TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA
Macroarea di Ingegneria

A QoS-Aware Broker for Multi-Provider Serverless Applications

Andrea Graziani
m. 0273395

Supervisor: Valeria Cardellini
Tutor: Gabriele Russo Russo

May 23, 2022

Serverless Computing: Overview

My thesis is focused on "**Serverless Computing**".

It is a development paradigm according to which:

- Administration tasks (provisioning, monitoring, scaling, etc.) are directly managed by the provider.
- Small-granularity billing pricing model: **pay-as-you-go**.
- Cloud application are abstracted as a group of so-called "**Serverless Functions**".

Serverless Computing: Overview

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

What is a serverless function?

A computation unit implementing a business functionality.

- **Stateless**
- **Event-Driven**
- **Short-Lived**

A serverless function is executed inside a containerized environment: the so-called “**Function Instance**”.

The FaaS platform **Automatically** scales the number of function instances.

FaaS platforms impose a **Limit** on the number of function instance runnable at the **Same Time** called **Concurrency Limit**.

A delay is observed when a new function instance is started by the provider: this event is called **Cold Start**.

Serverless Computing: Overview

To invoke a **serverless function**, users have to specify a so-called **serverless function configuration**.

To invoke a **serverless application**, users have to specify a configuration for **all** its functions.

Configuration parameters **significantly** affect the **cost** and **response time** of serverless functions.

Serverless Computing: Problems

- The fulfillment of **Non-Functional Requirements** concerning the **Quality of Service** (QoS) levels that should be guaranteed for any **generic** serverless workflow.
- The lack of support for application whose functions are hosted on **Multiple Providers**.
- The lack of support for serverless function implementations abstraction, that is, for the so-called **Concrete Functions**.
- The fulfillment of **Functional Requirements** concerning the **orchestration** of **Multi-Provider, Multiple-Implementation Serverless Applications**.

Solutions concerning QoS fulfillment **already exist**.

- Despite there are some exceptions, they are unaware of both the current status of FaaS platforms and user traffic.
- Some solutions provide no support for generic workflow.
- No support for Multi-Provider and Multiple-Implementations serverless applications.
  - No one provides an analytical model to evaluate their performance.
- Many solutions rely on QoS-aware scheduling algorithms while others rely on the formulation and solving of optimization problems.

Goal # 1

To guarantee the **Satisfaction of QoS Levels** for Multi-Provider and Multiple-Implementations serverless applications.

It was necessary to develop:

1. An **Analytical Model** to evaluate applications performance.
    - Supporting generic workflow (branch, loop, parallel)
    - Providing support for Multi-Provider, Multiple-Implementation Serverless Applications
    - FaaS-Status-Aware.

Thesis Goals

2. A **Methodological Way** to find the "best" configuration to satisfy QoS constraints.
   - By solving an **Optimization Problem** (LP).
3. A custom **Heuristic Algorithm** to rapidly resolve the aforementioned optimization problem.
   - Based on **Ant Colony Optimization** algorithm (ACO) family.

Goal # 2

The **Orchestration** for Multi-Provider and Multiple-Implementations serverless applications.

To achieve it, I had to build:

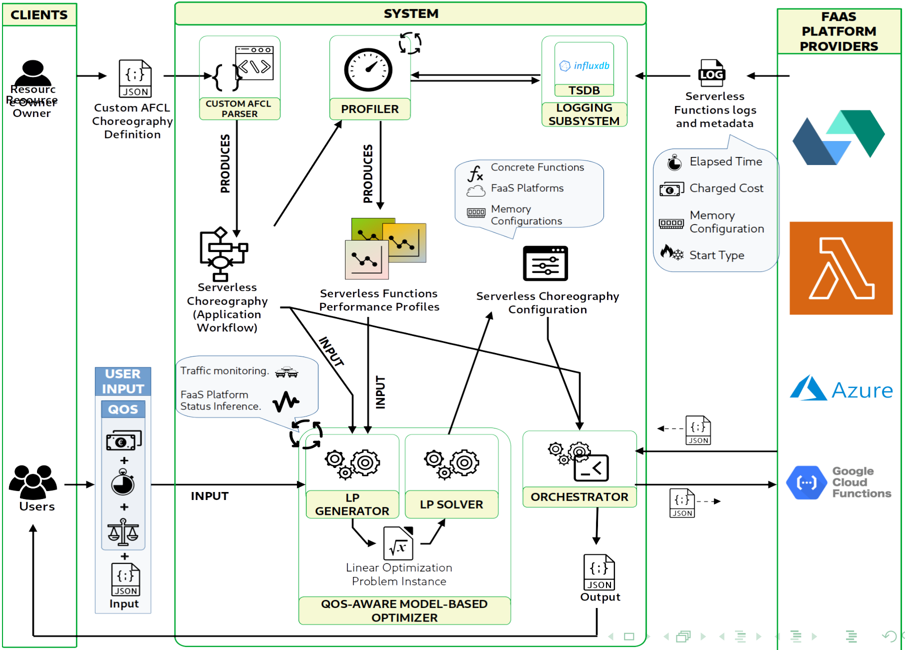
1. A **Software Framework**
   - Users/Serverless Application Management (CRUD operations, profiling tasks, orchestration task, etc.)
   - User-Traffic/QoS/FaaS-Status Aware.

