Politecnico di Milano, Italy
DEEPSe Group, DEIB

National University of Comahue,
Patagonia, Argentina

National Scientific and Technical
Research Council, Argentina

Towards the Serverless Continuum

Martin Garriga*
martin.garriga@polimi.it

*In collaboration with Luciano **Baresi**, Sam **Guinea**,
Danilo **Filgueira** and Giovanni **Quattrocchi**

About me

Born and raised (and studied!) in Neuquen, Patagonia, Argentina
Postdoc research fellow @ Politecnico Di Milano, Italy since 2016

Roadmap

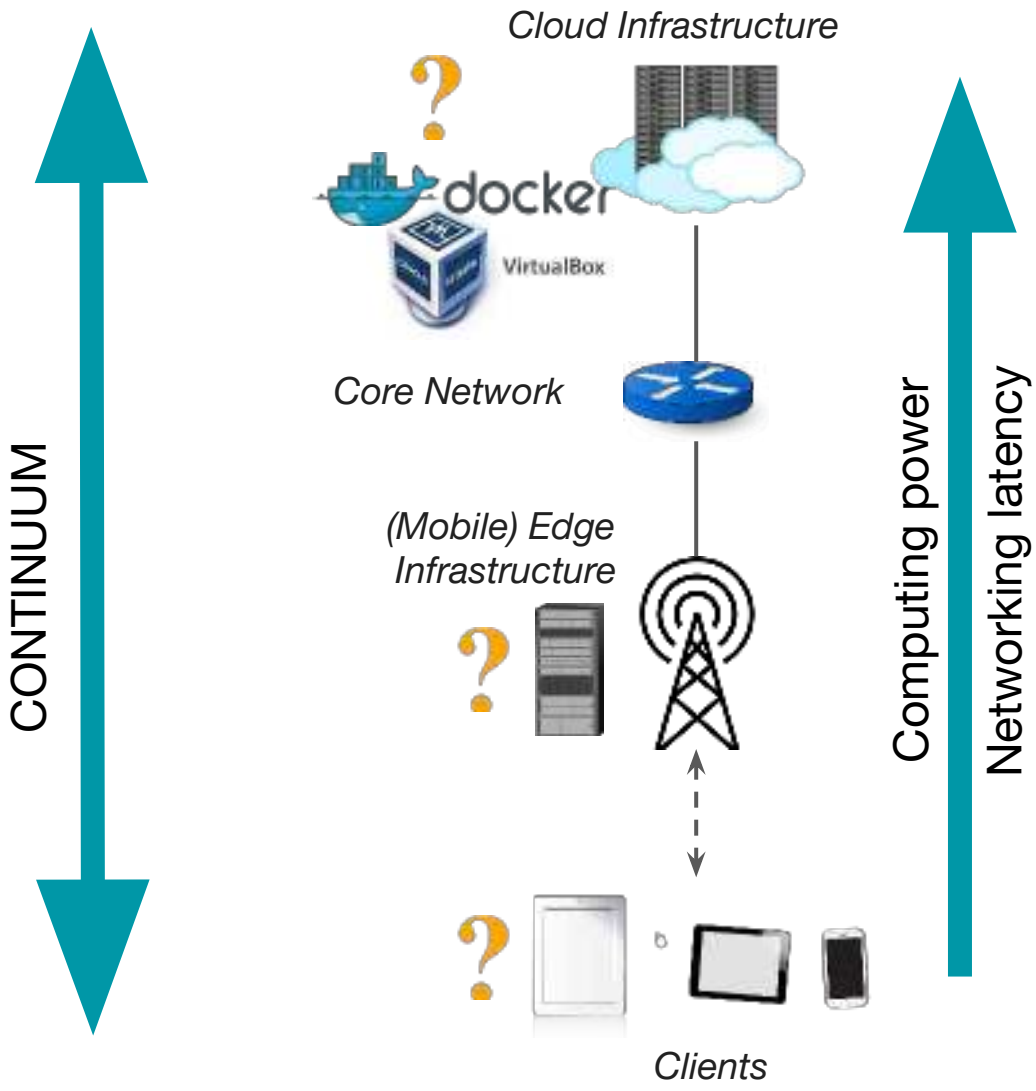
- ➔ Problem

  Pushing the boundaries of computation offloading
- ➔ Background

  Continuum?? Serverless??
- ➔ Proposal

  A Unified Model for the Mobile-Edge-Cloud Continuum
- ➔ Evaluation
- ➔ Final Remarks & Future Work

Problem



Cloud
- multiple hops away
- high network latency
+ 'unlimited' resources
+ virtualization and containerization

? execution model?

Mobile Edge
+ single or few hops away
+ low network latency
- limited resources

? execution model?

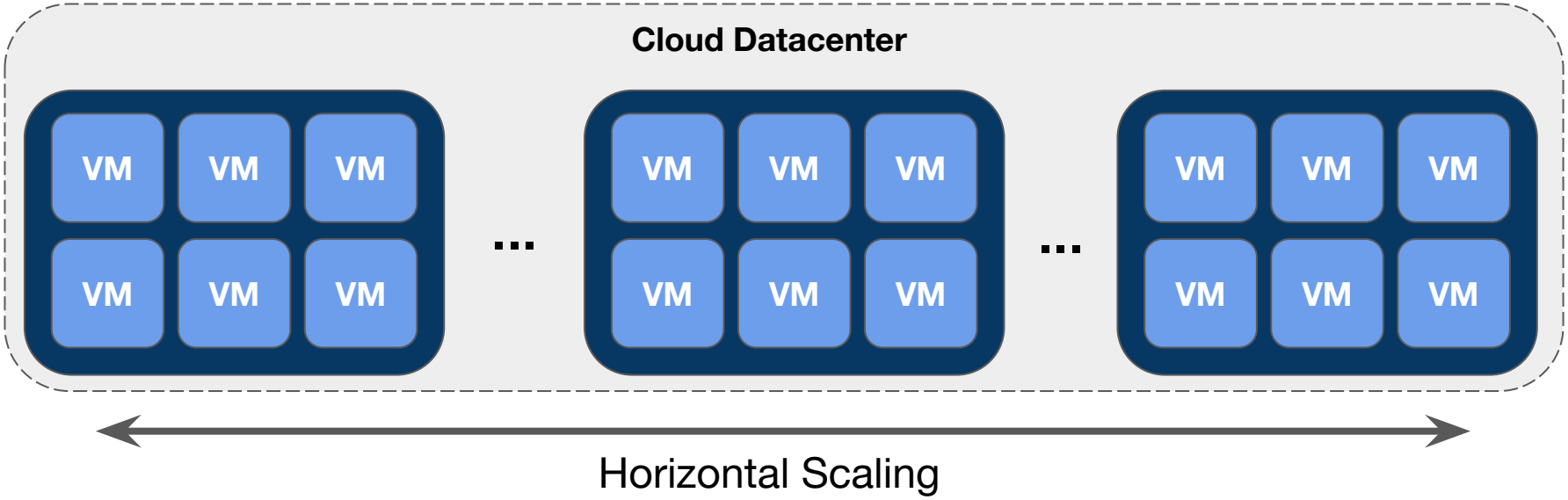
Mobile 'prosumers'
- constrained resources
- high processing latency
- battery limitation

? execution model?

