FogGuru: a Fog Computing Platform Based on Apache Flink

Davaadorj Battulga, Daniele Miorandi, Cédric Tedeschi

FogGuru: a Fog Computing platform based on Apache Flink

Davaadorj Battulga
U-Hopper, Univ Rennes, Inria, CNRS, IRISA
davaadorj.battulga@u-hopper.com

Daniele Miorandi
U-Hopper
daniele.miorandi@u-hopper.com

Cédric Tedeschi
Univ Rennes, Inria, CNRS, IRISA
cedric.tedeschi@inria.fr

***Abstract*—Fog computing infrastructure aims to offload computing resources from cloud providers by placing edge devices closer to end-users and/or data sources. Systems and methods for developing and deploying an application in Fog nodes are still in their infancy. In this work, we present FogGuru, a fog computing platform meant to facilitate application deployment in Fog environments. FogGuru builds upon the stream processing paradigm for fog applications design; its implementation makes use of the open-source Apache Flink stream processing engine. The demo will showcase a Fog-based traffic analytics application deployed using FogGuru and running on a fog node, composed of a cluster of five Raspberry Pis.**

***Index Terms*—Fog Computing, Edge Computing, Apache Flink**

I. Introduction

Fog Computing is an architectural approach for building computing infrastructure meant to best support Internet-of-Things (IoT) applications and services. The main idea behind fog computing is to enhance the performance of traditional cloud computing applications by deploying computing resources closer to end-users and data sources. Potential benefits include reduced latency, bandwidth optimization, privacy and security, lower energy consumption, and reduced dependency on cloud providers. According to the OpenFog reference architecture [1], the key technical challenges in the maturation of Fog Computing are scalability, autonomy, and programmability. Fog Computing could enable highly adaptive deployments of services, including support for programming at the software and hardware layers.

A challenge stands in the fact that Fog Computing platforms will include both computing resources at the edge and in more traditional Clouds. Operating a Fog which gathers heterogeneous, geo-distributed nodes has been only very recently addressed [2] and is still a largely open problem. This work focus on Fog nodes located at the edge of the network.

Re-tasking a fog node or cluster of nodes for accommodating operational dynamics addressing rapidly changing requirements can be automated. However, this is hindered by the lack of common models for programming fog nodes; as of today, most fog deployments require a lot of manual intervention and use solutions for specific use cases.

In this work, we aimed to design and develop a Fog platform, called FogGuru, for facilitating the development and deployment of Fog applications. Our approach is based upon the usage of a stream processing architecture for elaborating data from the IoT tier on the fog node (including operations such as aggregation and filtering). The prototype we built makes use of the real-time Stream Processing Engine (SPE) Apache Flink, is provided as an image ready to be deployed on resource-constrained devices. We further provide support for automating the service deployment procedure (on both single nodes as well as clusters) and demonstrate its usage on a traffic analysis scenario.

The rest of the paper is organized as follows. Section II presents the platform design and discusses some technology choices we made. Section III describes the prototype implementation. Section IV presents the actual demonstrator that will be showcased. Finally, conclusion and future work are presented in Section V.

II. Design and Technology

A. Hardware stack

In principle, any device with computing, storage, and network connectivity could act as a fog node [3]. Additionally, fog nodes should be able to be distributed geographically, to cope with different network types, to be cheap and easy to replicate. We chose to work with Raspberry Pi 3b+ (Quad-core 64-bit ARM processor, 1GB of RAM, 32GB of storage) single-board computers as standard devices. However, as such devices are rather limited in terms of processing capabilities (mostly related

to the little amount of RAM available), we decide to build a small cluster of five Raspberry Pi 3b+, and use such cluster as fog node hardware platform. In Figure 1 we present two images of the cluster, which we refer to as 'fridge'.

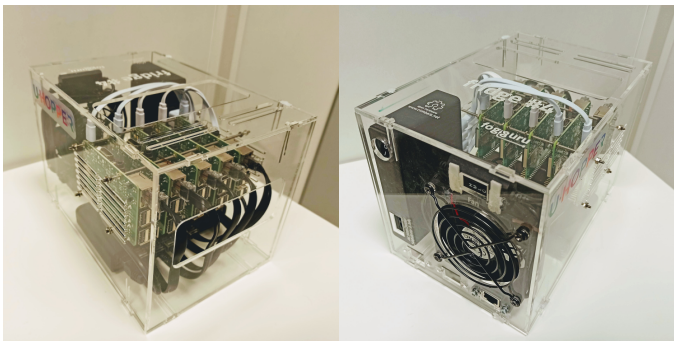

Fig. 1. The FogGuru hardware platform (cluster of 5 Raspberry Pi 3b+), commonly referred to as 'fridge'.

B. Platform architecture

The FogGuru platform high-level architecture is represented in Figure 2. It is consistent with standard architectures in edge/fog computing (see [4]) and in data processing pipelines [5]. Data from the IoT tier gets ingested through a suitable queueing system, from where it is fetched to be (stream-)processed. Intermediate results may be fed back to the message queueing system and/or stored persistently, depending on the expected usage. Processed data (represented as a stream) is pushed to the cloud tier for further aggregation/analysis. The operations of the stream processing engine are monitored, and relevant log data (or basic analytics) are also pushed to the cloud tier in batches.

