Apollo: Modular and Distributed Runtime System for Serverless Function Compositions on Cloud, Edge, and IoT Resources

Fedor Smirnov, Behnaz Pourmohseni, Thomas Fahringer

Presented by: Angela Mu, Xiaojie Li and Ruigeng Zhang
Team 6

Outline

Introduction

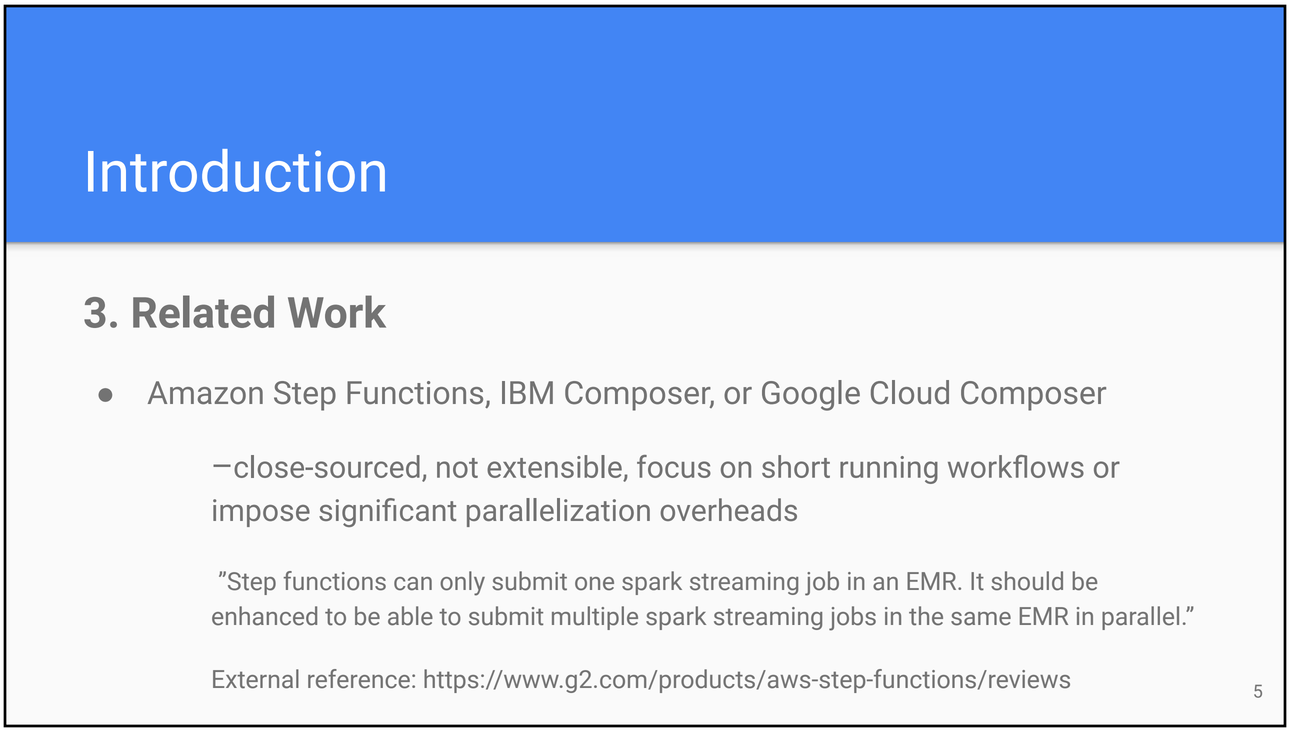
1. Overview

To fully exploit the potential for the optimization of the overall performance and costs of application implementations, Apollo, a runtime system for serverless function compositions distributed across the cloud-edge-IoT continuum is presented.

Introduction

2. Background

- FaaS offers many advantages of serverless computing and is particularly promising for usage in the context of ad hoc clouds consisting of mobile IoT devices.
- Yet the adoption of a loosely coupled application paradigm makes the implementation of applications significantly more complex.
- There is unprecedented potential for optimization, which can only be exploited by a sufficiently modular and flexible runtime system.