Offloading should be…

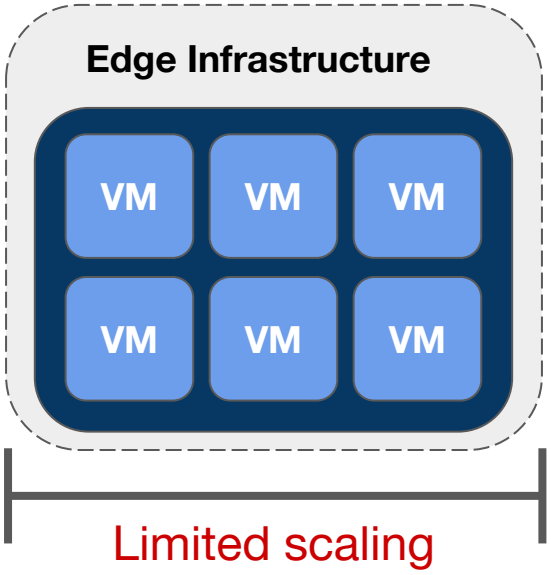
- **Dynamic**
- **Transparent**
- **QoS-Aware**
- **…**

Problem



Backend applications deployed in virtual machines
Works well in cloud infrastructure

Problem

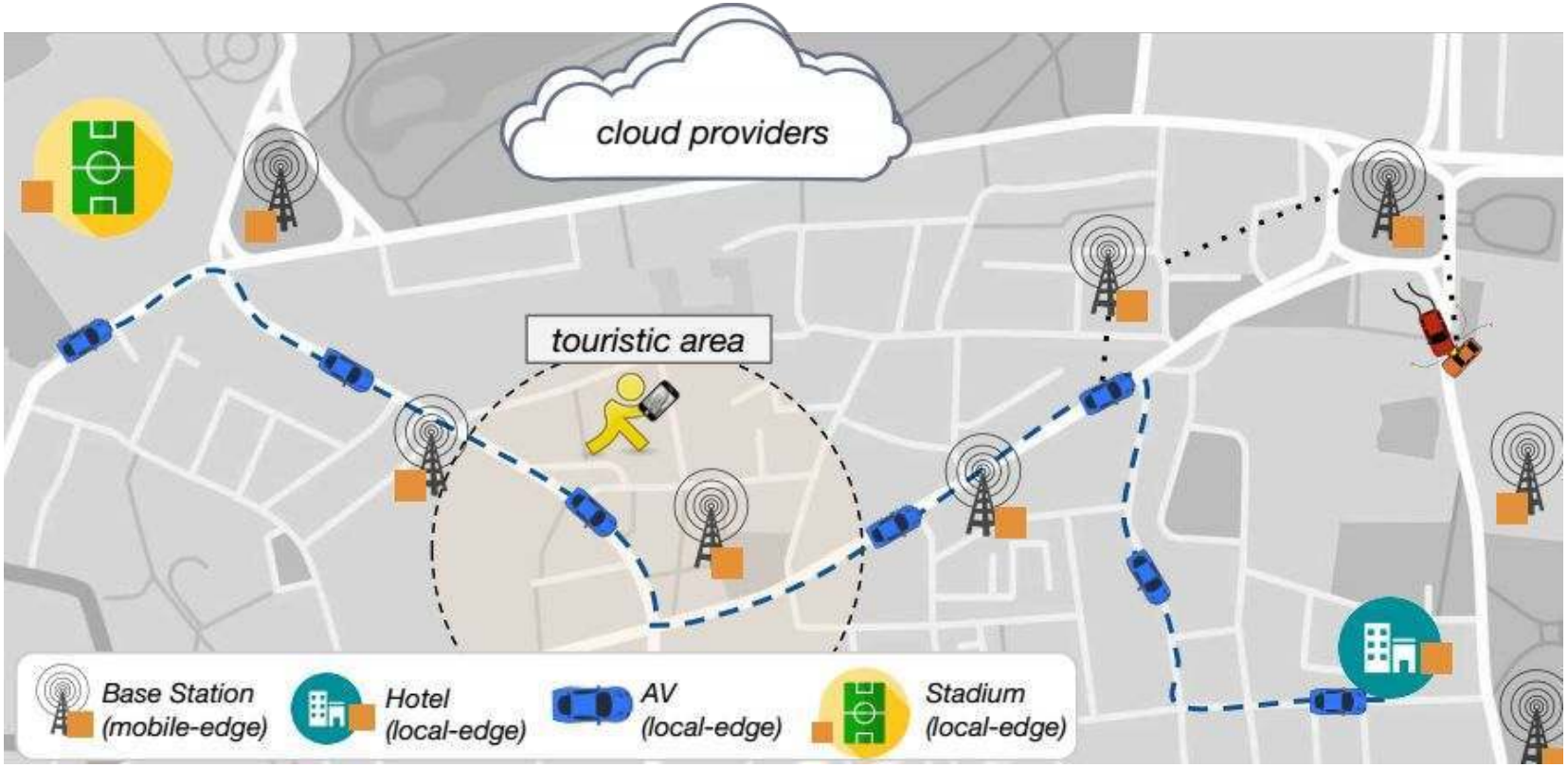


Backend applications deployed in virtual machines
Fail to scale in Edge infrastructure

Background

Example: Mobile-Edge-Cloud Continuum for Smart Cities

Augmented Reality app --- Autonomous Vehicles



Background

- Serverless Architecture
  - Emerged as an alternative computing model for cloud computing
- Function as a Service (FaaS)
  - Programming/Execution/Deployment model in a *serverless architecture*
  - There is no need for preallocation of resources
    - Resources are shared and managed by a platform
  - Functions can be exposed as RESTful services
  - Enables pay-per-use billing model
  - More elastic and reactive than scaling virtual machines and containers
    - Multiple instances of functions
    - Functions can be quickly instantiated