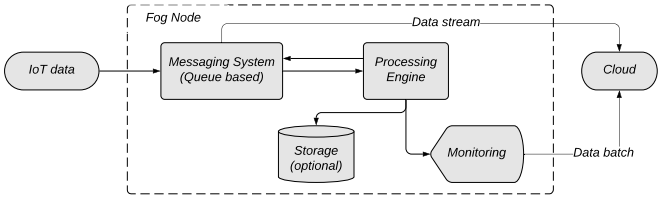

Fig. 2. The FogGuru platform architecture.

The choice of appropriate technologies/frameworks for implementing such an architecture plays a critical role. Here is a brief description of our design choices.

Messaging System: Message Queue Telemetry Transport (MQTT) is a lightweight publish-subscribe network protocol widely used in IoT applications, and better suited for fog applications than more powerful (but resource-hungry) cloud frameworks (such as, e.g., Apache Kafka). We decided to use the open-source Mosquitto[1] MQTT broker as our message handler.

Processing: Stream Processing Engines (SPEs) represent a good paradigm for Fog computing applications because of their extensive set of features, including support for event-driven, data pipeline, and data analytics applications. Constrained resources is also here an aspect to cater for; the literature includes surveys on how various SPEs perform in a fog environments [6], [7]. We decided to use Apache Flink[2] as our SPE of choice. Apache Flink is a distributed, open-source, stream processor with intuitive and expressive APIs to implement stateful stream processing applications.

Monitoring: We used Prometheus[3] and Grafana[4] for computing real-time performance indicators based on Apache Flink metrics APIs.

Containerization: Using container orchestration is mandatory for deploying fog applications at scale. It packages software components and their dependencies and deploys them in a standardized way. There are studies about performance evaluation of container orchestrators in Fog environment [8]. We used the Docker[5] ecosystem for its portable deployment.

III. Implementation

The resulting FogGuru deployment diagram is presented in Figure 3. It shows which components got deployed on which node, their roles and interfaces. To make it run, three main challenges had to be overcome.

Creating the swarm nodes: Docker Swarm can run on multiple resource-constrained devices such as Raspberry Pi. Using Docker CLI commands, it is possible to create cluster nodes. In our case, one node functions as a Swarm Manager node, others as Workers. The configuration deployed consists of 5 Raspberry Pis, interlinked locally with a docker swarm network.

Getting Apache Flink to run on Raspberry Pi: Apache Flink runtime consists of 2 types of processes: At least one Job Manager coordinates the distributed execution and Task Managers (workers) execute tasks and exchange data streams. The design is to run Job and Task managers on different swarm cluster nodes. To run Flink on Raspberry Pi, we configured the Job Manager heap-memory size to 512MB, and the Task Manager heap

[1]https://mosquitto.org/
[2]https://flink.apache.org/
[3]https://prometheus.io/
[4]https://grafana.com/
[5]https://www.docker.com/

memory size to 256MB. Also, we enabled Flink's built-in monitoring and metrics system to allow developers to monitor their Flink jobs. Flink Metrics were queried via REST API, which used with Prometheus and Grafana for monitoring and visualizing.

We used dockers buildx feature to create custom Apache Flink docker image for ARMv7 processor and hosted the image on DockerHub public repository[6].

Building the software: Docker Stack is included in the Docker Engine. We used Docker Stack to compose the services running on the cluster. Also, we designed the service placement: Which services should run on which nodes, how many replications, which services should connect using which network etc., and wrapped it in a YAML script for 1 click deployment[7]. The advantage of this method is that application developers can easily modify its internal services, and they only have to push a JAR file to deploy a fog application.

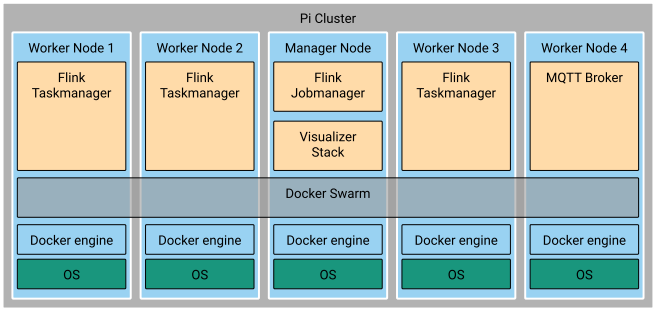

Fig. 3. FogGuru: deployment view.

IV. Demonstration

The demonstration setup is depicted in Figure 4. The demonstration mimics a use case in public transportation, whereby data on connected vehicles are collected and aggregated at some fog nodes (which estimate the traffic intensity in a given region). We use real-world data on connected vehicles, covering $245,369$ vehicles moving across Italy. Data consists of vehicle location, the timestamp of the event created, engine status and type of the vehicle. Events are created converting their recorded timestamp into real-time. On average, 0.8 events are generated per millisecond. The processing component implements the following functionality: It filters the incoming traffic by their recorded region and sends aggregated results every 10 seconds to the cloud. On the cloud tier, we deploy the Telegraf, InfuxDB and Grafana (TIG) stack to receive and visualize aggregated results. The dashboard is shown in Figure 5. The platform was able to tolerate the traffic without showing any back-pressure while maintaining available processing space for other tasks.

