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

*Authors: 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*

JINGRU ZHAO， ENBEI LIU

## OutLine

- For FAAS, the providers seek to achieve ***high function performance at the lowest possible resource cost***. There are 3 main aspects to achieve this.
- 1.More warm starts than cold starts
- 2.spend less memory for functions
- 3.Functions may have widely varying resource needs and invocation frequencies from multiple triggers. These characteristics severely complicate any attempts to predict invocations for reducing resource usage.
- This goal requires a deep understanding of the characteristics of the FaaS workload.

## OutLine

***In summary, main contributions in this article are:***

- • A detailed characterization of the entire production FaaS workload at a large cloud provider;
- • A new policy for reducing the number of cold start function executions at a low resource provisioning cost;
- • Extensive simulation and experimental results based on real traces showing the benefits of the policy;
- • An overview of our implementation in Azure Functions;
- • A large sanitized dataset containing production FaaS traces.

## Introduction

- **What's the problem**:
- Providing high function performance at low cost requires a deep understanding of the characteristics of the FaaS workload.
- But there has been no public information on the characteristics of production workloads.
- The provider needs to deeply understand the characteristics of the FaaS workload.

## Introduction

- **Why is it a problem:**
- Prior work focused on
- (1) running benchmark functions to

  assess performance and/or reverse-engineer how providers manage resources;
- (2) implementing prototype systems to run benchmark functions.

  We needed  a comprehensive characterization of the users' real FaaS

  workloads on a production platform from the provider's perspective.

## Introduction

- **Why are we interested in this problem**:
Providers seek to achieve high function performance at the lowest possible resource cost for FAAS. There are
three main aspects to how fast functions can execute and
the resources they consume.
  1. Functions warm start
  2. Memory storage
  3. Multiple triggers for functions

## Background / Related Work

**What have others done related to the problem?**

1.Works that characterize FaaS platforms and applications

2.Works that propose and optimize FaaS serving systems.

A few studies characterized the main commercial FaaS providers, but only from the perspective of external users.

For optimizing each cold start

## Background / Related Work

**What have AWS/AZURE done？**

AWS and Azure use a fixed "keep-alive" policy that retains the resources in memory for 10 and 20 minutes after a function execution.

**Advantage** of this policy: simple and practical

**Disadvantage** of this policy:  it disregards the functions' actual invocation frequency and patterns, and thus behaves poorly and wastes resources.

## Background / Related Work

**What have others done related to the problem？**

For optimizing each cold start

1.Mohan find that

pre-allocating virtual network interfaces that are later bound

to new function containers can significantly reduce cold start

times.

## Background / Related Work

**What have others done related to the problem？**

2.SOCK proposes to optimize the loading of Python functions in OpenLambda by smart caching of sets of libraries, and by using lightweight isolation mechanisms for functions.

3.SAND uses application-level sandboxing to prevent the cold start latency for subsequent function invocations within an application.

## Background / Related Work

Some people propose a policy for deciding on function multi-tenancy, based on a predictive model of resource demands of each function.

1.EMARS  proposes using predictive modeling for allocation of memory to serverless functions.

2. Kesidis proposes to use the prediction of the resource demands of functions to enable the provider to overbook functions on containers.

**In this article** the author track invocation patterns and use this knowledge to reduce cold starts and memory waste.

## new technology

In this article, the authors propose a different policy from the original policy being used by Azure/AWS. This policy has **2 main features**

(1) It uses a different keep-alive value for each user’s workload, according to its actual invocation frequency and pattern;

(2) It enables the provider in many cases to pre-warm a function execution just before its invocation happens (making it a warm start).

## FaaS Workloads

The author characterize the FaaS workloads seen by Azure Functions, focusing on characteristics that are intrinsic to the applications and functions (e.g., their arrival pattern). They collected data on all function invocations across Azure's entire infrastructure between July 15th and July 28th, 2019.