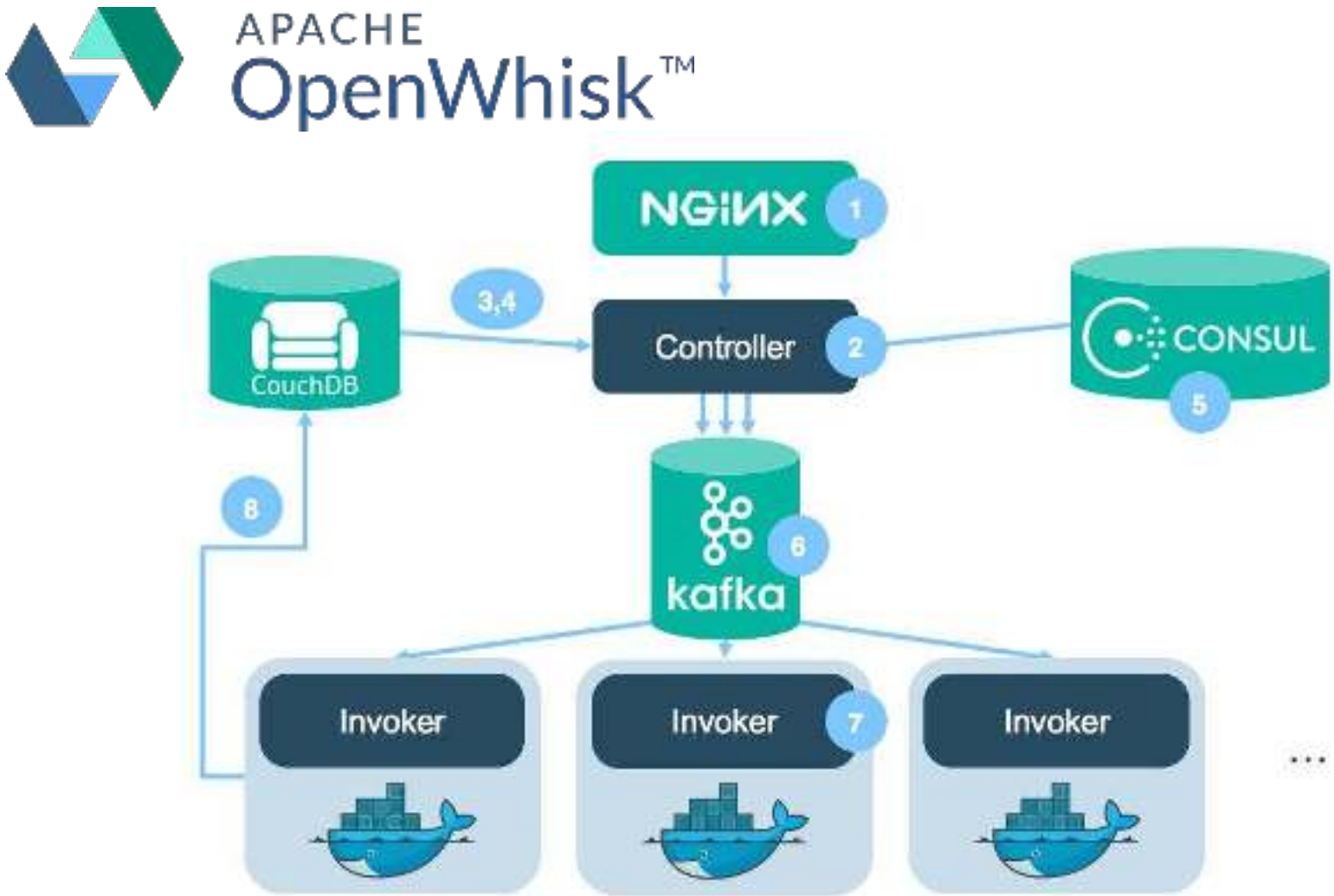
- No ‘Free Lunch’
  - Functions have to be *stateless* by definition

