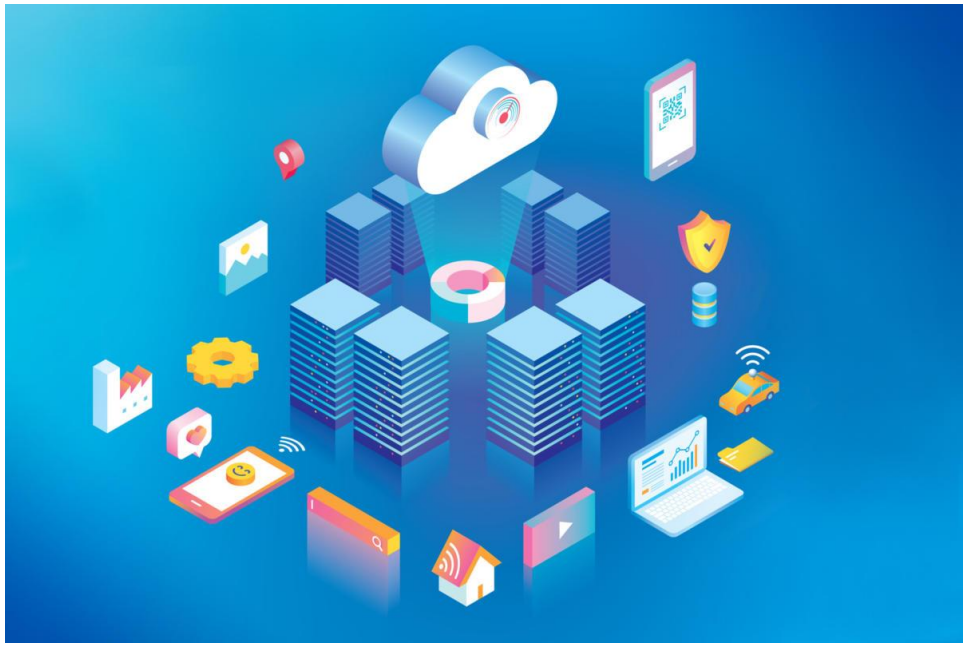
PAPS: A first step into the implementation phase. Partitioning a network.

Fabio Losavio, Oscar Francesco Pindaro

Table of contents

Context introduction

Edge Computing



"Edge computing is an open platform that integrates network, computing, storage, and application core capabilities on the edge of the network that is physically close to the data source."

Since the amount of data shared, is constantly growing, several different problems that have to be addressed:

- Bandwidth as a bottleneck.
- Privacy and data security in Smart-Home devices.
- Energy consumption in large-scale data centers will be difficult to meet in the future.

Context introduction

Serverless Computing

- Run the code without being concerned about operational issues.
- User are relieved from any involvement in the server provisioning and resource management. Developers can focus on the development of an application while ignoring the infrastructure on which the function will be deployed.
- Serverless applications are intended to be event-driven and stateless.
- The main system can dynamically allocate machine resources according to the specific needs of the final application.