It contain four main parts:

(1)Functions, Applications, and Triggers

(2)Invocation Patterns

(3)Function Execution Times

(4)Memory Usage

## Functions, Applications, and Triggers

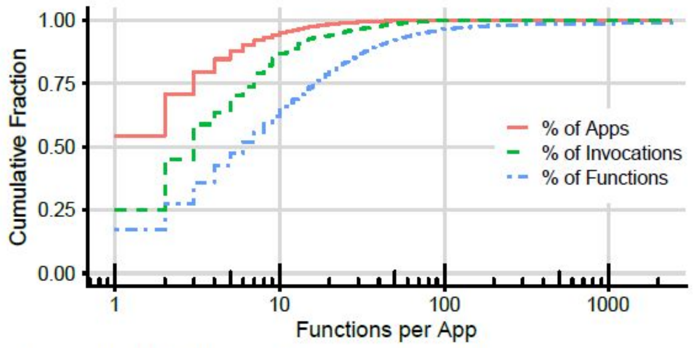
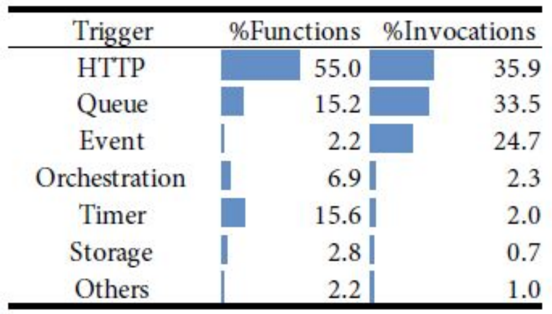
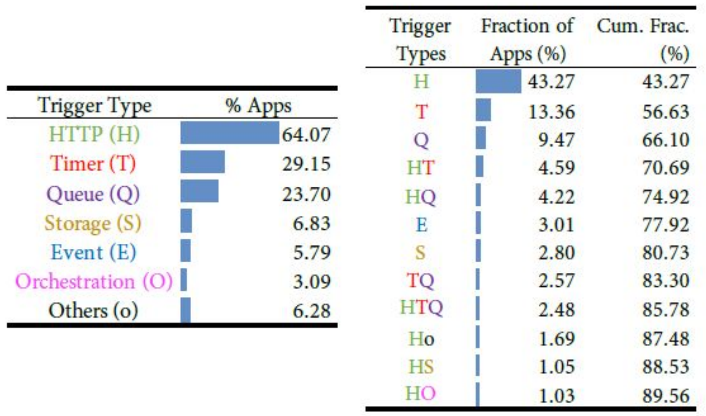


Figure 1: Distribution of the number of functions per app.



| Trigger | %Functions | %Invocations |
|---|---|---|
| HTTP | 55.0 | 35.9 |
| Queue | 15.2 | 33.5 |
| Event | 2.2 | 24.7 |
| Orchestration | 6.9 | 2.3 |
| Timer | 15.6 | 2.0 |
| Storage | 2.8 | 0.7 |
| Others | 2.2 | 1.0 |

Figure 2: Functions and invocations per trigger type.



| Trigger Type | % Apps |
|---|---|
| HTTP (H) | 64.07 |
| Timer (T) | 29.15 |
| Queue (Q) | 23.70 |
| Storage (S) | 6.83 |
| Event (E) | 5.79 |
| Orchestration (O) | 3.09 |
| Others (o) | 6.28 |

(a) Apps with ≥ 1 of each trigger.

| Trigger Types | Fraction of Apps (%) | Cum. Frac. (%) |
|---|---|---|
| H | 43.27 | 43.27 |
| T | 13.36 | 56.63 |
| Q | 9.47 | 66.10 |
| HT | 4.59 | 70.69 |
| HQ | 4.22 | 74.92 |
| E | 3.01 | 77.92 |
| S | 2.80 | 80.73 |
| TQ | 2.57 | 83.30 |
| HTQ | 2.48 | 85.78 |
| Ho | 1.69 | 87.48 |
| HS | 1.05 | 88.53 |
| HO | 1.03 | 89.56 |

(b) Popular trigger combinations.

Figure 3: Trigger types in applications.

Figure 1 shows the CDF of the number of functions per application (top curve).

Figure 2 shows the fraction of all functions, and all invocations, per type of trigger.

Figure 3 shows how applications combine functions with different trigger types.

