



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

My thesis is focused on “**Serverless Computing**”.

Two main goals:

- 1 The **orchestration** of a serverless workflow supporting:
 - Multiple FaaS Providers
 - Multiple Functions Implementations.
- 2 The fulfillment of **non-functional requirements** concerning the **quality of service** (QoS) levels that should be guaranteed.

What's meant by serverless computing?

An applications service model where:

- 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"**.

What is a serverless function?

A computation unit implementing a business functionality.

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

Serverless Application

A software system made up of a serverless functions set.

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

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

When a request comes in, only one of the following events can occur:

- **Warm start.**
- **Cold start.**

Concurrency Level

The limitation, imposed by FaaS platforms, on the **number** of function instance runnable at the **same time**.

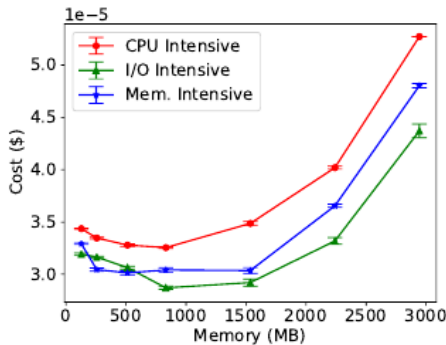
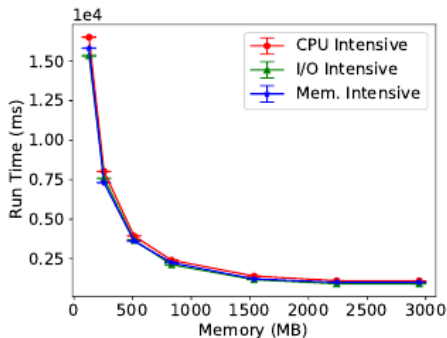
Providers apply that restriction differently:

- **Global (Per-Account) Concurrency Model.**
 - AWS Lambda.
 - IBM Cloud Functions.
- **Local (Per-Function) Concurrency Model.**
 - Google Cloud Functions.

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.



What current problems or limitations of the serverless computing paradigm I tried to solve?

- No support for application whose functions are hosted on multiple providers.
- No support for application whose functions have more than one implementation (**Concrete Functions**).
- Vendor lock-in issues.
- The fulfillment of quality of service (QoS) levels.

Why is so difficult to guarantee QoS levels?

- Lack of an analytical model to predict application performance.
- Lack of a methodological way to find a suitable configuration for a serverless application.

Solutions **already** exist, but:

- They are unaware of the current status of FaaS platforms.
- No support for Multi-Provider, Multiple-Implementations serverless applications.
- Many do not support generic workload applications.

My contributions respect to the current state of art:

- ① An **Analytical Model** to evaluate applications performance.
 - Supports generic workflow (branch, loop, parallel)
 - Multi-Provider support.
 - Support for Multi-Implementation for functions.
 - FaaS-Status-Aware.

- ② A **Methodological Way** to find the “best” configuration to satisfy QoS constraints.
 - By solving an **Optimization Problem** (LP).

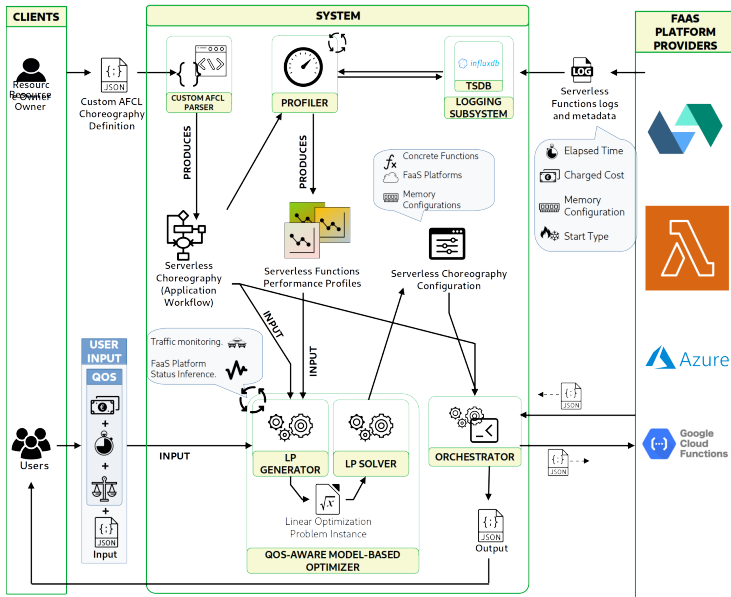
- ③ A custom **Heuristic Algorithm** to resolve the aforementioned optimization problem.
 - Based on **Ant Colony Optimization** algorithm (ACO) family.
- ④ A **QoS-Aware Software Framework** for the orchestration for multi-provider, multiple-implementations serverless applications.
- ⑤ An extension to an already existent **Representation Scheme** to define a serverless application workflow.
 - Based on an existing language called **Abstract Function Choreography Language** (AFCL).