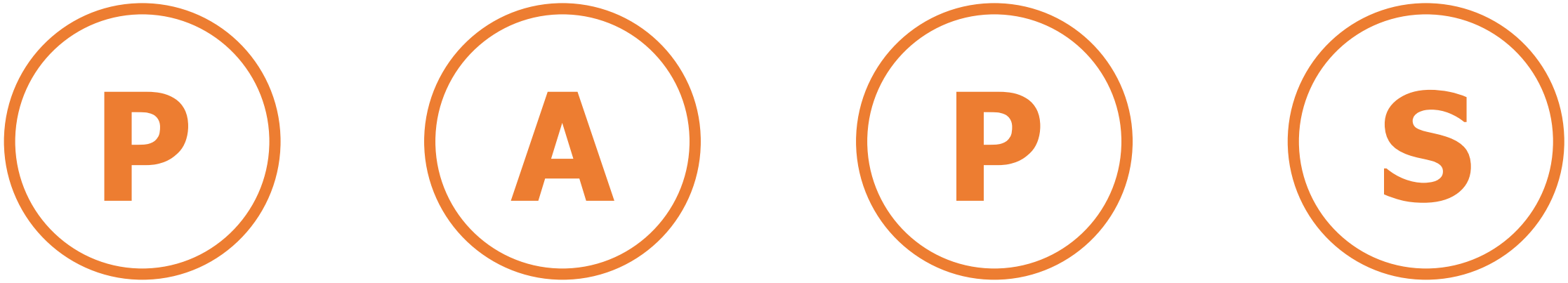
Context introduction

Containerization

- Easy cloud management
- Containers as self-contained, ready-to-deploy parts of applications
- Necessity of an orchestrator to manage multiple containers
- Allow dynamic resource allocation

PAPS: introduction to the framework

PAPS is a framework that tries to tackle the challenges related to the management of edge computing infrastructures automating the allocation of serverless function and the management of large-scale edge topologies.



Partition, divides the network in communities.

Allocation, allocates containters and manage workload changes.

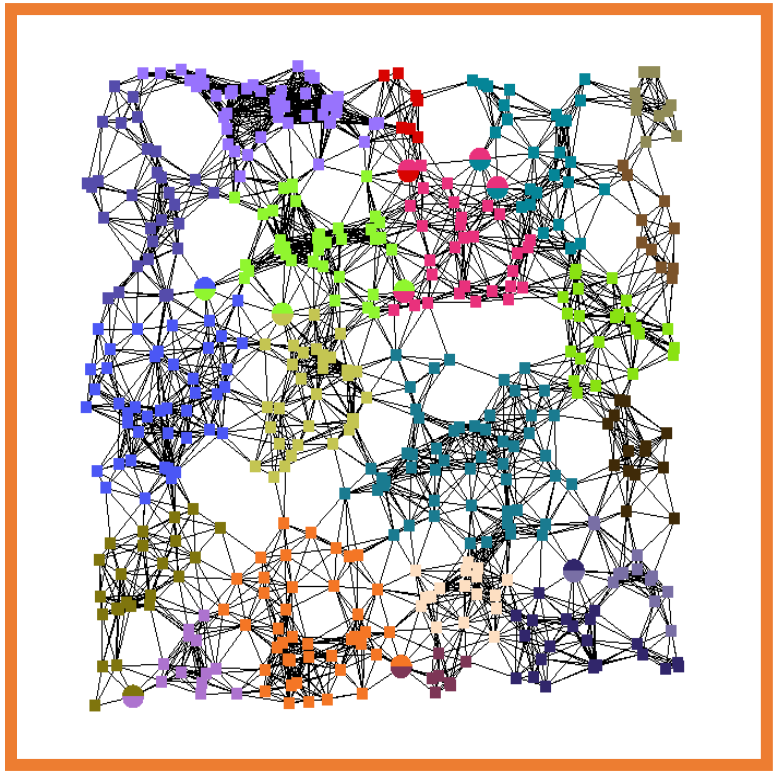
Placement, places containers on nodes.

Scaling, both horizontal and vertical scaling.

PAPS: System level

Split the network in delay-aware communities using SLPA, an algorithm which uses label spreading techniques to decide which community each node will belong.

- Variable number of iterations where nodes spread labels.
- Memory inside each node to store received labels.
- Listening and speaking rules to spread labels.