## Invocation Patterns

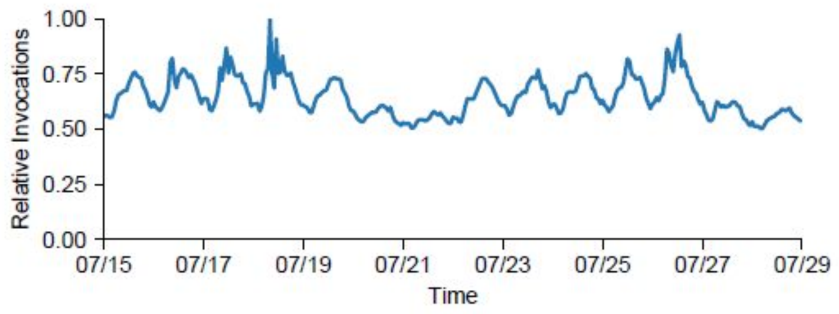
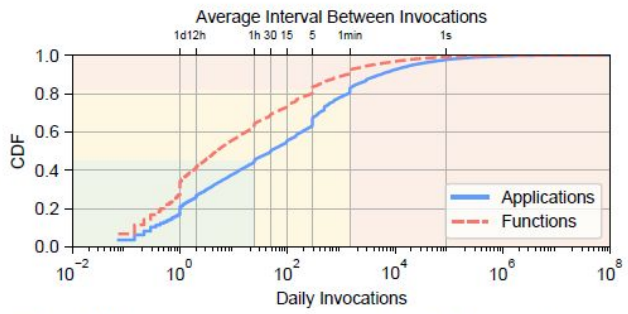
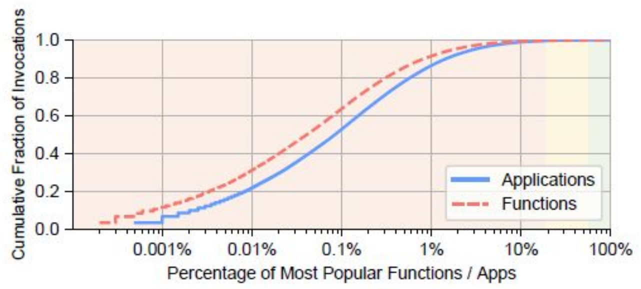


Figure 4: Invocations per hour, normalized to the peak.



(a) CDF of daily invocations per function and application, and the corresponding *average* interval between invocations. Shaded regions show applications invoked on average at most once per hour (green, 45% of apps) and at most once per minute (yellow, 81% of apps).



(b) Fraction of total function invocations by the fraction of the most popular functions and applications. Same colors as in Figure 5(a).

Figure 5: Invocations per application and per function for a representative sample of the dataset.

Figure 4 shows the volume of invocations per hour, across the entire platform, relative to the peak hourly load on July 18th.

Figure 5(a) shows the CDF of the average number of invocations per day, for a representative sample of both functions and applications.

Figure 5(b) shows the other side of the workload skewness, by looking at the cumulative fraction of invocations due to the most popular functions and applications in the sample.

## Invocation Patterns

The invocation rates provide information on the average inter-arrival time (IAT) of function and application invocations, but not on the distribution of these IATs. To gain insight into the IAT distributions of applications, we look at the coefficient of variation (CV) of each application.

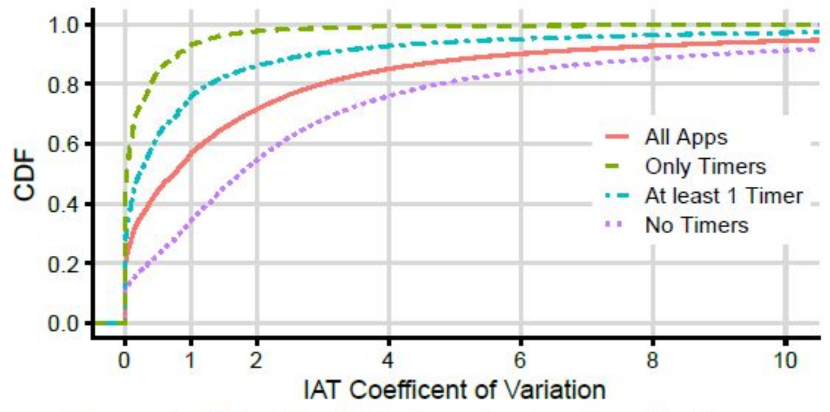


Figure 6: CV of the IATs for subsets of applications.