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.

Main logical entities of our system:

- A Logging subsystem.
- An Orchestrator.
- A profiler.
- A QoS-aware model-based optimizer.
- A Custom AFCL Parser.



The use of an unique representation scheme for serverless workflows **mitigates portability limitations and vendor lock-in issues.**

Serverless application are managed using identical operations based on standard HTTP methods providing **Access Transparency.**

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

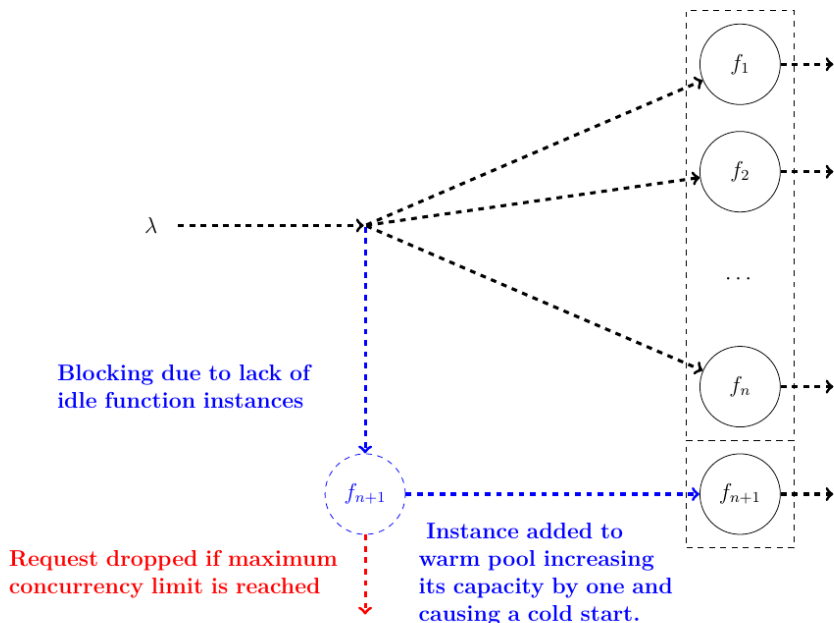
- Each **vertex** models an **abstract serverless function**, that is a computational unit required by business logic.
 - To each abstract function corresponds a set containing all concrete function implementing it.
- Each **edge** represents the calling relationship between two abstract functions.
 - Edge weights represent the so-called **transition probability**.

To evaluate the performance of a **Concrete Function**:

- 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**) is required.

To compute **Cold Start Probability**:

- 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.
- Aforementioned set depends on the concurrency model adopted by providers.
- Erlang-B** formula is used.



I formalized algorithms and equations to evaluate the performance of both a **serverless application** and of an **abstract function**.

Aforementioned algorithm depends on:

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

- 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 $\mathbf{x}_{\mathcal{C}}$ we want to find is such that:

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

The number of all possible configuration grow very fast.

- An application with:
 - 6 functions.
 - 46 possible memory choice.
 - **Only one** implementation for each function.

lead to $9.47 \cdot 10^9$ different configurations.

- An application with:
 - 6 functions.
 - 46 possible memory choice.
 - **Two** implementation for each function.

lead to $6.06 \cdot 10^{11}$ different configurations.

$$\max \sum_{i=1}^{|\mathcal{F}_{\mathcal{E}}(C)|} \sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} \cdot p_{\phi_{i_j}}(t) \quad (5.10)$$

$$\text{subject to } \sum_{i=1}^{|\mathcal{F}_{\mathcal{E}}(C)|} \sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} \cdot c_{\phi_{i_j}}(t) \leq C \quad (5.11)$$

$$\begin{aligned} & \sum_{\phi_i \in \mathcal{F}_{\mathcal{E}}(C) \setminus \mathcal{F}_{\mathcal{E}}(\tilde{\mathcal{P}})} \sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} \cdot rt_{\phi_{i_j}}(t) + \\ & + \sum_{\phi_h \in \delta_C} \sum_{j=1}^{|\mathbf{F}_{\phi_h} \times \mathbb{N}|} y_{\phi_{h_j}} \cdot rt_{\phi_{h_j}}(t) \leq RT \end{aligned} \quad \forall \delta_C \in \Delta_C \quad (5.12)$$

$$\sum_{i=1}^{|\mathcal{F}_{\mathcal{E}}(C)|} \sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} \cdot a_{(\phi_{i_j}, \omega_P^{(l)})} \leq l - R(\mathbf{Q}_{\omega_P^{(l)}}, t) \quad \forall \omega_P^{(l)} \in \tilde{\mathbf{S}}_C \quad (5.13)$$