Background

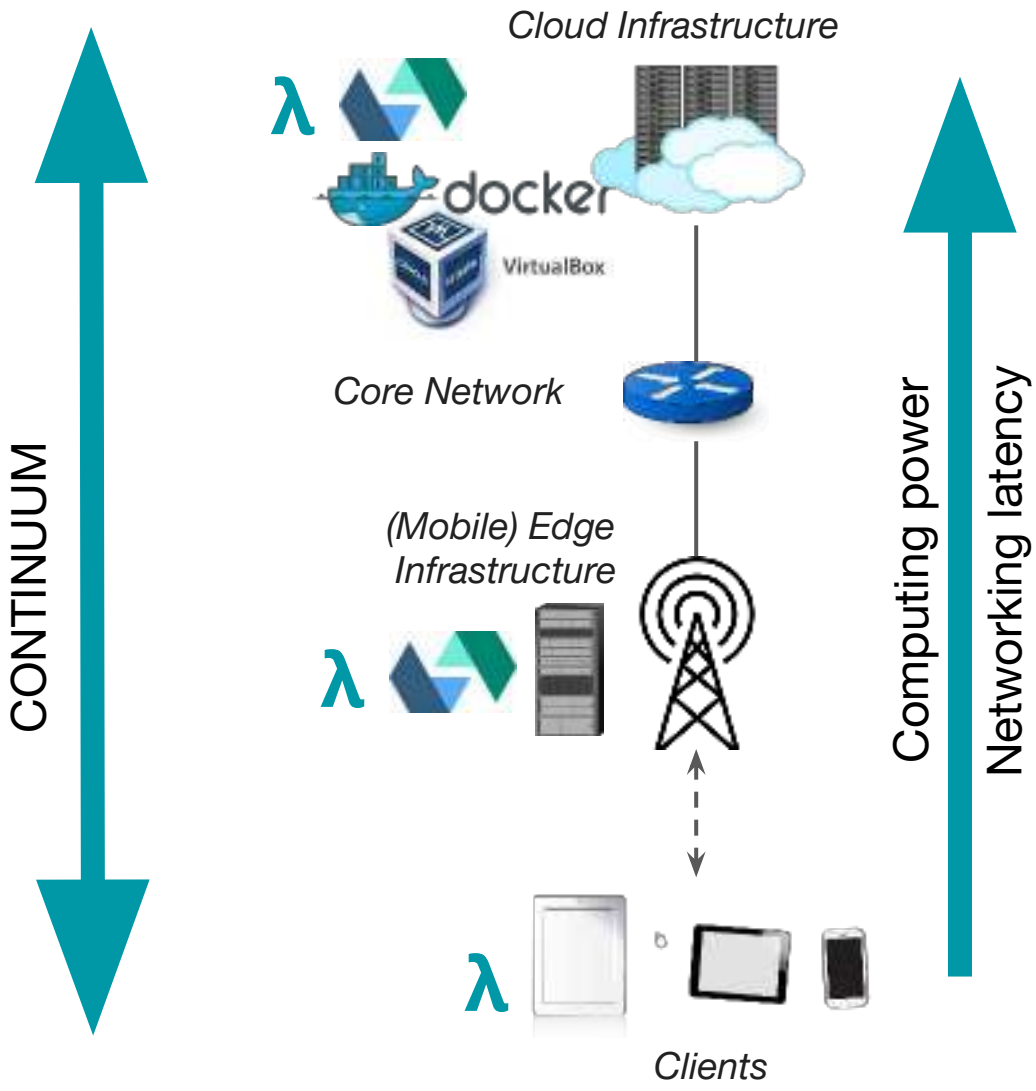
- Some FaaS vendors

Amazon	*Google*	*Microsoft*	*IBM/Apache*
			
Lambda Functions	*Cloud Functions*	*Azure Functions*	*Openwhisk Actions*

Background


APACHE
OpenWhisk™
NGINX
1
CouchDB
3,4
Controller
2
CONSUL
5
8
6
kafka
Invoker
Invoker
7
Invoker
...

Proposal



Cloud
- multiple hops away
- high network latency
- ‘unlimited’ resources
- virtualization and containerization

+ FaaS

Mobile Edge
- single or few hops away
- low network latency
- limited resources

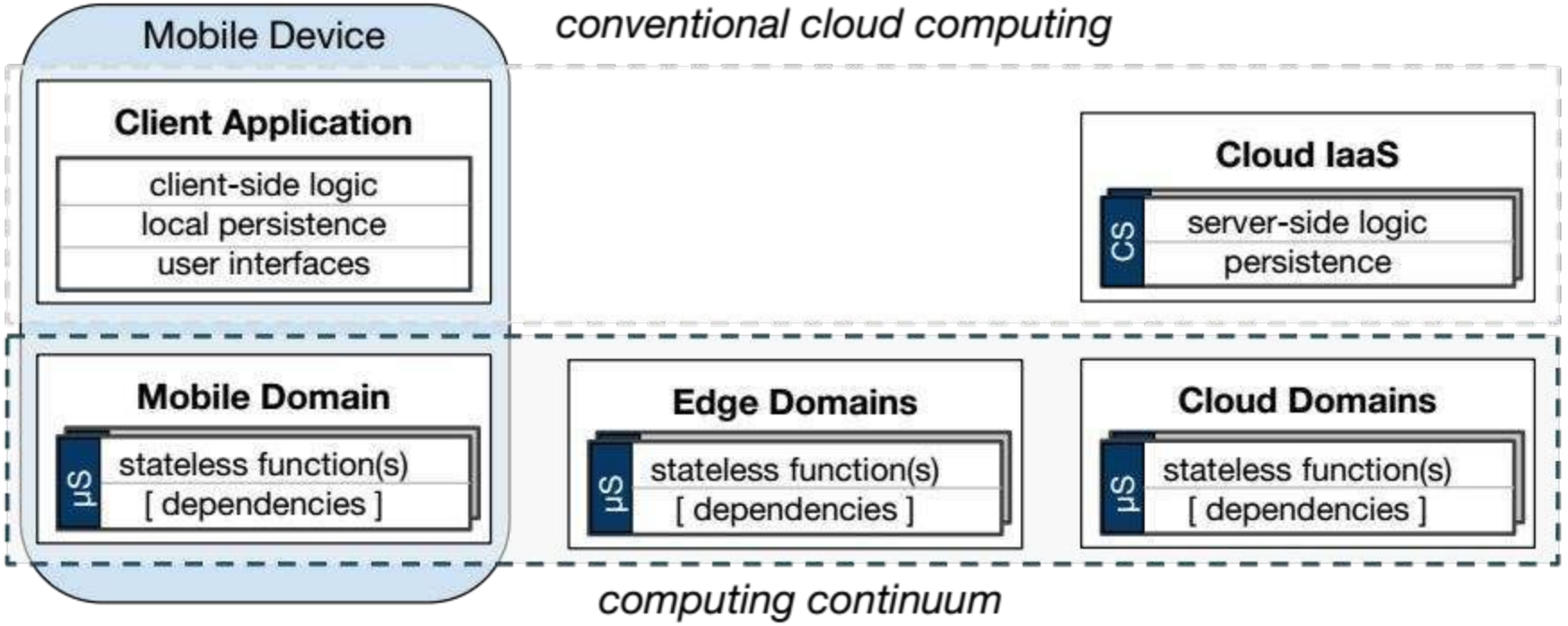
+ FaaS

Mobile ‘prosumers’
- constrained resources
- high processing latency
- battery limitation

+ FaaS

Proposal

Entering the Mobile-Edge-Cloud *Continuum*

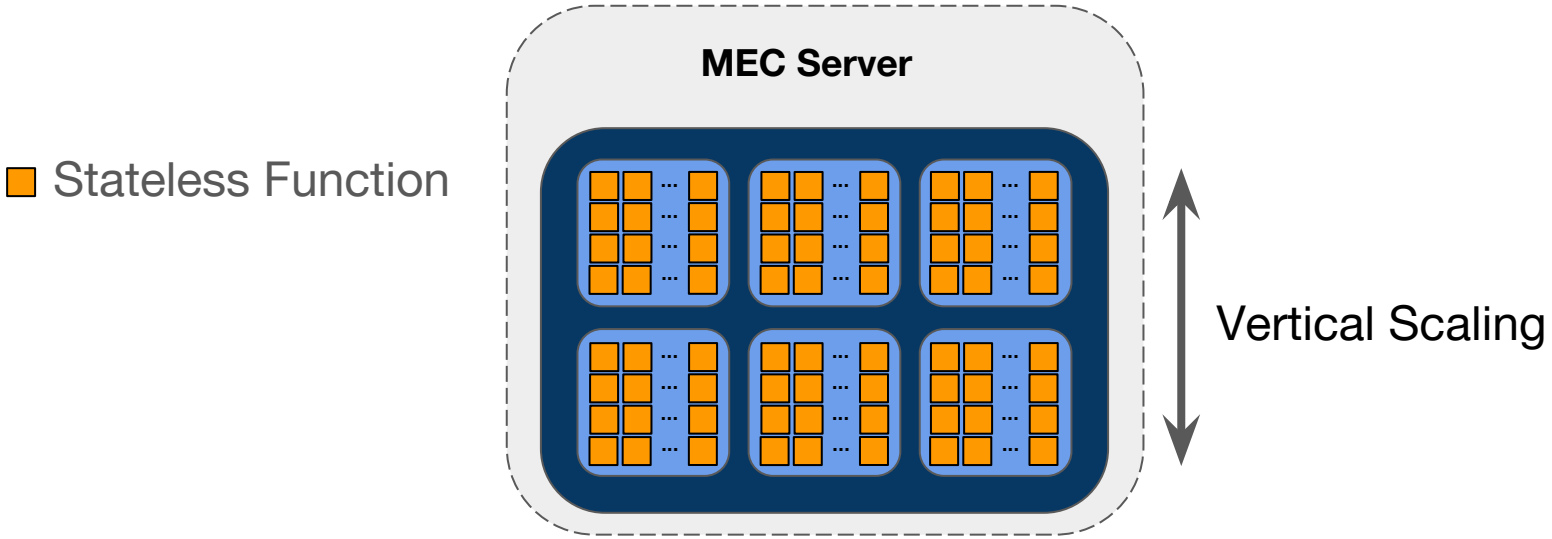


Applications are composed of:

