# Simulation and Modelling
# Assessed Coursework
# **Serverless Cold Starts**

Issue date: 24th October 2022
Working: Pairs or individuals
Submission deadline: 18th November 2022
Page limit: TWO sheets of A4 plus separate Appendix

## 1 Background

In this coursework, you will learn how to apply simulation and modelling to Function-as-a-Service (FaaS) platforms, which allow developers to deploy their codes as individual functions in the cloud without having to deal with the underlying infrastructure management.

The availability of open-source FaaS solutions, such as OpenFaaS, has made it possible to deploy FaaS platforms inside the private clouds of many companies. FaaS platforms are often best used with *event-driven* applications, which serve requests generated in association with events (e.g., if a water pipe leak is detected by a smart sensor, a call to a serverless function could enact a rule to alert the utility company, the municipality, the homeowner, etc). The logic that serves requests is coded as a *serverless function*, usually packaged within a container. Hence, throughout, we will use the terms “function” and “container” interchangeably.

When a new request for a serverless function arrives, if the container is not present in memory the request will be blocked until the container is loaded, introducing an overhead on the request response time called the *cold start* time. Instead, if the container is present in memory and can accept the incoming request, the cold start time will *not* be incurred. Note that the incoming request may be rejected, even though the contained is present in memory (see below).

Deallocating containers reduces memory usage, but cold start times may adversely impact performance. This coursework aims to study the performance impact of cold starts using simulation and to construct and solve a simple CTMC for the same system for a very simple parameterisation.

### 1.1 Reference Model

We assume the FaaS platform to be hosted on a single server machine. Formally, the system under consideration is described by the following input parameters:

- $\lambda_f$ : the arrival rate of requests to function $f$ (requests / second). Assume that inter-arrival times for each function are exponentially distributed; hence, the arrival process to each function is a Poisson process.
- $\alpha = 0.5$: cold-start overhead parameter (in seconds$^{-1}$). As the cold start time depends on various factors (machine load, image size, network bandwidth if the container is retrieved from remote, ...) you should assume that a request for a function $f$, when it incurs a cold start, suffers an exponentially distributed overhead with mean $1/\alpha$ before being loaded in memory. Thus, for $\alpha = 0.5$ the average cold start overhead is 2 seconds.

- $m_f$: the memory occupation of a loaded function $f$. Assume this to be identical for all functions and equal to 100MB.

## 1.2 Materials

You are supplied with this coursework a real-world serverless dataset tracing $F = 10861$ functions. For each function $f$, $1 \leq f \leq F$, the dataset includes:

- the total number of requests that arrived to each function in the observation period of $T = 30$ days.
- the *mean* service times $S_f$ of the functions (in milliseconds, rounded up).

The total number of request invocations allows you to estimate the arrival rate $\lambda_f$ for each function. For the purposes of the model you should assume that the service times for function $f$ are exponentially distributed with rate parameter $1/S_f$ (i.e. mean $S_f$).

## 1.3 Question 1: Simulation

### 1.3.1 Assumptions

You should also make these assumptions:

**A0.** At time $t = 0$, no function is in memory.

**A1.** If a request arrives for a function $f$ and there is not enough spare memory to load it, the function $g$ that is already loaded in memory and *has been idle the longest* is instantaneously deallocated and its memory space immediately allocated to $f$. If no function is idle, the incoming request to $f$ is lost.

**A2.** If a request for function $f$ is received while that function is serving another request, then the incoming request is lost.

**A3.** During the cold start period for function $f$, incoming requests to $f$ other than the one that triggered the cold start are lost. The request that triggered the cold start can begin service only after the cold start period ends.

**A4.** A function $f$ cannot be deallocated from memory during its cold start. The earliest time at which it can be deallocated is right after it serves the first job, if $f$ is idle then.

**A5.** The CPU capacity is overprovisioned and contention is negligible, so that any request to function $f$ either (i) receives its intended service time, (ii) receives the service time plus the cold start time, or (iii) it is lost, based on the rules given above.

**A6.** The total memory available is initially 4 gigabytes, i.e. there is capacity for $M = 40$ functions in memory at any time. To simplify the simulation you are advised to start the FaaS with the first $M$ functions in memory in the idle state and the remainder in the unloaded (not in memory) state – this means that there will be precisely $M$ functions in memory at all times, so you will not have to model unused memory that has yet to be loaded/initialised.

### 1.3.2 What to do

**Q1.a)** Implement a simulator of a FaaS system hosted on single machine and compute unbiased point and interval estimates for:

- $C_{ratio}$: the cold start ratio, i.e. the probability that a request incurs a cold start.
- $L_{rate}$: the loss rate, i.e., the rate at which requests are lost.

**Q1.b)** By changing the memory capacity of the FaaS system, $M$, estimate the minimum value of $M$ that achieves a cold start ratio no greater than 5%.

**Q1.c)** In addition to these estimates, briefly explain how you have chosen to model the Poisson processes for the incoming requests, including a description of how you advance time and how you choose the next function to be requested.

**Q1.d)** A small amount of credit ($\sim$ 10% of the marks for the simulation) is reserved for any additional experiments you might wish to perform, e.g. exploring changes to the exponential distributions assumed above or modelling additional functionality in the FaaS system[1]

Please attach your code as an Appendix.

## 1.4 Question 2: Analytical Modelling

You are now going to model very simple FaaS system as a CTMC. Doing this for any realistic scenario will involve an enormous number of states and state transitions, so in this exercise you should assume a "tiny" configuration where $F = 2$ and $M = 1$.

**Q2.a)** By considering the possible states that each function can be in, define the state space of the FaaS system.

**Q2.b)** Write down the infinitesimal generator for a continuous time Markov chain (CTMC) that models the system *and* draw its transition diagram, identifying clearly the transition rates between states.

**Q2.c)** Give general expressions for computing the cold start ratio $C_{ratio}$ and the loss rate $L$ in terms of the equilibrium state probabilities of this CTMC.

**Q2.d)** Use the data in the trace for the first two functions (the first two data rows in the trace excluding the header) to compute numerically the values of $C_{ratio}$ and $L_{rate}$ using the CTMC. Compare these values with the ones obtained using a simulation with the same parameters. Discuss your findings.

# 2 Assessment

We anticipate a relative weighting of 60% and 40% for the two questions. For the simulation you will be primarily assessed on your methods, results and insights gained from additional experiments, rather than on the quality of the code you write. However, if the code is unorderly to the point that is hard to follow you may lose some credit.

[1] For example, in a real FaaS system there might be a minimum idle time before which a function cannot be replaced.

## 3 EdStem

It is fine to use EdStem to ask for clarifications about the spec wording. However, EdStem should not be used to discuss the coursework solution. Therefore, please remember not to include in your questions any explanation about how you are trying to solve the problems.

## 4 Pairing

We recommend you pair with another student to reduce your workload. If you cannot find another student to pair with, you are welcome to use EdStem to message the class. Alternatively, you can solve the coursework on an individual basis.