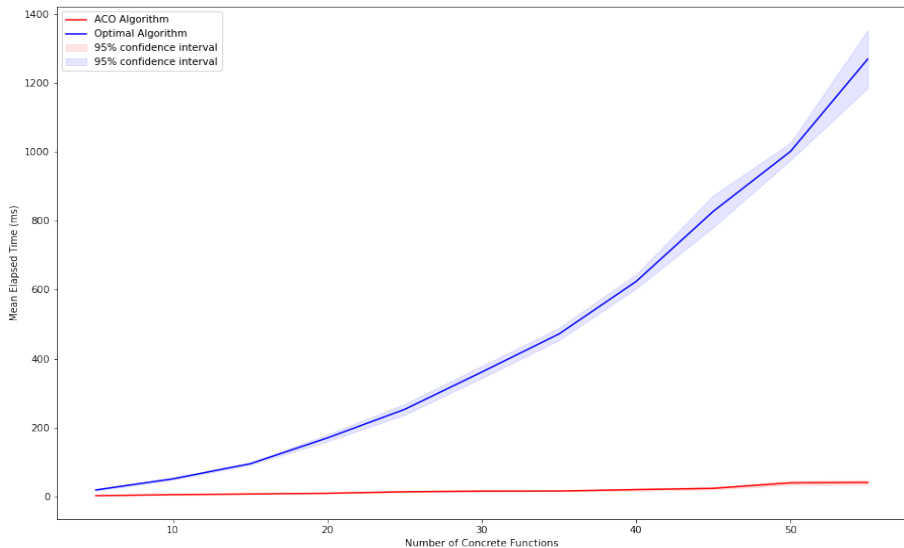
$$\sum_{j=1}^{|\mathbf{F}_{\phi_i} \times \mathbb{N}|} y_{\phi_{i_j}} = 1 \quad \forall i \in \mathbb{N} \cap [1, |\mathcal{F}_{\mathcal{E}}|] \quad (5.14)$$

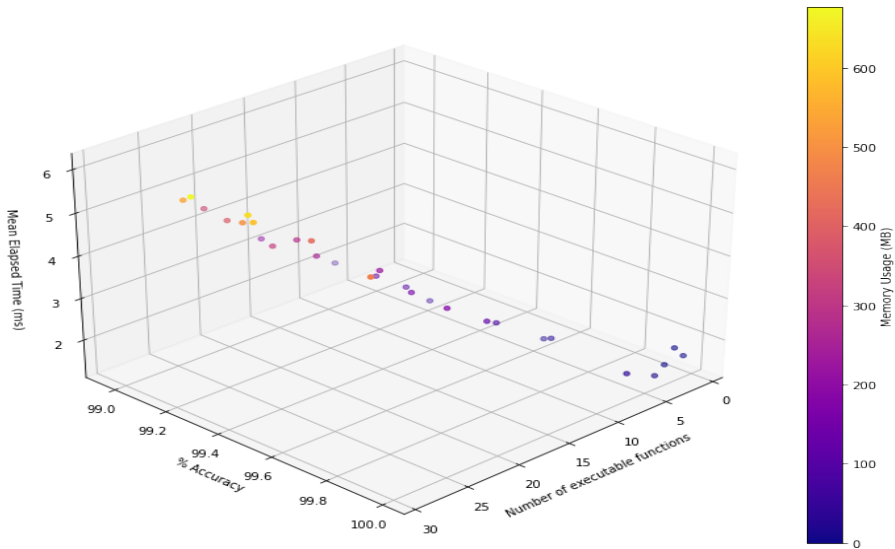
$$y_{\phi_{i_j}} \in \{0, 1\} \quad \forall i \in \mathbb{N} \cap [1, |\mathcal{F}_{\mathcal{E}}|] \\ \forall j \in \mathbb{N} \cap [1, |\mathbf{F}_{\phi_i} \times \mathbb{N}|]$$

The heuristic algorithm is based on Ant Colony Optimization (ACO), a class of stochastic meta-heuristics: 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 so-called partial solution.
- At each iteration, each artificial ant moves from a partial solution to another, applying a series of stochastic local decisions whose policy is based on following parameters
 - **Attractiveness.**
 - **Pheromone Trail.**

- Lazy approach for pheromone trails update on the solution component graph.
- We exploit pre-provisioning tactic to anticipates data needs assuring a lower latency.







Thanks for your attention!
Questions?