Microservices (μS) provided by mobile, edge, and cloud domains

Conventional components that run on the mobile device + cloud services

Proposal



A FaaS architecture copes with the resource limitations of Edge infrastructure by allowing multiple instances of functions to scale vertically

Proposal

A3-E, a model for supporting efficient and scalable placement of microservices along the continuum

	AWARENESS	ACQUISITION	ALLOCATION	ENGAGEMENT
DOMAIN MANAGER	**Advertisement** ← domain identification **Discovery** → APP identification ← µ-service identified ← client arrived ← client left	**Identification** → µ-service identified **Download & Installation** ← µ-service acquired ← µ-service denied	**Self-management loop** → µ-service acquired ← µ-service allocated ← µ-service deallocated → client arrived → client left	**Provisioning** → µ-service allocated → µ-service deallocated → µ-service request ← µ-service response
MOBILE MIDDLEARE	**Discovery** → domain identification ← domain found ← domain lost **Advertisement** ← client identification	**Identification** → domain found → domain lost → µ-service acquired → µ-service denied ← domain confirmed ← domain denied	**Self-management loop** → domain confirmed → domain denied → µ-service allocated → µ-service deallocated ← domain changed	**Invocation** → domain changed → C-request arrived ← µ-service request → µ-service response ← C-request reply

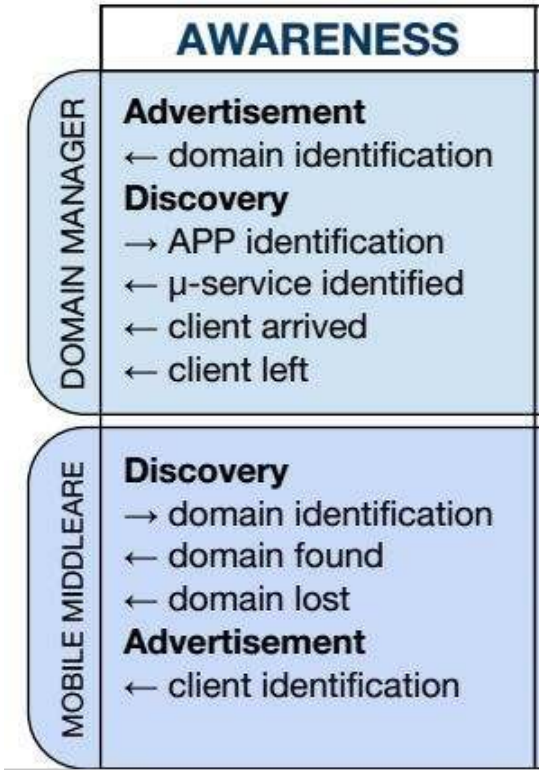
Satisfaction of application requirements (latency, battery, service availability)

Resource preservation (of mobile device and edge nodes)

Clients and domains take part in the automated and opportunistic placement decision

Proposal

Awareness

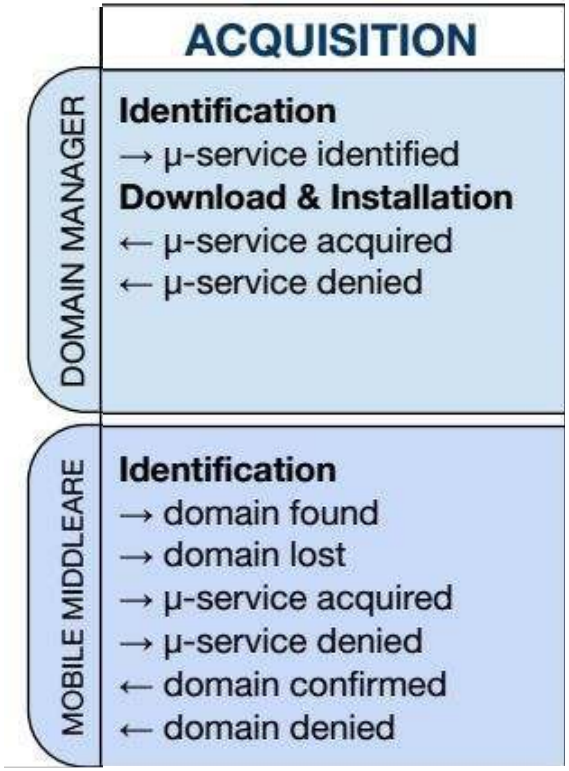


Mutual client/domain discovery

Allows to alleviate *cold-starts* typical in FaaS (by enabling the following activities…)

Proposal

Acquisition



Automated download and installation of microservices' artifacts

Eases OPS by means of a *pull*-based policy

So far, FaaS platforms offer a push-based one

Proposal

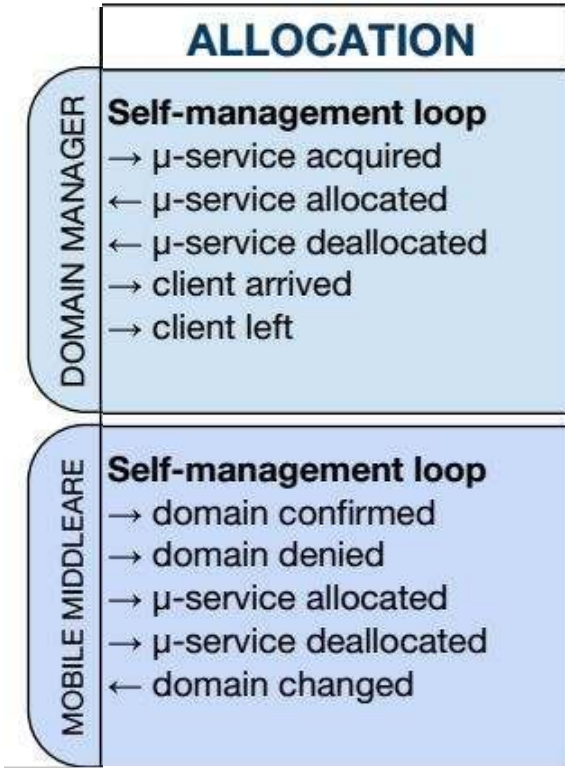
Allocation

Actual deployment of microservices to a certain domain

Existing FaaS solutions keep certain # of containers *warm* after a first request