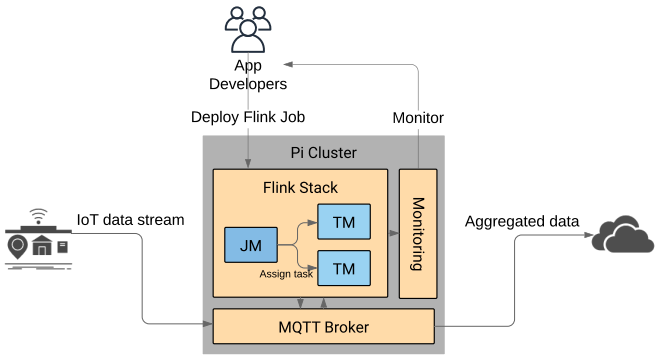

Fig. 4. Demonstration setup

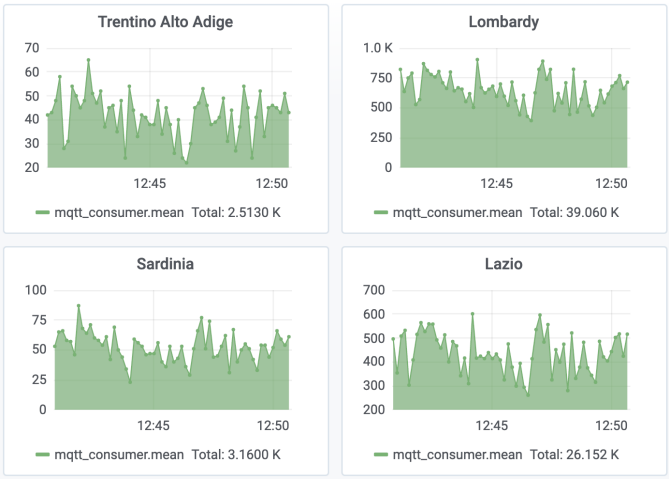

Fig. 5. Demonstration: the cloud dashboard for traffic monitoring at the regional level.

V. Conclusion and Future Work

In this work, we introduce and demonstrate FogGuru, a platform for developing and deploying fog applications and services based on the stream processing approach. FogGuru was designed to enable applications developers to program fog nodes easily; it is based on a set of open source components implementing open standards. We provide instructions for developers to quickly build and deploy FogGuru on single Raspberry Pis or a cluster thereof (using in the latter case Docker Swarm for orchestrating the deployment). The demonstration

[6]https://hub.docker.com/repository/docker/digitaljazz/flink-1.8.0-armv7

[7]https://flink-fog-cluster.readthedocs.io/en/latest

implements a traffic data analytics scenario based on real-world data from 245,369 Internet-connected vehicles. The practical experiments show that the FogGuru prototypical implementation can process in near-time fast data streams from IoT devices.

Future work includes the robust and fault-tolerant deployment of the platform, integrate autoscaling methods, further devise solutions to automate such deployments and their continuous optimization, and a more thorough evaluation of the performance attainable.

ACKNOWLEDGEMENTS

This work is part of the FogGuru project which has received funding from the European Unions Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant 765452. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

The authors thank G. Pierre and colleagues from INRIA in University of Rennes 1 for assembling the Raspberry Pi cluster.

REFERENCES

[1] IEEE Standards Association, "IEEE standard for adoption of OpenFog reference architecture for fog computing," 2018, https://standards.ieee.org/standard/1934-2018.html.

[2] P. Silva, A. Costan, and G. Antoniu, "Towards a methodology for benchmarking edge processing frameworks," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2019, pp. 904–907.

[3] Cisco. (2015) Fog computing and the internet of things: Extend the cloud to where the things are. https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.

[4] M. D. de Assuncao, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1–17, 2018.

[5] A. Ismail, H.-L. Truong, and W. Kastner, "Manufacturing process data analysis pipelines: a requirements analysis and survey," *Journal of Big Data*, vol. 6, no. 1, p. 1, 2019.

[6] H. Lee, J. Oh, K. Kim, and H. Yeon, "A data streaming performance evaluation using resource constrained edge device," in *Proceedings of the International Conference on Information and Communication Technology Convergence (ICTC)*, 2017.

[7] S. Zeuch, B. D. Monte, J. Karimov, C. Lutz, M. Renz, J. Traub, S. Breß, T. Rabl, and V. Markl, "Analyzing efficient stream processing on modern hardware," *Proceedings of the VLDB Endowment*, vol. 12, no. 5, 2019.

[8] S. Hoque, M. S. de Brito, A. Willner, O. Keil, and T. Magedanz, "Towards container orchestration in fog computing infrastructures," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2017, pp. 294–299.