Introduction

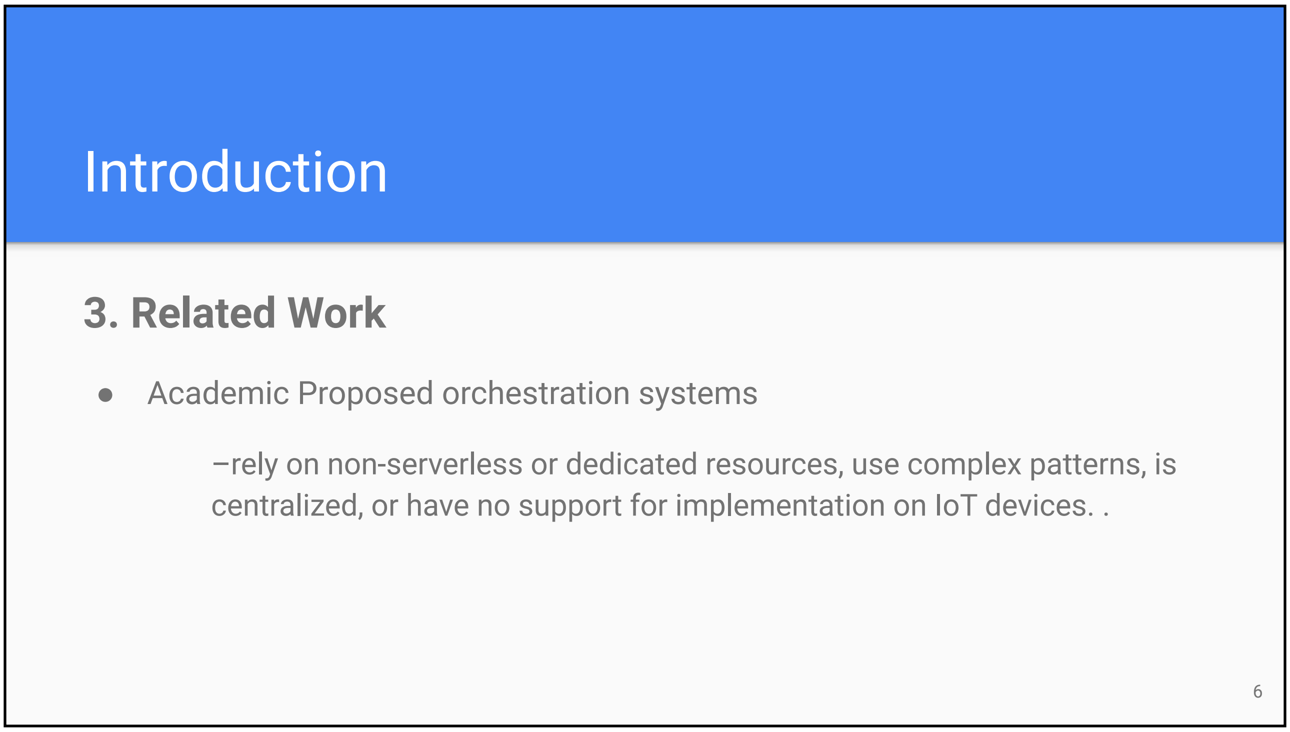
3. Related Work

- Amazon Step Functions, IBM Composer, or Google Cloud Composer

  –close-sourced, not extensible, focus on short running workflows or impose significant parallelization overheads

  ”Step functions can only submit one spark streaming job in an EMR. It should be enhanced to be able to submit multiple spark streaming jobs in the same EMR in parallel.”