## Function Execution Times

Another aspect of the workload is the function execution time, i.e. the time functions take to execute after they are ready to run. Figure 7 shows the distribution of average, minimum, and maximum execution times of all function executions on July 15th, 2019.

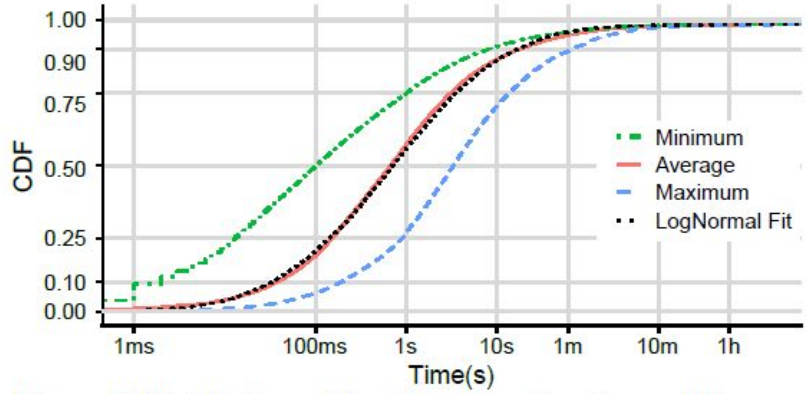


Figure 7: Distribution of function execution times. Min, avg, and max are separate CDFs, and use independent sorting.

## Memory Usage

Application is the unit of memory allocation in the platform. Figure 8 shows the memory demand distribution, across all applications running on July 15th, 2019.

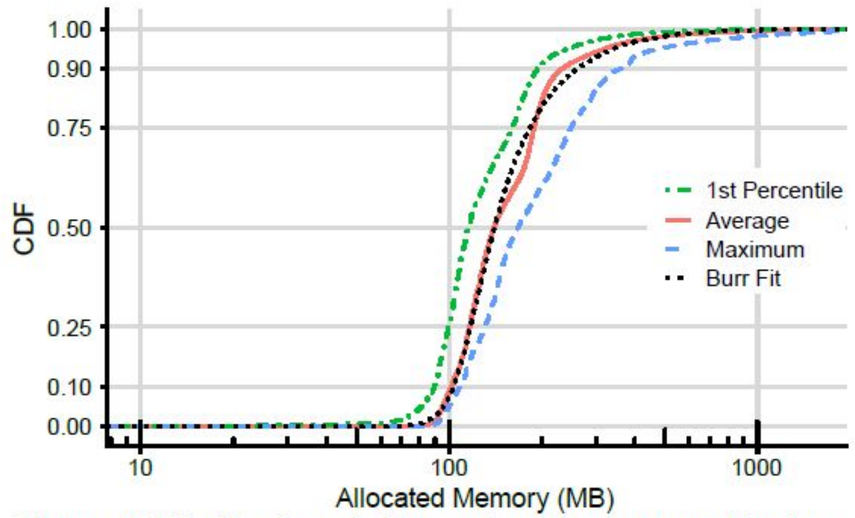


Figure 8: Distribution of allocated memory per application.

## Conclusion

From the observations, we now have three main conclusions:

(1)Functions vary a lot when it comes to invocation frequency, making it necessary to customize resource management policies for different functions.

(2)Since the vast majority of functions have execution times on the same order of magnitude of their cold start times, it's critical to reduce the number of cold starts.

(3)40% of functions have a CV of their IATs higher than 1, which makes the prediction challenging. Especially when invocation is infrequent, the prediction may not be an effective solution

## Evaluation

We focus on section 3 of this paper. The author didn’t give an evaluation about this section.

## Critique: Strengths

1.This paper is from the perspective of cloud providers.

2.There has been little to no public information on these characteristics of the FaaS workload. This paper characterize the FaaS workloads seen by Azure Functions from four aspects.

3.This paper use a lot of figures to make the data more specific.

## Critique: Weaknesses

Given the extreme scale of Azure Functions, the invocation counts are binned in 1-minute intervals, i.e. their dataset does not allow the precise reconstruction of interarrival times that are smaller than one minute.

**For this paper, this granularity is sufficient.**

## Question?