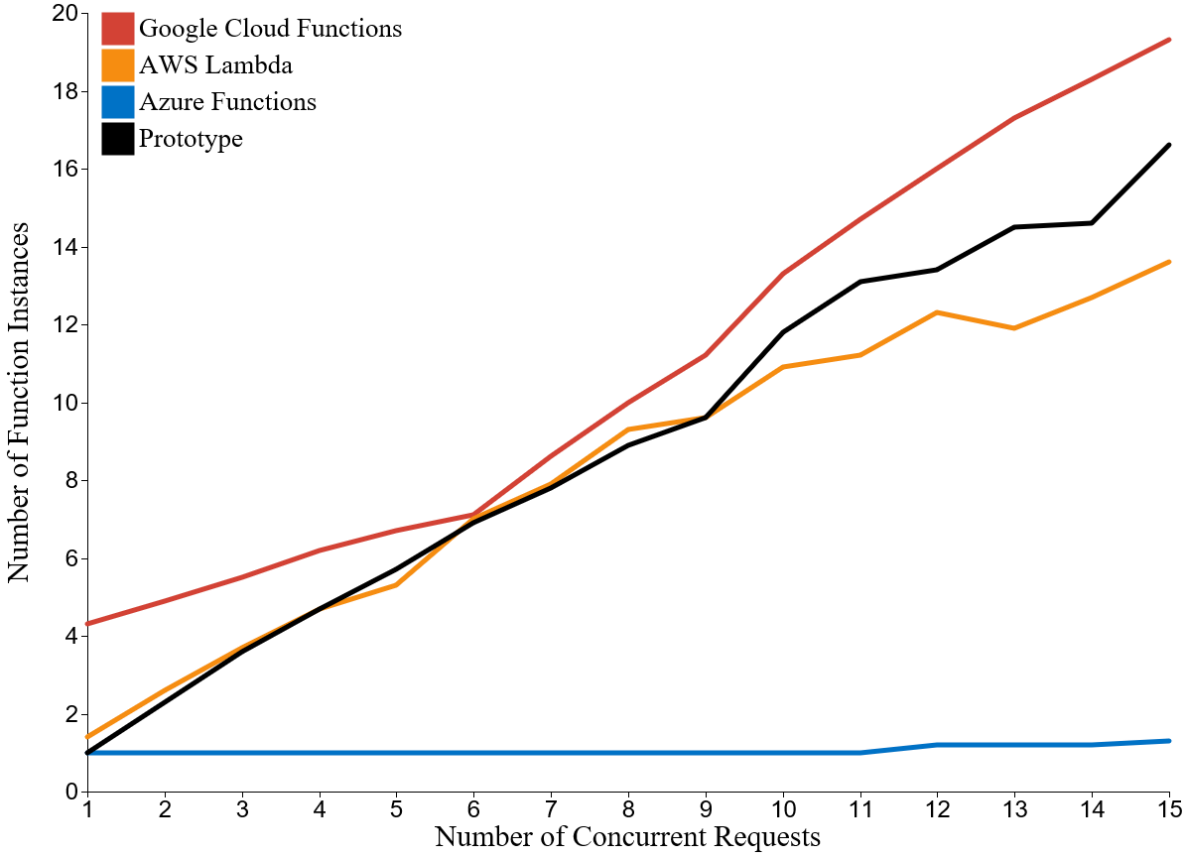


Performance Results

UNIVERSITY OF NOTRE DAME

Concurrency Test

- Designed to measure throughput of serverless platforms
- Reissues each request immediately after receiving the response from the previous call
- Increase concurrent requests from 1 to 15