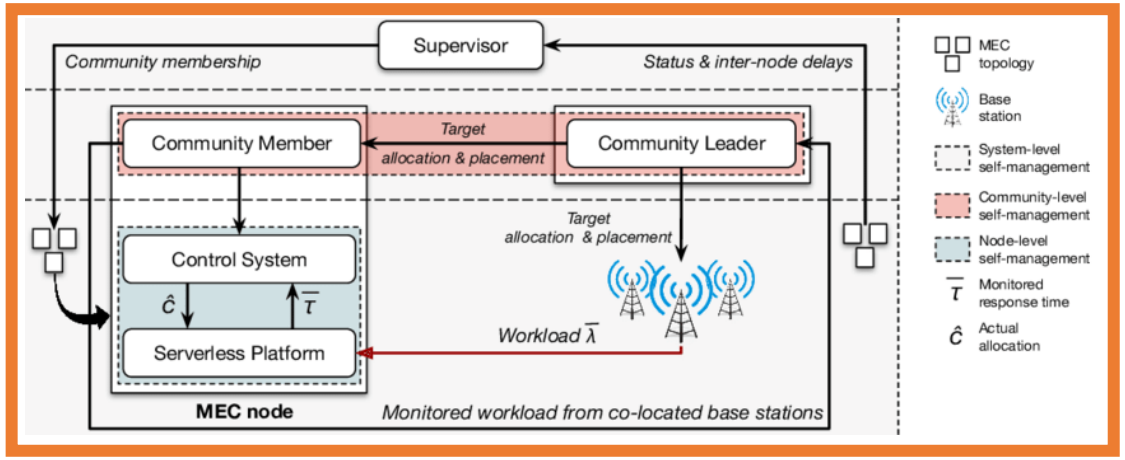


PAPS: Community level

- Minimize the likelihood of SLA violations so that the MEC nodes can operate under feasible conditions.
- Each community leader has to supervise its community by ensuring that every node it's still up and running.
- Supervise allocation and placement considering services demands and node capabilities
- Solve a mixed integer programming (MIP) problem to minimize the overall delay.

PAPS: Node level

- Keep the response time for a given function fixed
- Deploy and remove containers according to the measured workload.



Framework research

The goal was to find a framework to allow users to easily interact with the system also speeding up the development process:

OPENWHISK

- ✓ **PRO**: Wide language support, event reaction (Triggers), big community support.
- × **CON**: Complex internal structure, written in Scala and Akka.

OPENFAAS

- ✓ **PRO**: RESTful system interactions, integrated metrics tool, customizable scheduling policy, simple UI and CLI, simple internal structure.
- × **CON**: Hidden underlying infrastructure.

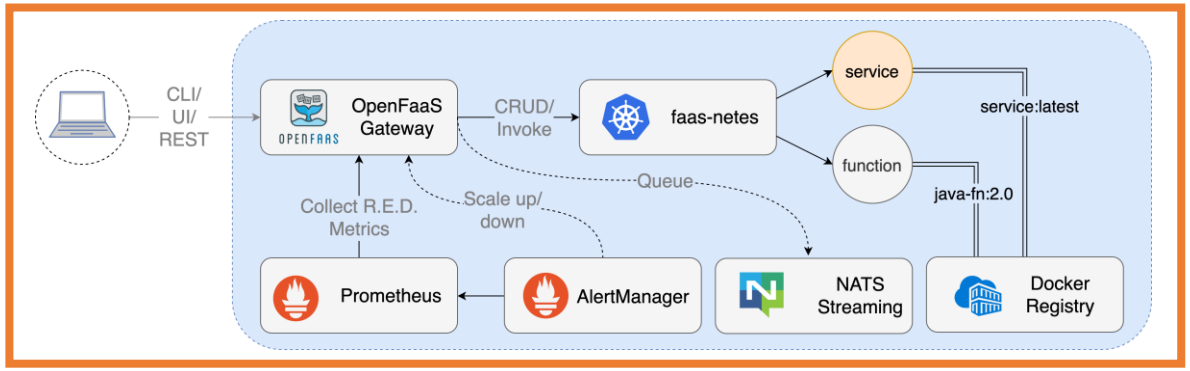
KUBELESS

- ✓ **PRO**: Directly extends Kubernetes API, RESTful system interactions.
- × **CON**: Written in Scala and Akka, lack of community support, stability problems.