  External reference: https://www.g2.com/products/aws-step-functions/reviews



Introduction

3. Related Work

- Academic Proposed orchestration systems

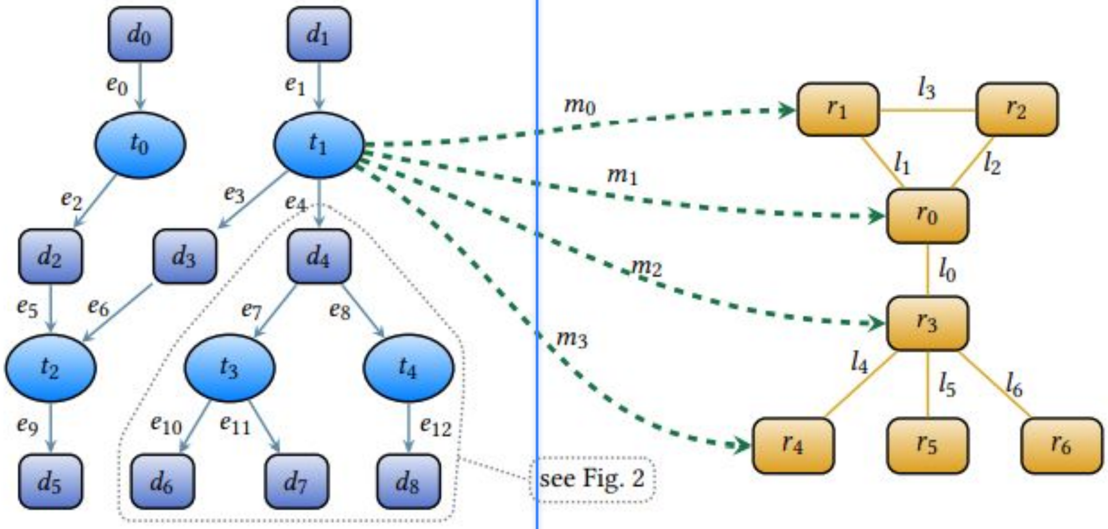
  –rely on non-serverless or dedicated resources, use complex patterns, is centralized, or have no support for implementation on IoT devices. .

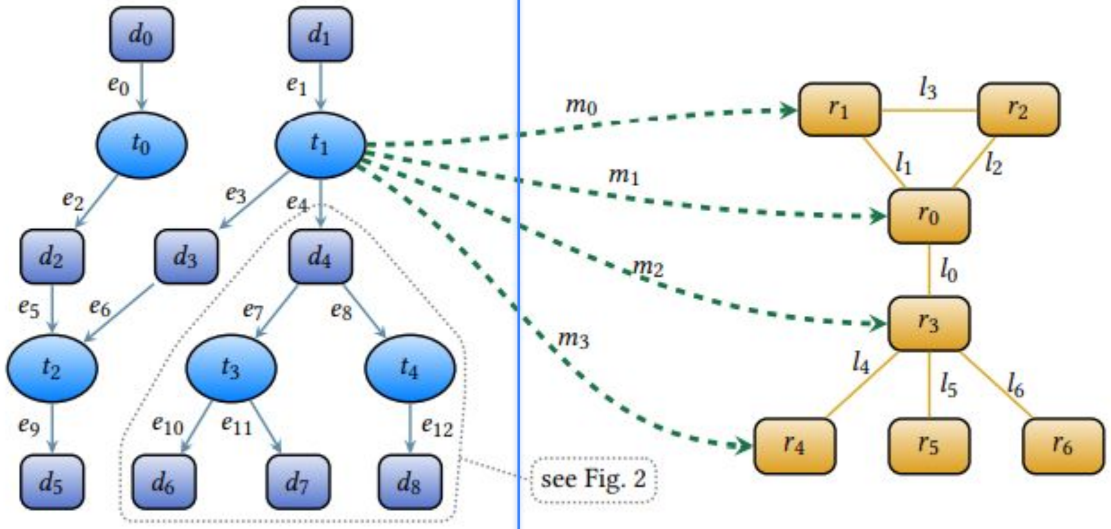
Introduction

4. A New Runtime System

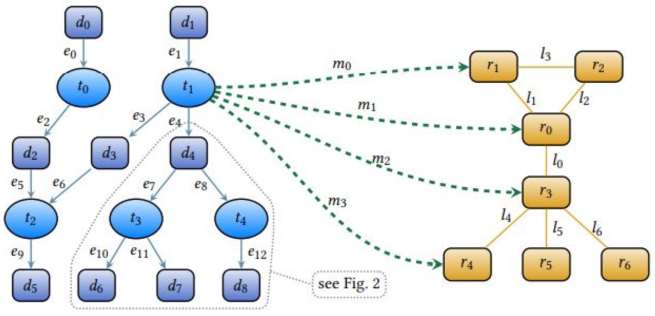
Apollo

developed specifically for the implementation of loosely coupled distributed applications based on FaaS services





Apollo models more complex data and control flow structures such as parallel, conditional, or cyclic execution of tasks by means of special task nodes and edges.