2. An extension to an already existent **Representation Scheme** to define a serverless application workflow.
   - Based on an existing language called **Abstract Function Choreography Language** (AFCL).

The Prototype

Main features of our software framework:

- **Client-Server architecture**.
- **Cloud-native** application.
- Includes a set of **adapters** to interact with following FaaS providers:
  - AWS Lambda.
  - Apache OpenWhisk.
- **REST** architectural style.


CLIENTS
SYSTEM
FAAS PLATFORM PROVIDERS
Resource Owner
Custom AFCL Choreography Definition
JSON
CUSTOM AFCL PARSER
PROFILER
influxdb
TSDB
LOGGING SUBSYSTEM
LOG
Serverless Functions logs and metadata
Elapsed Time
Charged Cost
Memory Configuration
Start Type
PRODUCES
PRODUCES
Serverless Choreography (Application Workflow)
Serverless Functions Performance Profiles
Concrete Functions
FaaS Platforms
Memory Configurations
Serverless Choreography Configuration
INPUT
INPUT
Traffic monitoring.
FaaS Platform Status Inference.
USER INPUT
QOS
Input
Users
INPUT
LP GENERATOR
LP SOLVER
Linear Optimization Problem Instance
QOS-AWARE MODEL-BASED OPTIMIZER
ORCHESTRATOR
Output
Azure
Google Cloud Functions

The Analytical Model

Serverless applications are abstracted to a
Weakly Connected Weighted Directed Graph.

- Each **vertex** models an **abstract serverless function**.
  - To each abstract function corresponds a set containing all concrete function implementing it.
- Each **edge** represents the calling relationship between two abstract functions.
  - Edge weights represent the so-called **transition probability**.

The Analytical Model

The model provides algorithms and equations to evaluate the performance of a **serverless application** under a given configuration.

Aforementioned algorithm depends on:

- The topological properties of the graph representing the application.
- The actual value of transition probabilities of all edges.
- The estimation of concrete function performance.

The Analytical Model

To evaluate the performance of a concrete function is required:

- An estimations about the **Average Response Time** (**Cost**) in case of cold (warm) start are required.
  - To compute aforementioned estimations, **Exponential Moving Average** based approach is adopted.

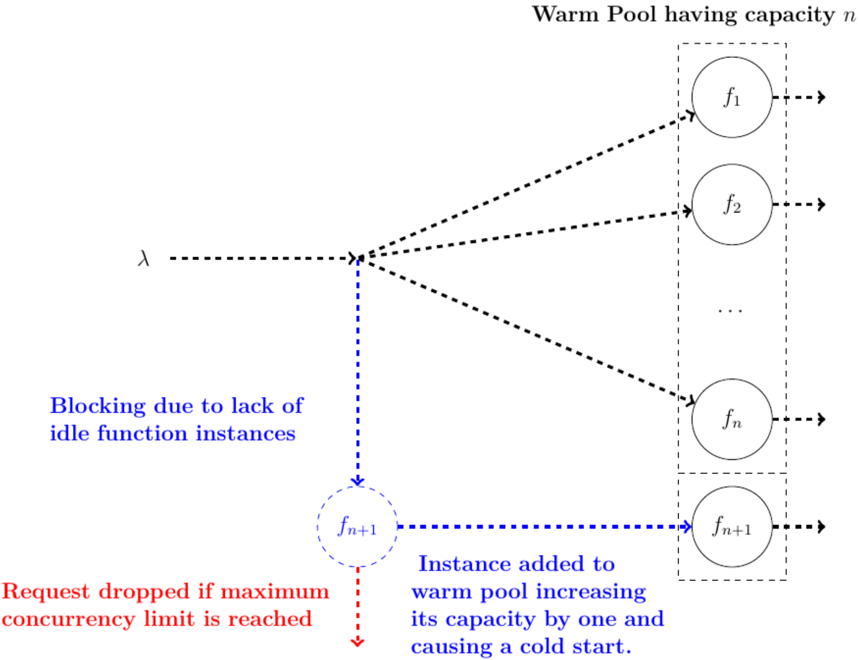
- An estimation of the probability according to which a request follows a cold start (**Cold Start Probability**)

The Analytical Model

Any FaaS platform provider is modeled by set of $M/G/K(t)_{C_{max}}/K(t)_{C_{max}}$ queueing systems.