OpenFaaS

- Delegates infrastructure management to IaaS.
- Simplify creation, management and deployment of functions inside the cluster

Internal structure



- **API-gateway:** expose REST-API, contact other components, invoked using CLI or UI.
- **WatchDog:** start and manage functions.
- **Alert Manager:** controls the traffic for each function, alert the API-gateway to trigger changes.

All those components are standalone containers.

Kubernetes

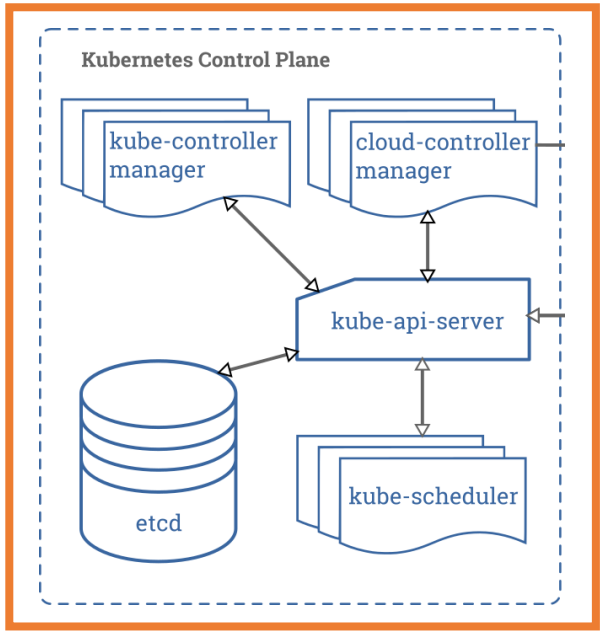
Automates deployment, scaling and management of containerized applications on a multi-node cluster. Containers are bundled in Pods.

- REST API to create, read, update and delete object.
- Add labels to the objects to mark them.

Kubernetes – Control plane components

Manage the cluster by making global decisions

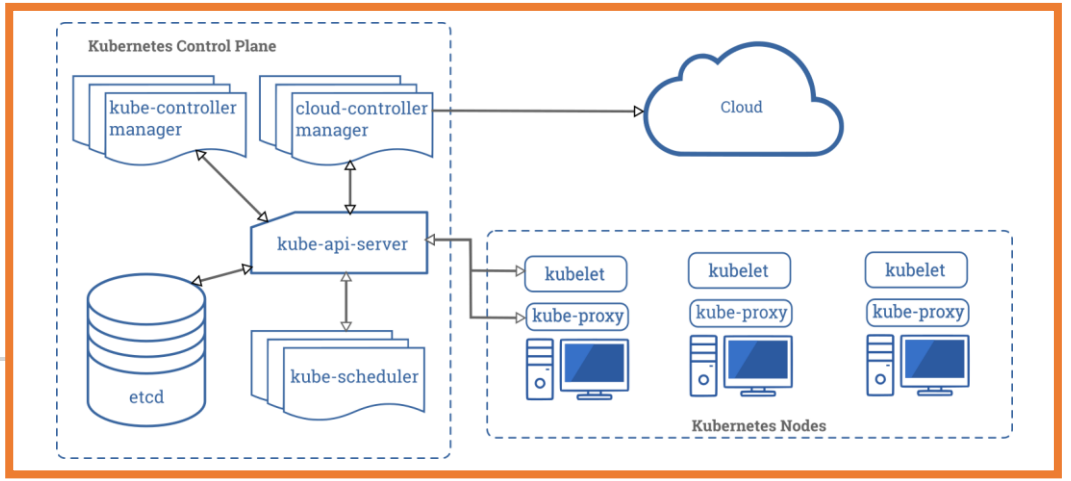
- kube-api-server: frontend module for the control plane, allows to create and manage resources.
- etcd: key-value based storage to save cluster data.
- kube-scheduler: places pods on nodes according to a scheduling rule.
- kube-controller-manager: checks nodes status and pods replicas.