A3-E features a self-management loop to approximate the amount of (pre)allocated resources:

- Service latency
- # of instances
- # of requests
- # of clients

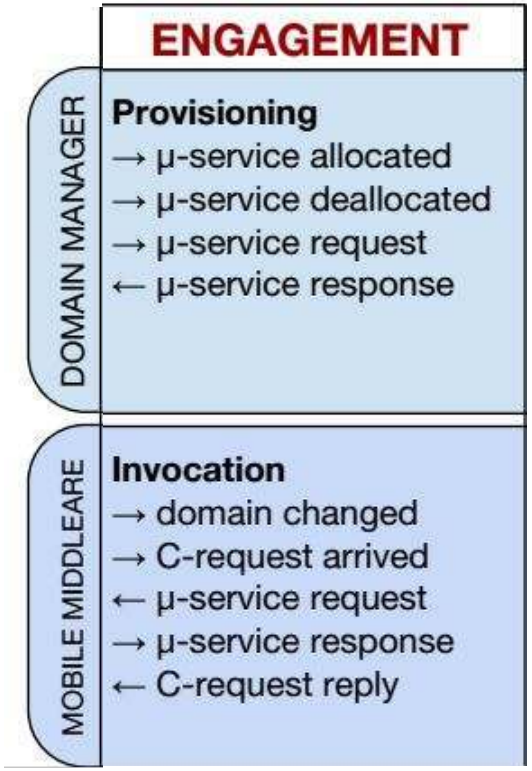


Proposal

Engagement

And finally… actual invocation of microservices!

As usual through RESTful interfaces to the functions



Evaluation -- Goals

- Latency and scalability of remote (cloud, edge) domains
- Client’s perspective: latency, battery
- Dynamic domain selection in the Continuum (mobile, cloud, edge)
- Deployment overhead (acquisition and allocation)

Evaluation -- Sample App

Augmented Reality Processing:

