FogFlow: Orchestrating IoT Services over Cloud and Edges

CHENG Bin, KOVACS Ernoe, KITAZAWA Atsushi, TERASAWA Kazuyuki, HADA Tooru, TAKEUCHI Mamoru

Abstract

Nowadays IoT infrastructure providers for smart city, smart industry, and connected vehicles are facing huge complexity and cost to manage their geo-distributed infrastructures for supporting various IoT services, especially those that require low latency. FogFlow is a distributed execution framework to dynamically orchestrate IoT services over cloud and edges, in order to reduce internal bandwidth consumption and offer low latency. By providing automated and optimized IoT service orchestration with high scalability and reliability, FogFlow helps infrastructure providers to largely reduce their operation cost. FogFlow also provides a data-centric programming model and a development tool chain for service developers and system integrators to quickly realize IoT services with low development cost and fast time-to-market. This has been proven in the labs as well as in real smart city projects done by NEC Solution Innovators.

Keywords

Internet of Things, edge computing, serverless computing, service orchestration, dynamic data flow

1. Introduction

The Internet-of-Things (IoT) enables a hype-connected society in which more and more things (e.g., connected cars, flying drones, and houses) are connected and become smart, meaning they are able to sense and react on real-time situations by utilizing data from different sensors and data sources. All of these smart things are empowered by their backend services, which usually consist of a set of sophisticated data processing logics. One of the big challenges to make constraint IoT devices smart is to easily, fast, and efficiently orchestrate their backend service logic so that they can react as fast as possible and utilize as much data as they can.

Unlike traditional big data analytics, the orchestration of IoT services must take into account the following issues:

(1) Data are constantly generated by sensors over time and it is not economically sustainable to send all raw data to the centralized cloud due to the high bandwidth cost and latency. More data processing must be dynamically offloaded to the edges close to data sources.

(2) Data and derived knowledge must be shared and exchanged across devices, services, applications, and platforms. Such a hyper-connected IoT system needs to manage thousands of linked IoT application executions in the same edge cloud environment with shared data resources.

(3) System workloads are more dynamic. Devices, services, and end user applications appear, move around, reconnect, and disappear. This leads to constantly changing workloads.

(4) Low latency and fast response time are required by many IoT services.

(5) The backend infrastructure needs to manage highly heterogeneous and geo-distributed resources of edge nodes at different layers.

All of these technical issues introduce new challenges and lots of complexity for infrastructure providers to manage their IoT services. FogFlow is designed to take care of all these complex issues for infrastructure operators and to help them automatically and efficiently manage various IoT services within a shared and geo-distributed environment.

As illustrated in **Fig. 1**, FogFlow provides a stan-

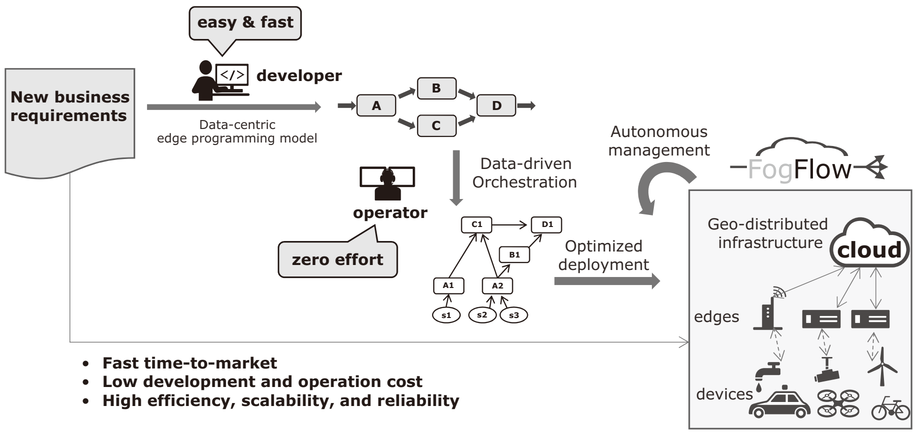

easy & fast
developer
New business requirements
Data-centric edge programming model
A
B
C
D
operator
zero effort
Data-driven Orchestration
C1
D1
B1
A1
A2
s1
s2
s3
Optimized deployment
Autonomous management
FogFlow
Geo-distributed infrastructure
cloud
edges
devices
• Fast time-to-market
• Low development and operation cost
• High efficiency, scalability, and reliability

Fig. 1 Value propositions of FogFlow.

dard-based and data-centric edge programming model for IoT service providers to easily and fast realize their services for various business demands. With its data-driven and optimized service orchestration mechanism, FogFlow helps infrastructure providers to automatically and efficiently manage thousands of cloud and edge nodes for city-scale IoT services to achieve optimized performance. In large scale IoT projects like Smart Cities or Smart Factories, FogFlow can therefore save development and operation cost, improve productivity, provide fast time-to-market, as well as increase scalability and stability.