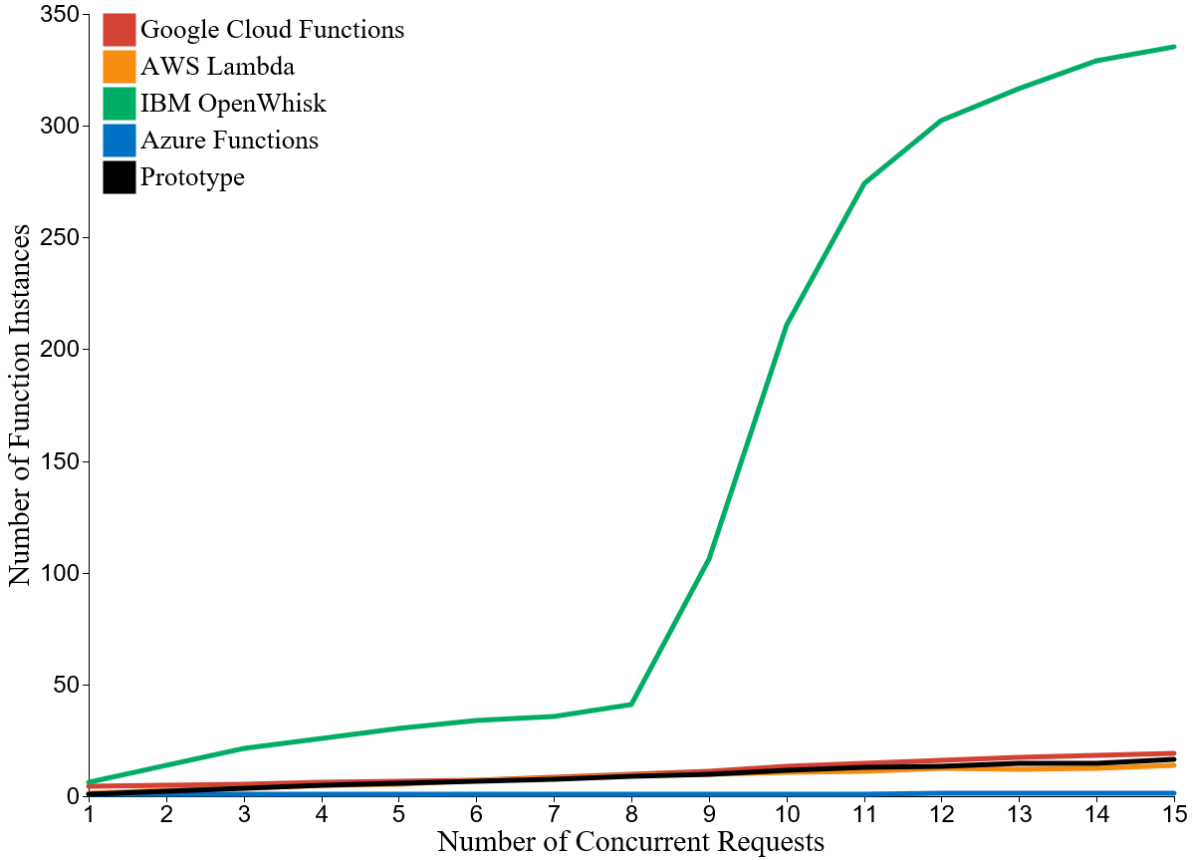


Performance Results

UNIVERSITY OF NOTRE DAME

Concurrency Test

- Designed to measure throughput of serverless platforms
- Reissues each request immediately after receiving the response from the previous call
- Increase concurrent requests from 1 to 15

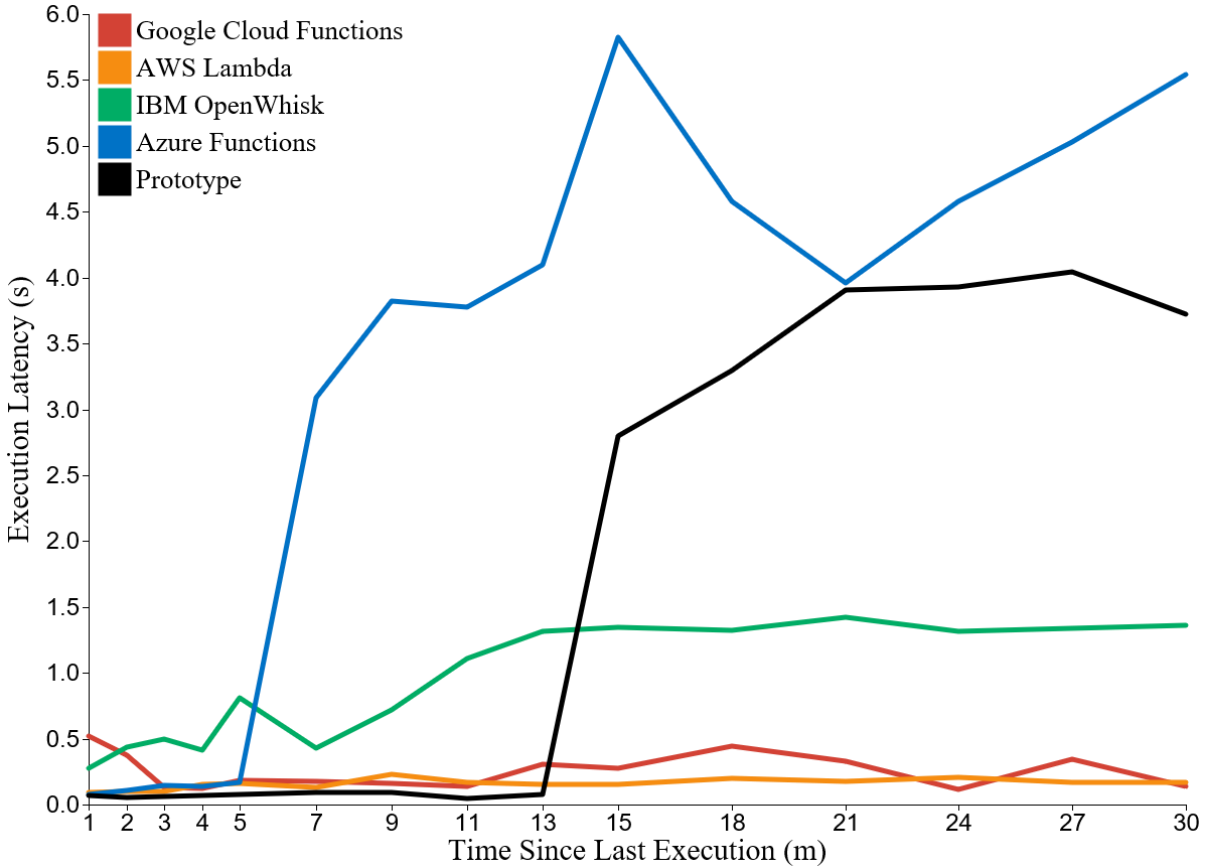


Performance Results

UNIVERSITY OF NOTRE DAME

Backoff Test

- Designed to measure latency of serverless platforms and show container expiration thresholds
- Increase time between consecutive requests from 1 to 30 minutes



Future Work

UNIVERSITY OF NOTRE DAME

Serverless Prototype

- Asynchronous executions
  - More difficult because executions must be guaranteed once success is returned to client
  - Durable tracking of active executions alongside existing execution pipeline
- More advanced container support
  - Windows Server Containers are limited in their operations (pausing/resizing)
  - Support for Linux Containers opens up opportunities to improve cold start performance
  - Docker's path towards modularization with Moby could be useful in tailoring Docker for FaaS

Performance Framework

- Asynchronous execution performance testing
  - Accurate timing is more difficult
  - Services like X-Ray in AWS help, but are not cross-platform

Questions?