- $K(t)$: the number of function instances at time t.
- C_{max}: the concurrency limit.

To compute the cold start probability the **Erlang-B** formula is used.



Warm Pool having capacity n
f_1
f_2
λ
...
f_n
Blocking due to lack of idle function instances
f_{n+1}
f_{n+1}
Instance added to warm pool increasing its capacity by one and causing a cold start.
Request dropped if maximum concurrency limit is reached

The Optimization Problem

- To achieve our goal consisting in finding the best configuration to guarantee QoS constraints, we have to solve an **optimization problem**.
  - It is based on **Multi-Dimensional Multi-Choice Knapsack Problem Formulation**.

Formally, the configuration vector $\mathbf{x}_{\mathscr{C}}$ we want to find is such that:

$$
\begin{aligned}
\mathbf{x}_{\mathscr{C}} &\overset{\text{def}}{=} \left\{ x_{\phi_1}, \dots, x_{\phi_k} \right\} \\
&\in \left\{ \left\{ \bigcup_{j=1}^{|\mathbf{F}_{\phi_1}|} f_{\phi_{1_j}} \times \mathbf{M}_{f_{\phi_{1_j}}} \right\} \times \dots \times \left\{ \bigcup_{j=1}^{|\mathbf{F}_{\phi_k}|} f_{\phi_{k_j}} \times \mathbf{M}_{f_{\phi_{k_j}}} \right\} \right\} \\
&= \bigtimes_{i=1}^{k} \left\{ \bigcup_{j=1}^{|\mathbf{F}_{\phi_i}|} f_{\phi_{i_j}} \times \mathbf{M}_{f_{\phi_{i_j}}} \right\} \\
&\subseteq \bigtimes_{i=1}^{k} \left\{ \mathbf{F}_{\phi_i} \times \mathbb{N} \right\} = \mathbf{X}_{\mathscr{C}}
\end{aligned} \tag{1}
$$

$$
\begin{aligned}
max \quad & \sum_{i=1}^{|\mathcal{F}_{\mathcal{E}}(\mathcal{C})|} \sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} \cdot p_{\phi_{i_j}}(t) && (5.10)\\
\text{subject to} \quad & \sum_{i=1}^{|\mathcal{F}_{\mathcal{E}}(\mathcal{C})|} \sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} \cdot c_{\phi_{i_j}}(t) \leq C && (5.11)\\
& \sum_{\phi_i \in \mathcal{F}_{\mathcal{E}}(\mathcal{C}) \setminus \mathcal{F}_{\mathcal{E}}(\widetilde{\mathcal{P}})} \sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} \cdot rt_{\phi_{i_j}}(t) + \\
& \quad + \sum_{\phi_h \in \delta_{\mathcal{C}}} \sum_{j=1}^{|\mathbf{F}_{\phi_h} \times \mathbb{N}|} y_{\phi_{h_j}} \cdot rt_{\phi_{h_j}}(t) \leq RT && \forall \delta_{\mathcal{C}} \in \Delta_{\mathcal{C}} \\
& && (5.12)\\
& \sum_{i=1}^{|\mathcal{F}_{\mathcal{E}}(\mathcal{C})|} \sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} \cdot a_{(\phi_{i_j}, \omega_P^{(l)})} \leq l - R(\mathbf{Q}_{\omega_P^{(l)}}, t) && \forall \omega_P^{(l)} \in \widetilde{\mathbf{S}}_{\mathcal{C}} \\
& && (5.13)\\
& \sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} = 1 && \forall i \in \mathbb{N} \cap [1, |\mathcal{F}_{\mathcal{E}}|] \\
& && (5.14)\\
& y_{\phi_{i_j}} \in \{0, 1\} && \begin{array}{r} \forall i \in \mathbb{N} \cap [1, |\mathcal{F}_{\mathcal{E}}|] \\ \forall j \in \mathbb{N} \cap [1, |\mathbf{F}_{\phi_i} \times \mathbb{N}|] \end{array}
\end{aligned}
$$