- Video capture -- **Features extraction**
- Rendering -- **Features matching**

Evaluation -- Simulated Architecture

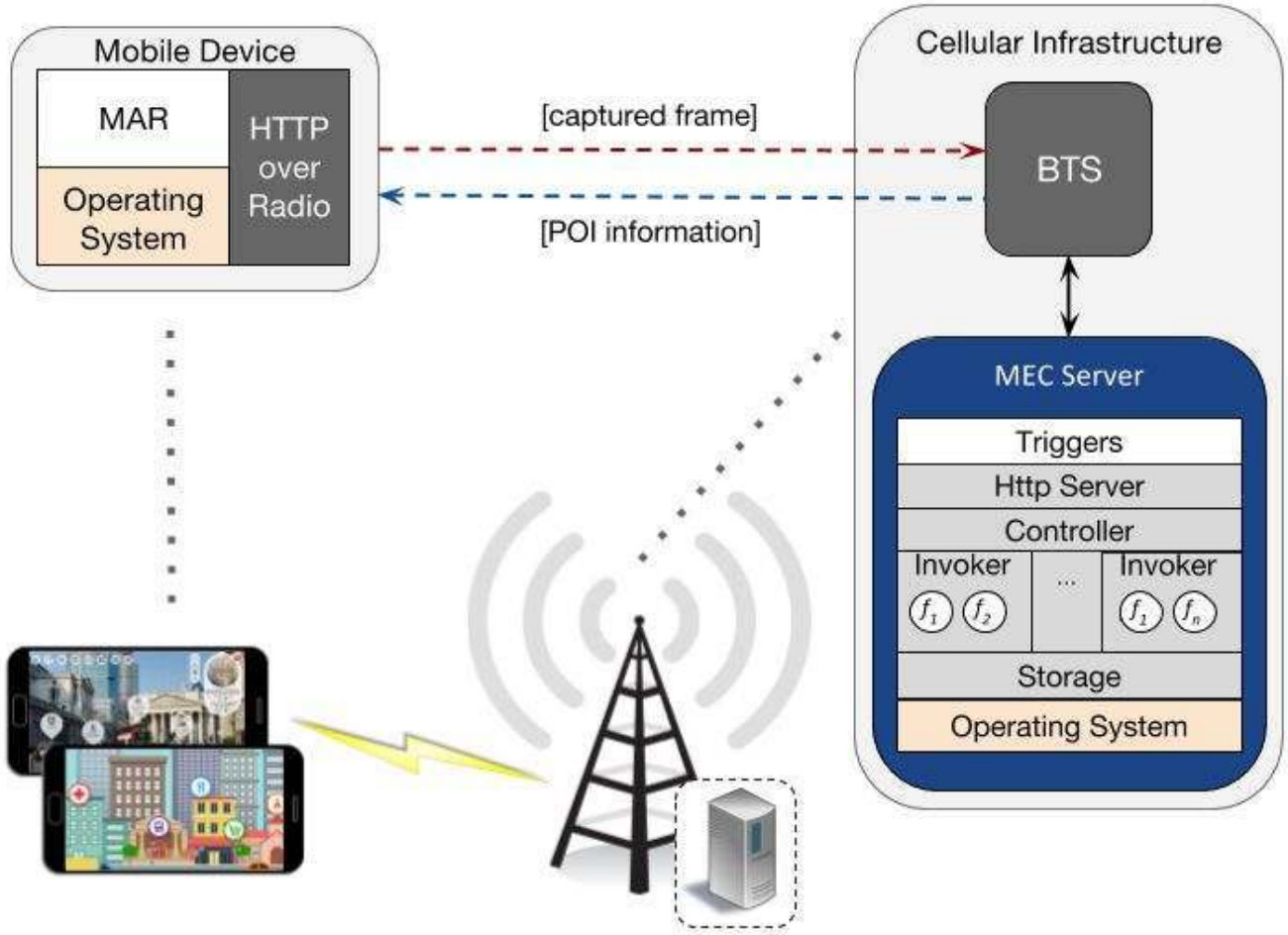


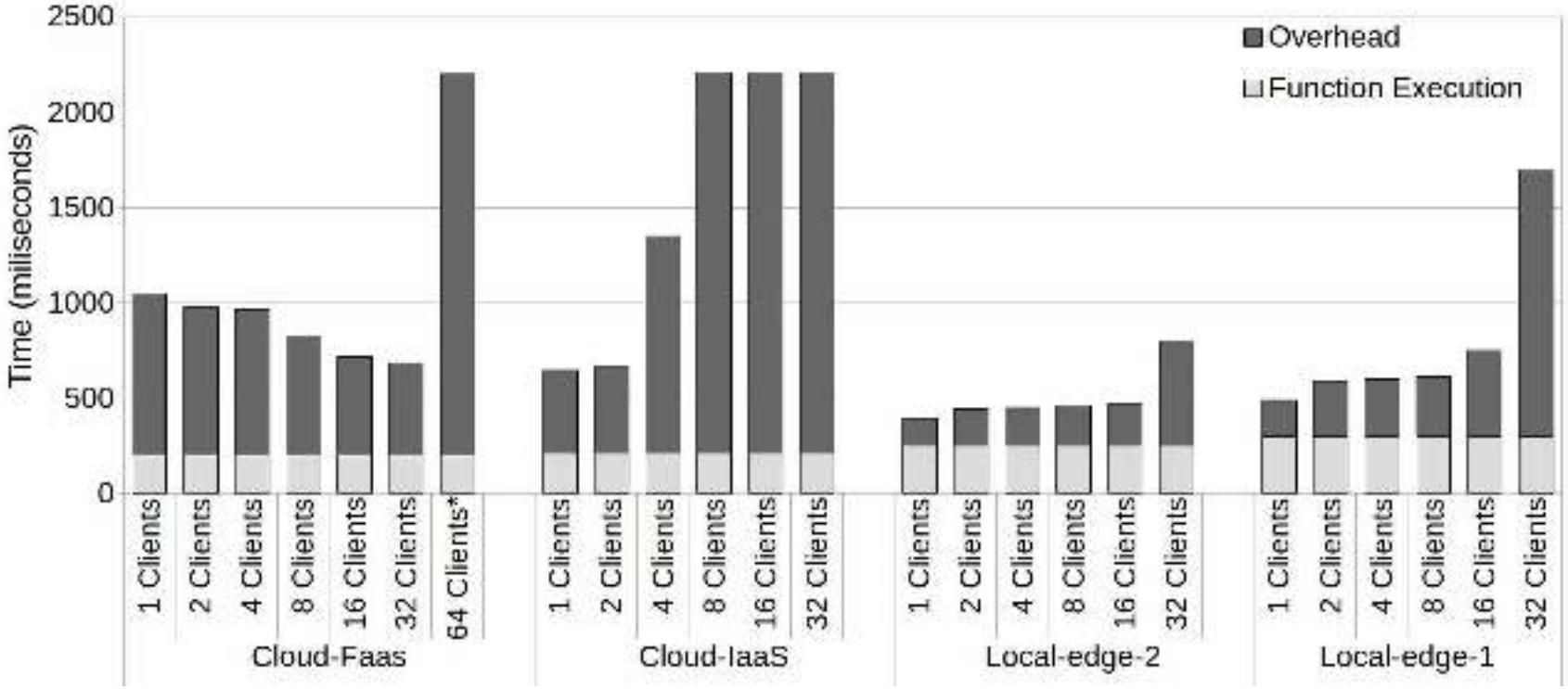
Fig: Feature extraction and matching are offloaded as functions to (Mobile) Edge servers hosting a serverless platform (Apache OpenWhisk)

Evaluation

Domain	Machine Resources	Execution Environment
Mobile	Samsung Galaxy S6 SM-G90, 3Gb RAM, 8x Cortex CPU 2Ghz	Android 5.0.2 + Java Functions + OpenCV
Local-edge-1	ubuntu/trusty64-2, 4x vCPUs, 4Gb RAM	OpenWhisk, 256 Mb/Action, Python 2.7 + OpenCV
Local-edge-2	ubuntu/trusty64-2, 8x vCPUs, 16Gb RAM	OpenWhisk, 256 Mb/Action, Python 2.7 + OpenCV
Cloud-FaaS	N/A	AWS Lambda, 256 Mb/Function, Python 2.7 + OpenCV
Cloud-IaaS	Auto Scaling Group with t2.micro instances + Amazon Linux AMI 2017	NodeJs 6.11 server + Python 2.7 + OpenCv

Results

Latency and scalability of remote domains



Latency per call for different workloads

Results

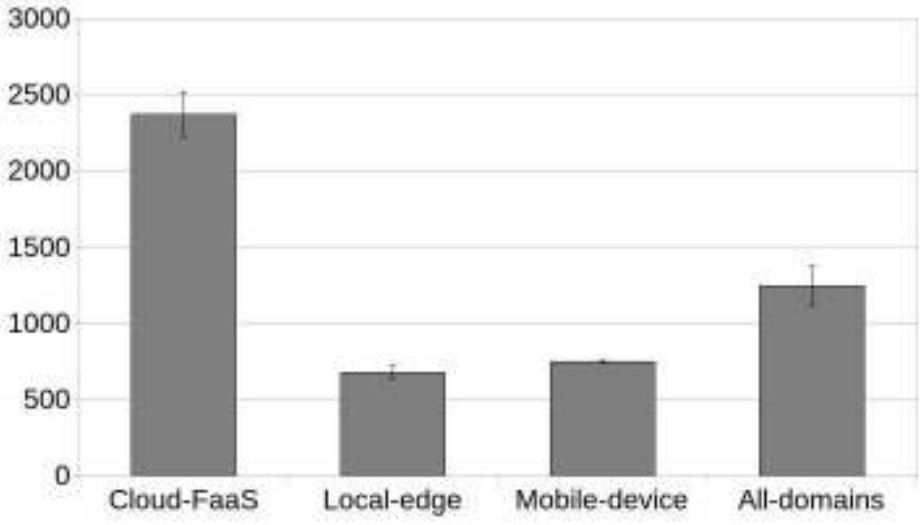
- The edge-based solution performed better than the cloud-FaaS platform for up to 16 simultaneous clients (32 reqs per second)
- AWS lambda diminished latency with more calls!
  - Less cold starts with higher stress levels of requests (higher reuse rates)
- A traditional Cloud-IaaS solution does not scale at the required rate
- The Edge latency could be lower in a Mobile Edge scenario where the edge server is located in the cellular infrastructure