Kubernetes – Kubernetes node components

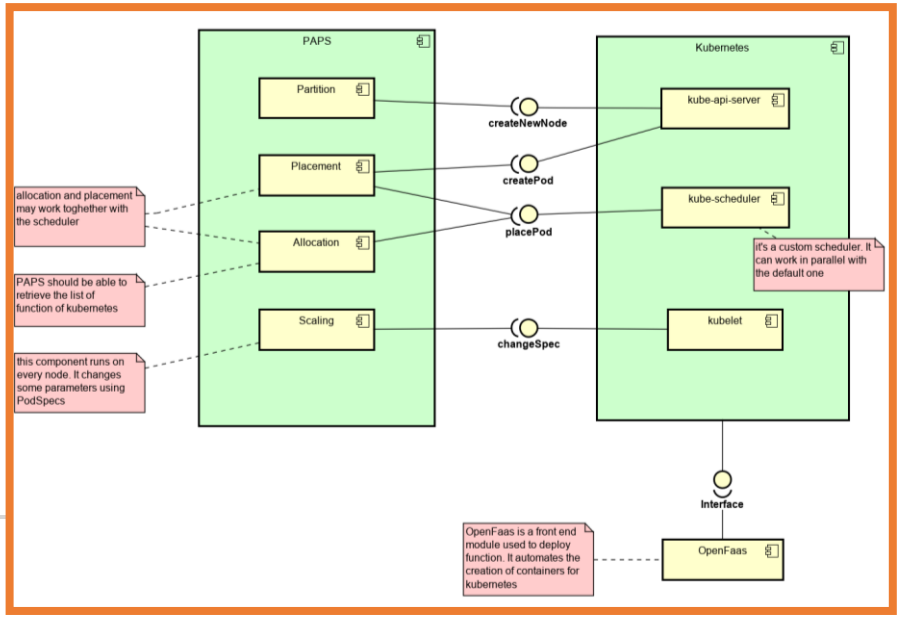
Placed on each node, handle the real creation and destruction of containers.

- kubelet: checks the status of Pods in the node according to a PodSpec.
- kube-proxy: connects the node with the api-server, maintain networks status allowing Pods to communicate.
- Container runtime: piece of software responsible for running containers.



General implementation idea

- **Partition:** divide the original topology in delay-aware communities using SLPA, input parameters can be passed using OpenFaas. Split oversized communities using Round Robin.
- **Allocation:** MIP solver in a dedicated container inside the community leader, access to community resources using labels. Determine how pods will be assigned to nodes.
- **Placement:** deploy pods on the actual node using the kube-scheduler.
- **Scaling:** use a PID controller, deployed as a pod on each node, to keep meeting the SLA



Partition Implementation

Parse the input delay matrix

Get the list of Kubernetes nodes involved (V1Node objects)

Wrap those nodes in an SLPA node

Run SLPA to populate the memory of each node

Post process the memory to assign a node to a community

Apply Round Robin to further divide big communities

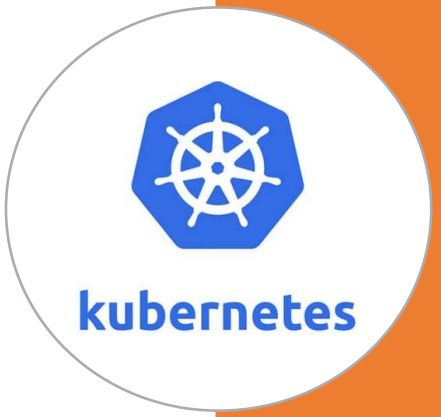
Update labels using Kubernetes API

Testing

Prototype developed using Java and REST API to connect with Kubernetes. Shipped as a container built using OpenFaas