Introduction

5. Key Contributions

- improves the performance by enabling a full parallelization of the implementation process
- high modularity of the system
- Flexibility for the optimization of different implementation decisions to take advantage of available resources
- avoids vendor lock-in by supporting FaaS services of different providers and is open-source

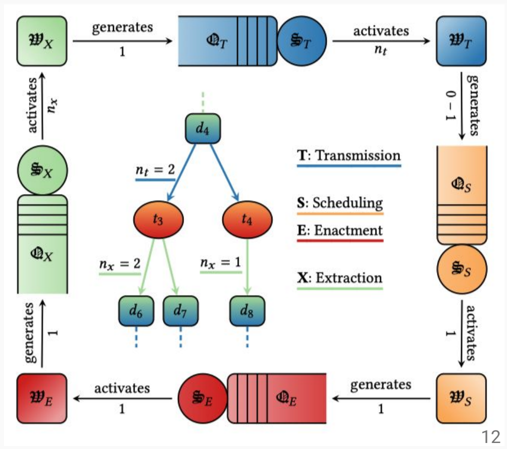
Implementation

The implementation process is realised by coordinating three types of **implementation agents**

- Main agent **M**
- Supervisor agents **S** ⟵ blocking queues **Q** which contain activation tokens
- Worker agents **W**

Four types of **activation tokens**

- Transmission **T** ⟶ blocking queue Q_T
- Scheduling **S** ⟶ blocking queue Q_S
- Enactment **E** ⟶ blocking queue Q_E
- Extraction **X** ⟶ blocking queue Q_X

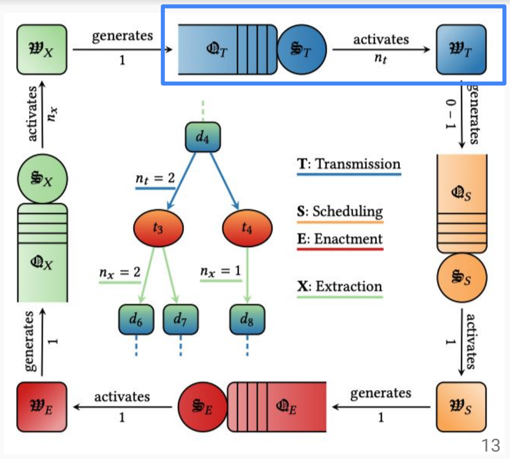


Implementation - Transmission

Transmission queue/token Q*T*
- Activation token associated with a data node in the enactment graph (a data payload for the subsequent tasks)

Transmission supervisor ST
- Activates one transmission worker for each successive task

Transmission worker W*T*
- Transmits the payload to consumer task
- In case that all data required by successor task is present, generates a scheduling token and places it into the scheduling queue Q*S* .

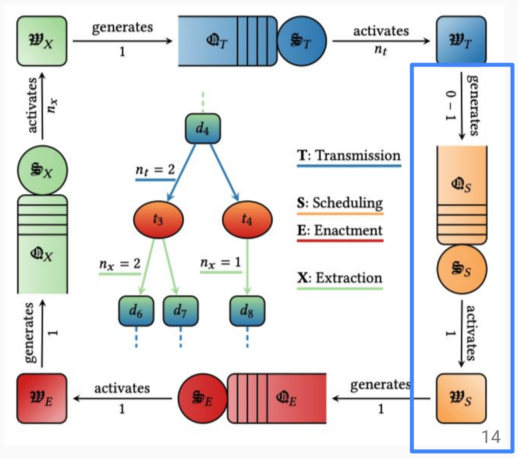


Implementation - Scheduling

Scheduling queue/token Qs
- Activation token associated with a data node in the enactment graph
- Represents a task whose input data is available.

Scheduling supervisor Ss
- Activates one scheduling worker for each task

Scheduling worker Ws
- Schedules the enactment of the task (chooses one or multiple resources from the set of mapping targets)
- Generates an activation token and places it into the enactment queue Q*E*

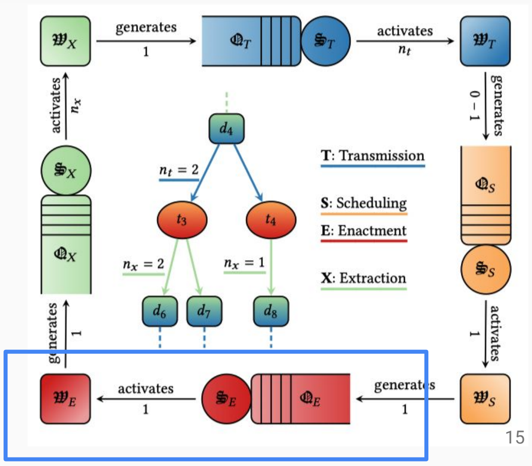


Implementation - Enactment

Enactment queue/token QE
- Activation token associated with a data node in the enactment graph
- Represents a task which is scheduled