The Heuristic Algorithm

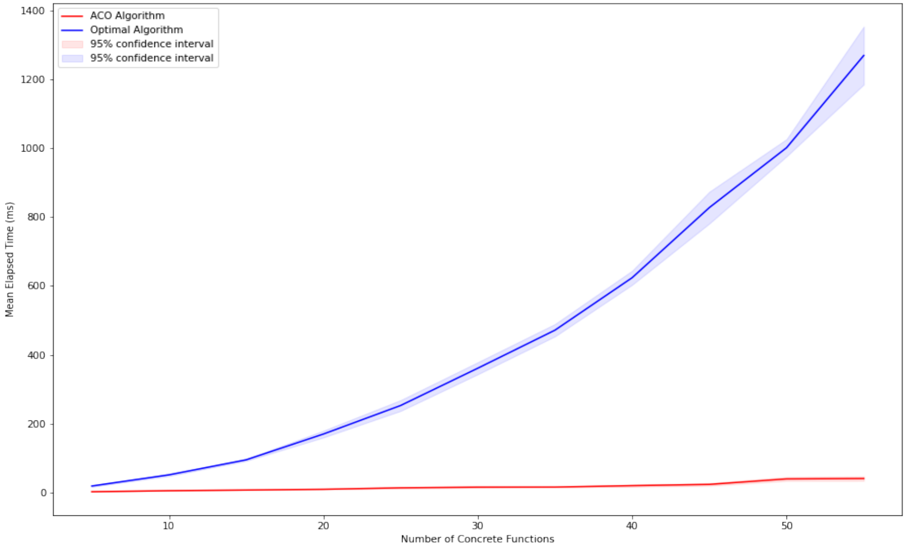
I develop a custom heuristic algorithm based on **Ant Colony Optimization** (ACO); it is called **Pre-provisioned Colony Optimization Algorithm with Lazy Pheromone Update**.

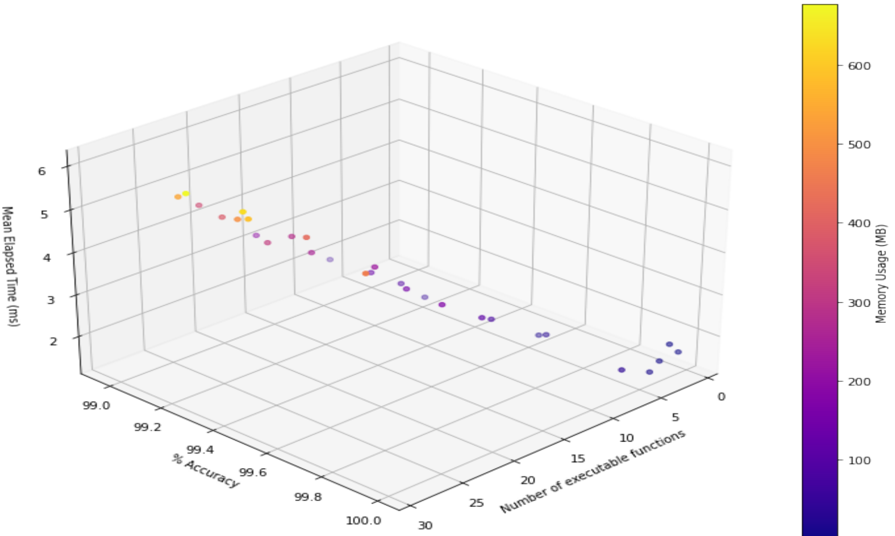
- It is based on a set of computational agents, called **artificial ants**, which **iteratively** construct a solution.
- At each iteration, each agent moves from a solution to another, applying a series of stochastic **local** decisions whose policy is based on following parameters:
  - **Attractiveness**.
  - **Pheromone Trail**.

Pheromone trails are updated during **every iteration**.

Pheromone trails are used to decide which solutions should be preferred during **next iterations**.

- A **Lazy Approach** for pheromone trails update is used.
- A **Pre-Provisioning Tactic** is used to anticipates data needs assuring a **lower latency**.


ACO Algorithm
Optimal Algorithm
95% confidence interval
95% confidence interval
Mean Elapsed Time (ms)
Number of Concrete Functions
0
200
400
600
800
1000
1200
1400
10
20
30
40
50



Mean Elapsed Time (ms)
6
5
4
3
2
99.0
99.2
99.4
99.6
99.8
100.0
% Accuracy
0
5
10
15
20
25
30
Number of executable functions
Memory Usage (MB)
600
500
400
300
200
100
0

Conclusions

- I presented an analytical model to compute the performance of a generic serverless workflows having multiple concrete functions hosted on multiple FaaS providers.
- I defined an optimization problems formulation to address the problem of finding a suitable configuration in order to meet user defined QoS constraints
- We developed a heuristic algorithm to rapidly solve it.
- We verified the validity of both the proposed algorithms and the analytical model through test and experimental evaluations developing a prototype supporting AWS and OpenWhisk.

Thanks for your attention!
Questions?