Testing Kubernetes connection

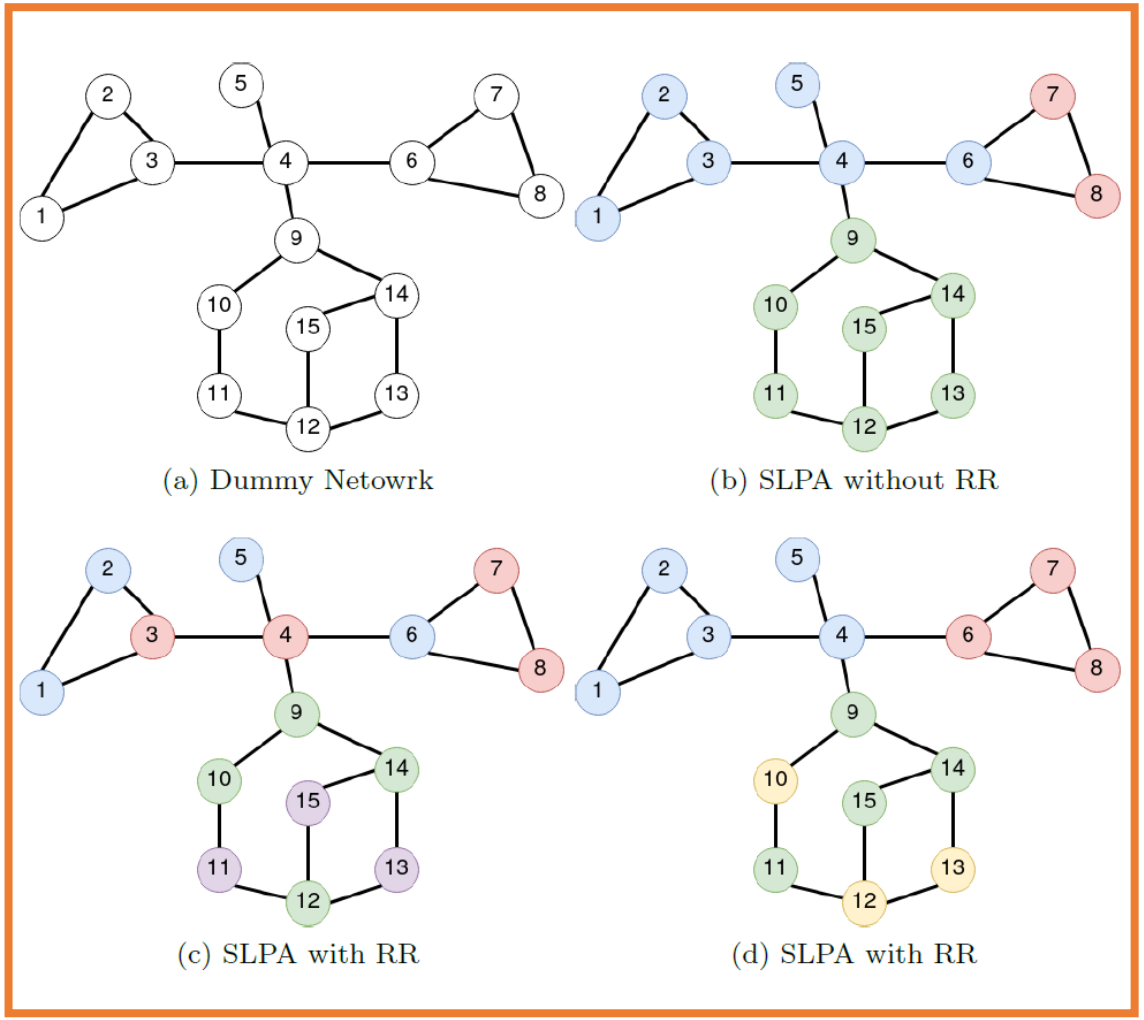
- Get access to kubernetes API (locally and on OpenFaas)
- Create a Role with access privileges
- Link the Role to kubernetes



