Enabling QoS-Aware Task Execution on Distributed Node-RED Cluster for Fog Computing Environments

Project status 16th of august 2019

Project status: Object detection

- Improved object detection time for a single image (still using Tensorflow)
  - ~ 0.2 seconds on a MacBook Pro
  - ~ 2 seconds on a Raspberry Pi 4
- Optimized docker image size
  - before: 1.7 GB; now: 936 MB
- node-RED
  - Object detection server runs a REST API inside of a docker container
  - API is realized using Flask
  - A node-RED flow still has to be created which uses the flow and starts / stops the container
- Questions
  - Which image source should be send to the object detection server?
    - Local computer (Webcam via a Website)
    - Camera connected to Raspberry Pi
  - Where should the end result (detected image) be send to? What is done with the end result?

Project status: Sensor data

- Created a node-RED flow which generates and processes some random data
- Consists of three modules / flows
  - data-creator
    - Generates random sensor data.
    - Restriction: Can only be executed on a node which has a sensor connected (e.g. temperature), otherwise all modules would always be executed on the fastest node
  - data-processor
    - Processes the sensor data and outputs a result. The faster the CPU, the faster this process is done. Can be done on any node.
  - data-viewer
    - Views the result. Can be done on any node, but further restrictions possible.
- Communication
  - The output of one module (e.g. data-creator) is sent to the input of another module (e.g. data-processor) via HTTP

Project status: Fog orchestrator - MAPE-K

- Monitor / Knowledge base
    - nodes send heartbeats every 2 seconds via MQTT
    - if the node is not known to the orchestrator, some measurements will be performed
        - get system information about node (hostname, RAM, storage)
        - run CPU benchmark to determine the nodes’ CPU speed
        - measure ping and bandwidth to neighbor nodes
        - store the node in the **K**nowledge base
    - if the node is known, the heartbeat timestamp will be updated
    - if a timestamp is missing for 5 seconds, the orchestrator will check the availability of the node and remove it from **K** if it is not available anymore
    - network link quality is checked at a given interval for existing nodes

Project status: Fog orchestrator - MAPE-K

- Analyze / Plan
  - Initial idea: Use FogTorch
  - Faced issue: link bandwidth and CPU speed not taken into account
  - Solution: Implemented an own algorithm, inspired by concepts of FogTorch and iFogSim
  - Algorithm takes a given Infrastructure and Application as input (stored in **K**)
    - Application has several modules, each with different hardware requirements and costs
    - The end result of one iteration (initiating module to terminating module in the loop) should be available as fast as possible
    - The algorithm ("virtually") deploys modules on the nodes and calculates which deployment is optimal (fastest total time) by taking into account processing and transfer times
  - Algorithm output: Optimal deployment strategy
    - Which module should be deployed on which node?
    - Where should the result of an intermediate module be sent to?
- *Side note: Constructing and implementing the algorithm was quite expensive and can not easily be described here. Describing the algorithm in the thesis will easily fill some pages*

Project status: Fog orchestrator - MAPE-K

- Execute
  - Output of **A** is the optimal deployment strategy
  - To execute this, the node-RED flows running on the fog nodes must be updated
  - Executor reads flows (“application modules”) from a database and deploys them the to the right nodes using the node-RED admin API
  - The executor manipulates each flow in such way that the output is sent to the correct next module (HTTP input on the same or different node).
- Result
  - Every iteration (AppLoop) produces a result at the end
  - Additionally, statistics containing execution and transfer times are created and send to the MQTT broker
  - Orchestrator reads these statistics to see if the result arrived in time, and if not, problems in the deployment can be spotted (e.g. one message transfer took longer than expected, meaning there are problems with a network link), so that the infrastructure can be updated, **A** can be rerun and eventually there is a new optimal deployment strategy

Next steps

1. Fog orchestrator
    - Finalize code
2. Object detection
    - Create a node-RED flow, have a “real” use-case
3. Write thesis
    - Taking the proposal as a basis for the scientific background
    - Describe the implementation, encountered problems and solutions
        - Resource allocation / scheduling algorithm
        - Implementation of object detection server
    - Describe the result
        - Questions
            - What is the end result? What should be observed?
            - When is the orchestrator or result “successful”?