



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**.

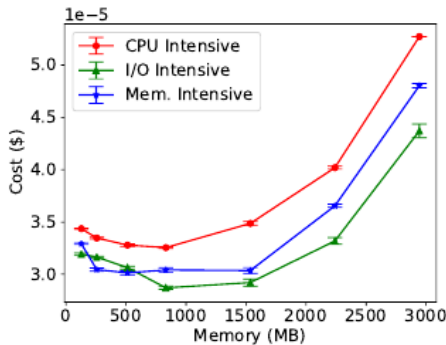
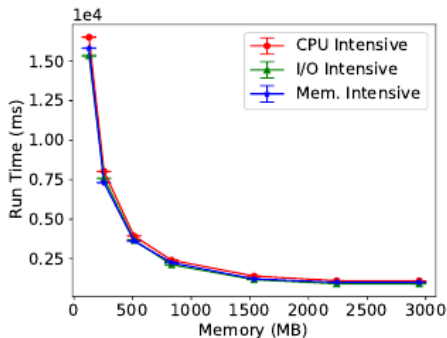
Providers apply that restriction differently:

- **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).

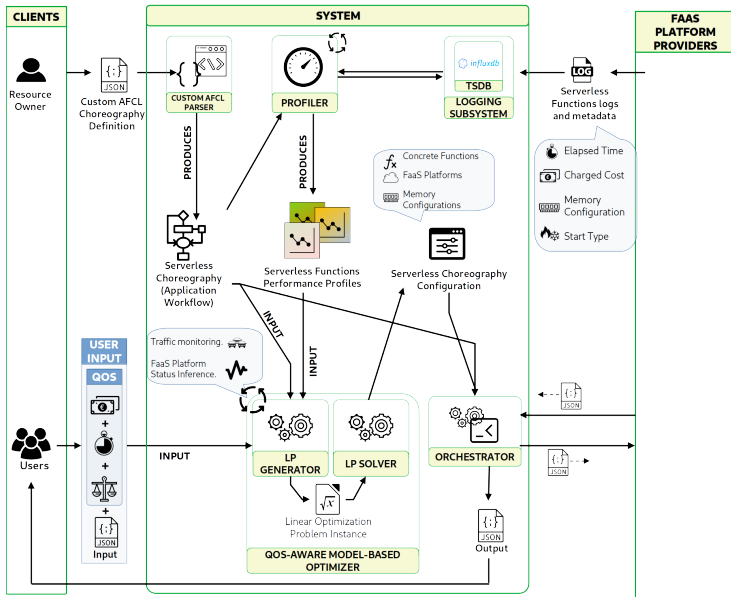
- ③ A custom **Heuristic Algorithm** to resolve the aforementioned optimization problem.
 - Based on **Ant Colony Optimization** algorithm (ACO) family.
- ④ A **QoS-Aware Software Framework** for the orchestration for multi-provider, multiple-implementations serverless applications.
- ⑤ An extension 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.



Formally, let a choreography $\mathcal{C} = (\Phi, E)$, when an invocation request of \mathcal{C} arrive on our system, the latter acts as follows:

- For each $\phi \in \mathcal{F}_{\mathcal{C}}(\mathcal{C})$, it selects only one concrete function $f_{\phi} \in \mathbf{F}_{\phi}$, which will be effectively invoked and executed on its corresponding FaaS platform.
- For each selected concrete function f_{ϕ} , it selects a value for memory size.

According to our model point of view, at any time t , any FaaS platform provider acts as a “set” of $M/G/K(t)_{c_{max}}/K(t)_{c_{max}}$ queueing systems, also called $K(t)$ -server loss systems, where:

Then, let $\phi \in \mathcal{F}_{\mathcal{E}}(\mathcal{C})$ an executable function and \mathbf{F}_{ϕ} the corresponding implementation-set, formally an executable function configuration x_{ϕ} for the executable function ϕ is a two-dimensional vector defined as follows:

$$x_{\phi} = (f_{\phi}, m) \in f_{\phi} \times \mathbf{M}_{f_{\phi}} \subseteq \mathbf{F}_{\phi} \times \mathbb{N} \quad (1)$$

where:

- $f_{\phi} \in \mathbf{F}_{\phi}$ denotes a particular concrete function implementing the executable function ϕ .
- $m \in \mathbf{M}_{f_{\phi}}$ represents the allocated memory size during the execution of f_{ϕ} , where $\mathbf{M}_{f_{\phi}} \subseteq \mathbb{N}$ is the set holding all available memory size configurations allowed by provider where the concrete function f_{ϕ} is executed.

Formally, a serverless choreography configuration $\mathbf{X}_{\mathcal{C}}$ for the choreography \mathcal{C} is a vector such that:

$$\begin{aligned}
 \mathbf{x}_{\mathcal{C}} &\stackrel{\text{def}}{=} \left\{ x_{\phi_1}, \dots, x_{\phi_k} \right\} \\
 &\in \left\{ \left\{ \bigcup_{j=1}^{|\mathbf{F}_{\phi_1}|} f_{\phi_{1j}} \times \mathbf{M}_{f_{\phi_{1j}}} \right\} \times \dots \times \left\{ \bigcup_{j=1}^{|\mathbf{F}_{\phi_k}|} f_{\phi_{kj}} \times \mathbf{M}_{f_{\phi_{kj}}} \right\} \right\} \\
 &= \bigtimes_{i=1}^k \left\{ \bigcup_{j=1}^{|\mathbf{F}_{\phi_i}|} f_{\phi_{ij}} \times \mathbf{M}_{f_{\phi_{ij}}} \right\} \\
 &\subseteq \bigtimes_{i=1}^k \left\{ \mathbf{F}_{\phi_i} \times \mathbb{N} \right\} = \mathbf{X}_{\mathcal{C}} \tag{2}
 \end{aligned}$$

Concrete Functions

Set of semantically and logically equivalent serverless function implementations, accepting the same set of input data and returning the same type of output data. They may expose different performance or cost behavior.



Thanks for your attention!
Questions?

Formally, a serverless choreography configuration $\mathbf{X}_{\mathcal{C}}$ for the choreography \mathcal{C} is a vector such that:

$$\begin{aligned}
 \mathbf{x}_{\mathcal{C}} &\stackrel{\text{def}}{=} \left\{ x_{\phi_1}, \dots, x_{\phi_k} \right\} \\
 &\in \left\{ \left\{ \bigcup_{j=1}^{|\mathbf{F}_{\phi_1}|} f_{\phi_{1j}} \times \mathbf{M}_{f_{\phi_{1j}}} \right\} \times \dots \times \left\{ \bigcup_{j=1}^{|\mathbf{F}_{\phi_k}|} f_{\phi_{kj}} \times \mathbf{M}_{f_{\phi_{kj}}} \right\} \right\} \\
 &= \bigtimes_{i=1}^k \left\{ \bigcup_{j=1}^{|\mathbf{F}_{\phi_i}|} f_{\phi_{ij}} \times \mathbf{M}_{f_{\phi_{ij}}} \right\} \\
 &\subseteq \bigtimes_{i=1}^k \left\{ \mathbf{F}_{\phi_i} \times \mathbb{N} \right\} = \mathbf{X}_{\mathcal{C}}
 \end{aligned} \tag{3}$$