Testing SLPA

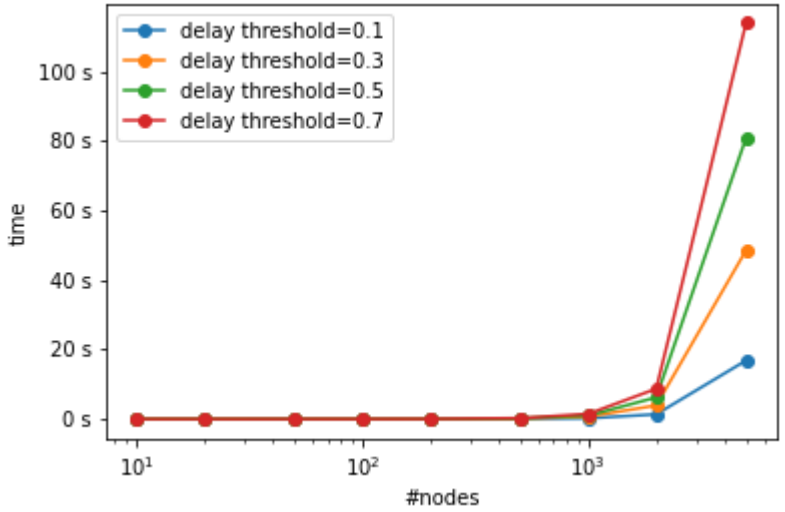
Tested locally to avoid the real presence of host in a network.

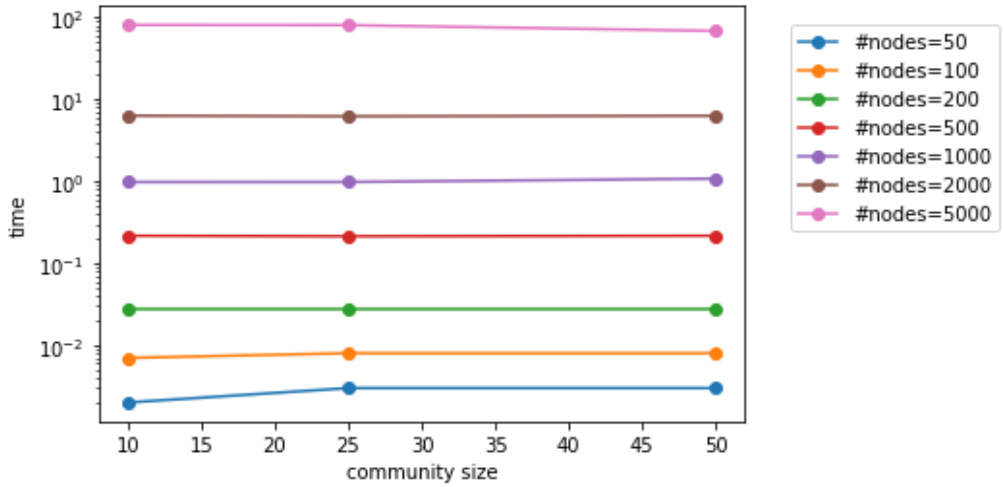
- Create an NxN dummy network and associate it to a delay matrix
- Eliminate edges whose delay is lower than a given threshold
- Run SLPA on the obtained graph (delays are no longer considered)



Testing SLPA Performances

- Randomly generated networks with different sizes.
- Cycle on different values of delay threshold and community size.
- Measure the execution delay.
- Plot the results.





Conclusions

- Work mainly focussed on researching frameworks and understanding a feasible implementation plan
- Implementation of the partition module implementing a custom SLPA version (GANXis can be an alternative)

Future work

Implement allocation and placement working with kubernetes scheduling

THANKS FOR THE ATTENTION