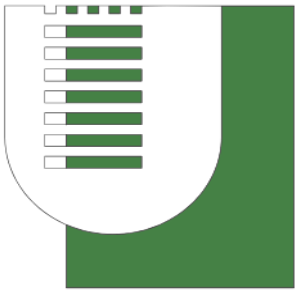


TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

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

Goals

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.

Serverless Computing: Overview

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 Computing: Overview

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

Any function instance can be in:

- Initialization State.
- Idle State.
- Running State

Warm Pool

The set of all function instances whose state is either **idle** or **running** state.

Serverless Computing: Overview

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.

Serverless Computing: Problems

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

- Vendor lock-in issues.
- The fulfillment of quality of service (QoS) levels.

Serverless Computing: Problems

Why is so difficult to guarantee QoS levels?

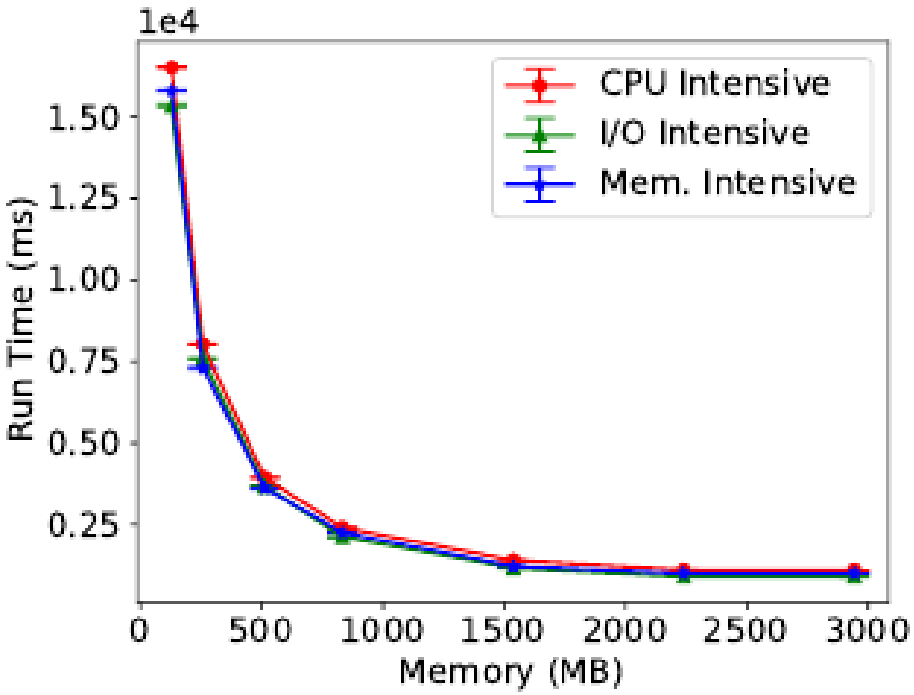
- Lack of an analytical model to predict application performance.
- Lack of a methodological way to find a suitable "**configuration**" for a serverless application.