Enactment supervisor SE
- Activates one enactment worker for each task

Enactment worker WE
- Triggers the enactment task on the binding targets chosen in the scheduling step, monitors the task execution, and reacts to potential errors
- Generates an activation token and places it into the extraction queue QX

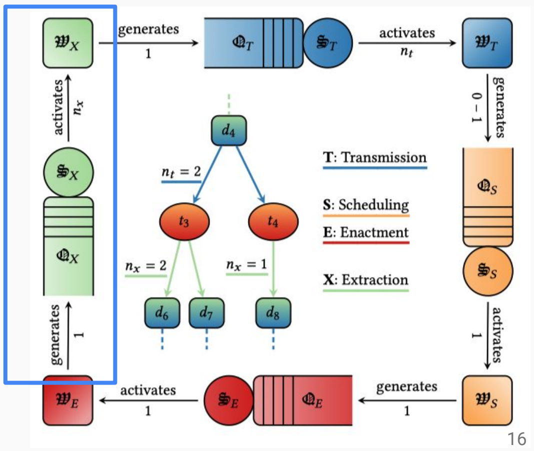


Implementation - Extraction

Extraction queue/token QX
- Activation token associated with a data node in the enactment graph
- Represents a task which was successfully enacted, so that its output data is available at its binding target(s).

Extraction supervisor SX
- Activates one extraction worker for each out-edge of the task

Extraction worker WX
- Makes the part of output data of the task available for further processing as part of input of the next task
- Generates an activation token and places it into the transmission queue QT



Implementation - Optimization

The system is optimised through a series of runtime implementation decisions

- Task Scheduling
- Data Transmission

Optimization method: Configure the behaviour of each agent

Transmission

- transmission methods (e.g., decide between transmitting the data as the request body or uploading it to an S3 bucket)
- transmission destinations (e.g., transmit the data to every mapping target of the task or decide to transmit the data to merely a subset of the mapping targets)

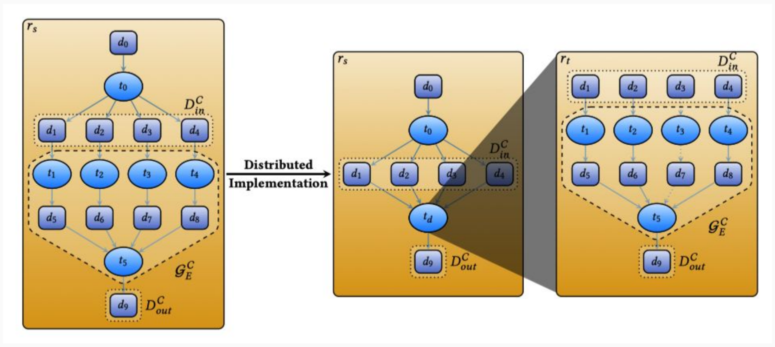
Extraction

- Transmission decision (e.g. transmits the data from the resource that carrying out the computation to the host machine or annotates the data node to indicate that the data is available).

Implementation - Distributed system

- Parts of the implementation process are offloaded from one resource to another resource

- Definition of cut C: two sets of severed edges, i.e. the cut-input edges Ein and the cut-output edges $Eout$.



Conclusion

In this paper, we have presented Apollo, a novel runtime system for serverless function compositions distributed across the cloud-edge-IoT continuum.

We have described its system model, as well as its implementation and distribution process, highlighting Apollo's modular design and the scalability resulting from its high degree of parallelization and arbitrarily fine-grained distribution.

Critique: Strengths

Strengths:

(a) It enables moving implementation operations closer to the processing to optimize performance and costs by leveraging data locality, while at the same time ameliorating the downsides of centralized implementation schemes.

(b) It becomes possible to adjust the amount of compute power available for making the implementation decisions. Consequently, while the parts of the implementation process which can be managed by lightweight heuristics can be executed directly on edge/IoT devices, the more complex implementation decisions can be made in the cloud, where the abundant compute power can be used for powerful global optimizers.