Results

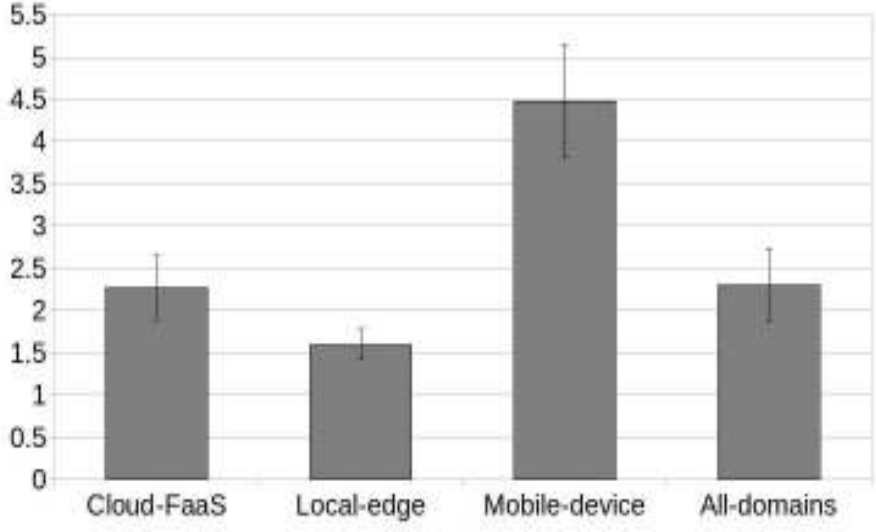
Client's perspective: latency, battery

Dynamic domain selection

2000 Sequential Requests



Total Execution Time



Battery Consumption

Results

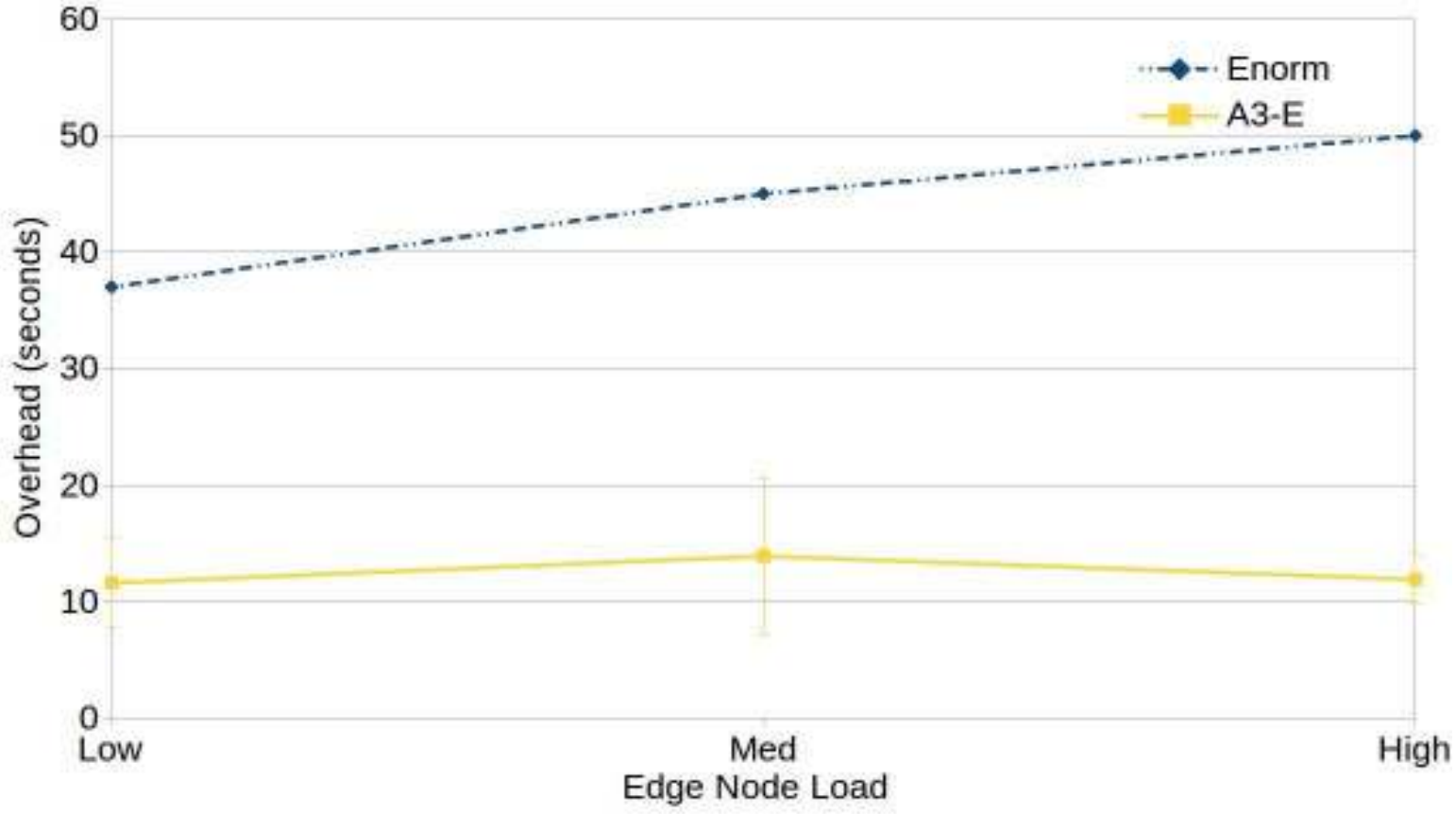
- Total execution time, when using only the cloud is two times higher than when using the continuum
- Using only the mobile lasted half the time, but used twice as much battery (20%/hour) than the continuum

Results

Deployment overhead (acquisition and allocation)

Completely cold start situation

Compared against state-of-the-art framework Enorm



Results

- Provisioning overhead of 12.5 seconds! (from nothing installed to response delivered)
- Reduces burden of downloading, installing and deploying: up to 70% less vs. a traditional container deployment

Final Remarks

- The Continuum is coming… with challenges!
  - heterogeneity
  - resource constraints @edge and mobile

- Serverless@edge can bridge both gaps at once!
  - Unified FaaS model
  - Optimizes use of limited resources
  - Improved elasticity, low reaction time

Future Work & Open Directions

- Resource management along the continuum
  - Low latency requirements
  - dynamic workloads

- Decentralized placement and coordination
  - Coordination among several edge servers and cloud
  - Device-to-device computation offloading

- Let’s see what 5G has to offer
  - Bandwidth and throughput
  - Fine-grained infrastructure...
  - Edge servers at each block!
- And Much More! (Security, Reliability, Error handling, Testing…)

Publications

Empowering Low-Latency Applications Through a Serverless Edge Computing Architecture
ESOCC'17 Conference
https://link.springer.com/chapter/10.1007/978-3-319-67262-5_15

A Unified Model for the Mobile-Edge-Cloud Continuum
ACM Transactions on Internet Technology (to appear)
Ping me for a copy!