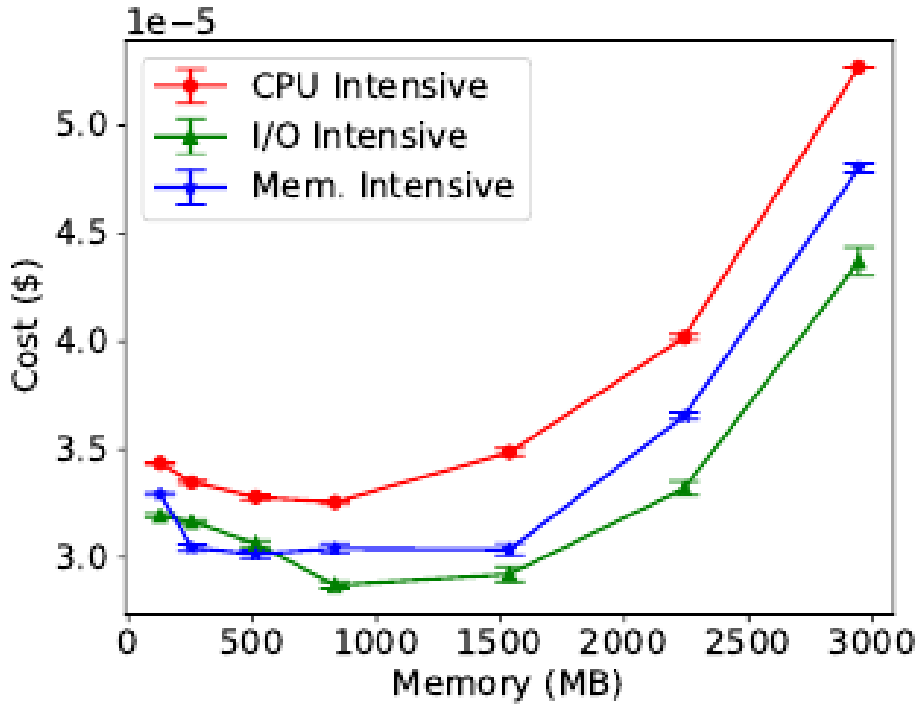
Serverless Computing: Problems

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Why is so difficult to find a suitable configuration for a serverless application?

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





Related Works

Solutions **already** exist, **but**:

- They are unaware of the current status of FaaS platforms.
- No support for multiple implementations, or versions, of a same serverless function.
- No support for multiple FaaS platforms.
- Limited support for generic workload applications.

My Contributions

My contributions respect to the current state of art:

1. An **Analytical Model** to evaluate applications performance.
   - Multi-Provider.
   - Multi-Implementation.
   - QoS-Aware.
   - FaaS-Status-Aware.

2. A **methodological way** to find the "best" configuration to satisfy QoS constraints.
   - By solving an optimization problem.

My Contributions

3. A custom **heuristic algorithm** to resolve the aforementioned optimization problem.

4. A **software framework** for the orchestration of serverless applications supporting "hybrid-scheduling".
   - Following REST architectural style.