Critique: Implementation

1. The description of Apollo's implementation process is different from the actual work. It only contains the comprehensive description of the core functionality. In the actual system, there are additional implementation steps (e.g., graph-transformation operations performed at runtime), each of which has an extra token queue and an extra supervisor agent which creates specific workers.
2. The actual interaction between the implementation steps is more dynamic. For instance, an enactment worker agent which encounters an error during the enactment will place the activation token into QS instead of QX to trigger a rescheduling of the corresponding task so that it can be re-executed (potentially on a different resource).

Future Works

Future works:

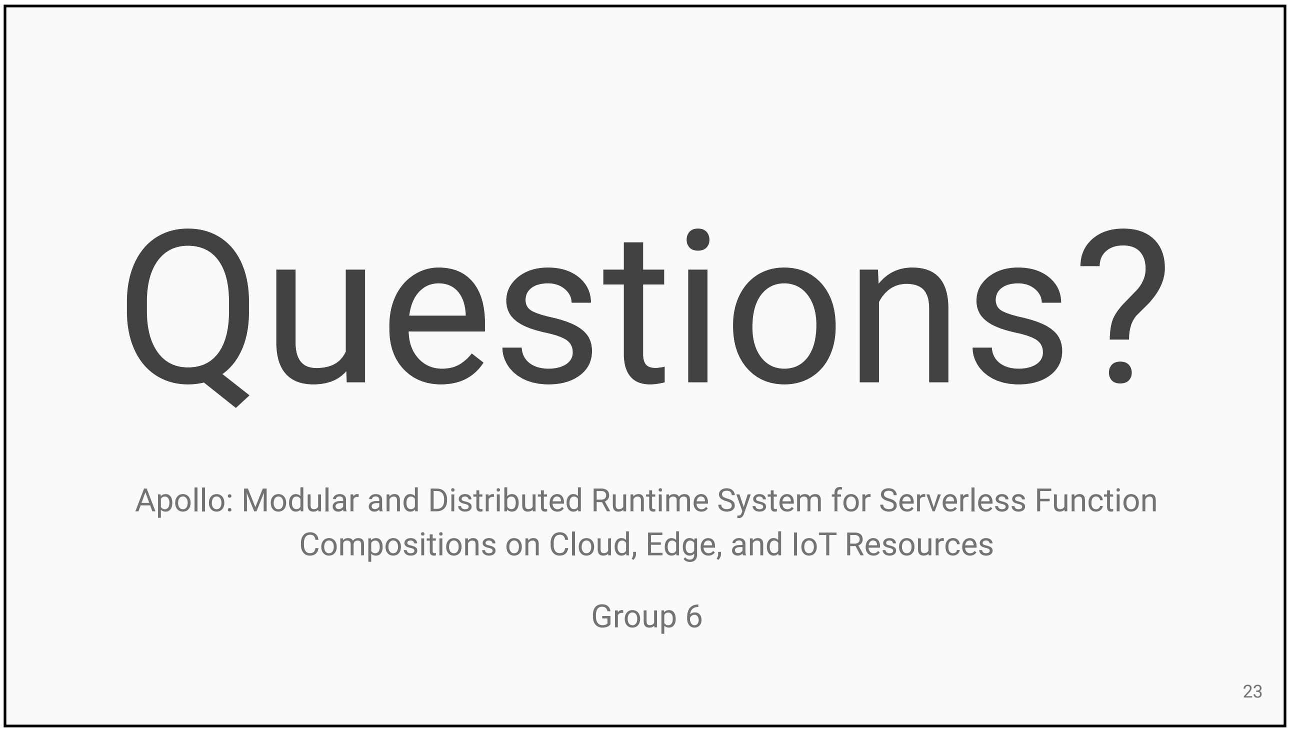
Develop optimization approaches for:

(a) task scheduling

(b) data transmission

(c) the optimal distribution of Apollo across the available resources, with a particular focus on a combination of global and local optimization approaches.

Lack of experiment and performance evaluation.（This appears to be in the long paper version published at the IEEE/ACM UCC conference 2021: https://dl-acm-org.offcampus.lib.washington.edu/doi/pdf/10.1145/3468737.3494103）



Questions?

Apollo: Modular and Distributed Runtime System for Serverless Function Compositions on Cloud, Edge, and IoT Resources

Group 6