5. An extension to an already existent **representation scheme** to define a serverless application workflow.

The Software Framework

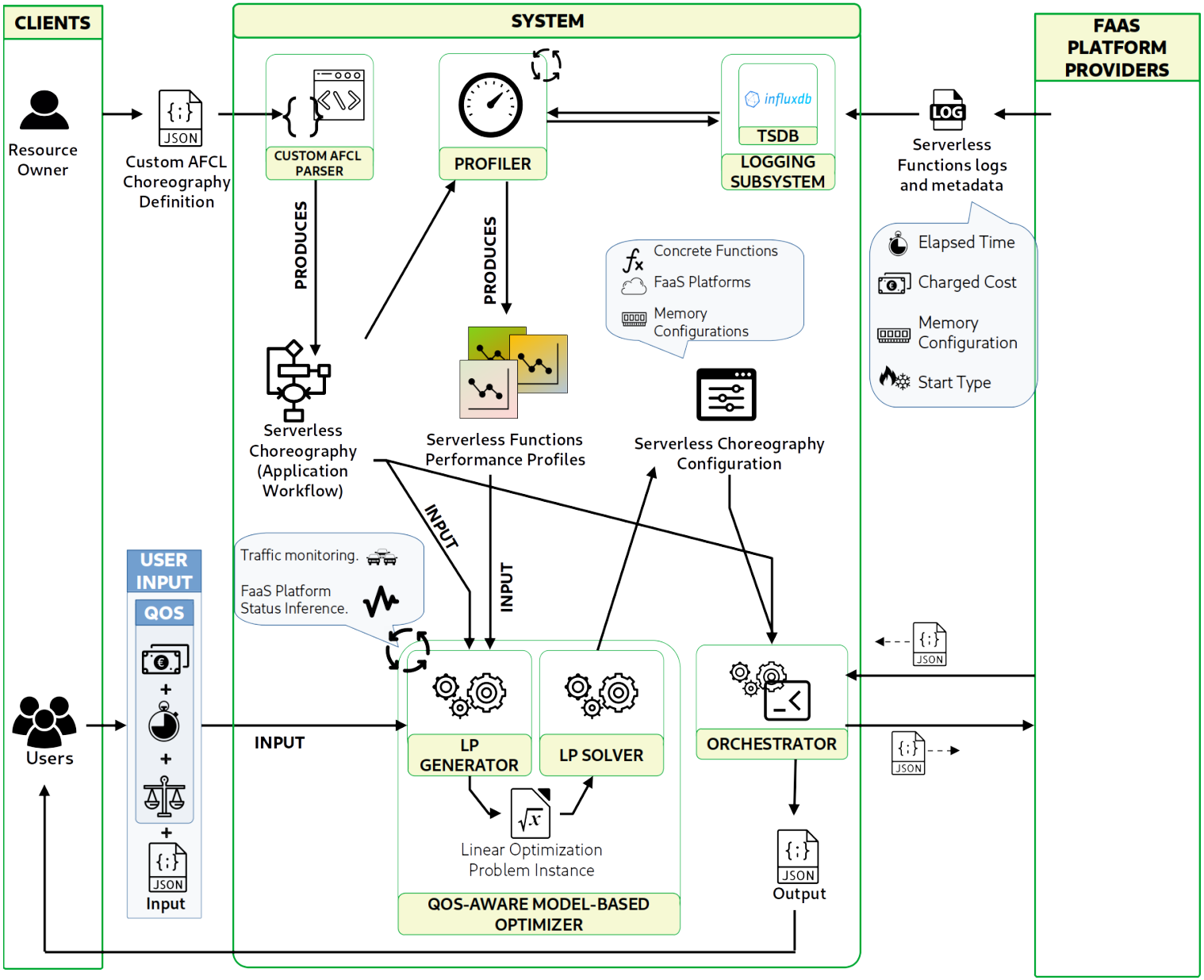
Main features of our software framework:

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

The Software Framework

Main logical entities of our system:

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


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
PRODUCES
PRODUCES
Concrete Functions
FaaS Platforms
Memory Configurations
Elapsed Time
Charged Cost
Memory Configuration
Start Type
Serverless Choreography (Application Workflow)
Serverless Functions Performance Profiles
Serverless Choreography Configuration
INPUT
INPUT
Traffic monitoring.
FaaS Platform Status Inference.
USER INPUT
QOS
Users
INPUT
LP GENERATOR
LP SOLVER
ORCHESTRATOR
Linear Optimization Problem Instance
QOS-AWARE MODEL-BASED OPTIMIZER
Output
Input

The Software Framework

An unique representation scheme is used for serverless workflows.

- It is based on an existing language called **Abstract Function Choreography Language** (AFCL).
- I extend the original one to include the support for multiple serverless function implementations hosted on multiple FaaS.

Advantages

- Access transparency.
- It overcomes portability limitations and vendor lock-in issues.

The Analytical Model

Informally, that abstraction has been derived from that of a control-flow graph which, as known, describes, using graphs notation, all paths that might be traversed through a serverless application during its execution. Similarly, a choreography describes calling relationships between functions belonging to an application in a serverless environment, combining them using several types of control-flow structures, like sequence, branch, loop or connectors for a parallel execution.

System Model

System Model

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

- For each $\phi \in \mathscr{F}_{\mathscr{E}}(\mathscr{C})$, 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.

System Model

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

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

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

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 $\mathbf{X}_{\mathscr{C}}$ for the choreography $\mathscr{C}$ is a vector such that:

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

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 $\mathbf{X}_{\mathscr{C}}$ for the choreography $\mathscr{C}$ is a vector such that:

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