2. FogFlow

In FogFlow, an IoT service is defined as a data processing flow represented by a graph of linked operators. An operator takes input from the IoT devices or from earlier parts of the processing flow. It performs the business logic of the IoT service and passes the intermediate result to the next operator. An operator is realized as a dockerized application and instantiated by FogFlow to run as a task within a dedicated docker container. Tasks are linked with each other during runtime based on the data dependency of their inputs and outputs.

As illustrated by **Fig. 2**, IoT services are orchestrated as dynamic data processing flows between producers (e.g., sensors) and consumers (e.g., actuators or applications) to perform necessary data processing. The following steps are designed to carry out the orchestration of an IoT service.

First, service developers create a service template to define the service logic, using the FogFlow Task Designer, which is a web-based graphical flow editor as shown in **Fig. 3**. The service template represents the abstract,

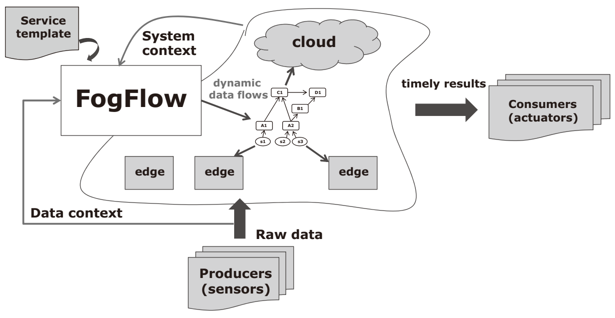
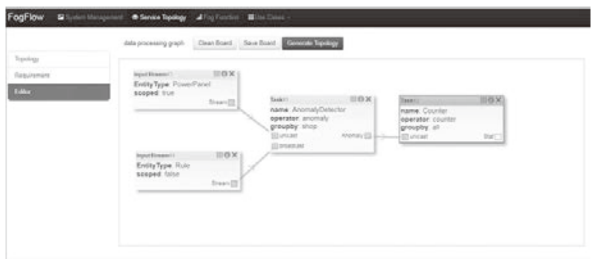


Fig. 2 High level model of FogFlow.



InputStream
Entity Type: PowerPanel
scoped: true
Stream

InputStream
Entity Type: Rule
scoped: false
Stream

Task
name: AnomalyDetector
operator: anomaly
groupby: shop
unicast
broadcast
Anomaly

Task
name: Counter
operator: counter
groupby: all
unicast
Stat

Fig. 3 FogFlow Task Designer.

static data processing logic of the IoT service, including the details on which operator is utilized to take which type of input for producing which type of output, and also when and how the operator should be triggered. Service providers can reuse the operators registered by others or implement their own operators.

Once the service template is submitted, FogFlow will monitor the context of available data in its runtime system to determine when and how the submitted service should be instantiated. For example, a flow can be started when a data item arrives at an IoT gateway. FogFlow determines how many task instances are required for each operator and also the detailed configurations of each task instance. The data structure of all data flows is described based on the same standardized data model called NGSI[1]. Therefore, FogFlow can learn which type of data is created at which edge node. It then triggers and launches dynamic data processing flows for each edge node based on the availability of registered context metadata.

Third, using some optimization algorithms, FogFlow decides where to deploy which task instance according to the real-time system context, including how much resources are available on each edge node, where the data sources are located, and how the predicted workload is changing over time.

As compared with other edge computing frameworks like Azure IoT Edge and Amazon Greengrass, FogFlow has the following unique features.

Data-centric Programming: In FogFlow we can ease the design and usage of IoT services by providing a data-centric programming model for different roles to express their goals at different levels in a more data-centric and intuitive way.

As shown in **Fig. 4**, at the operator level, operator providers just need to annotate which type of data their operators can handle, which functionality is provided, and which type of results are produced. At the service layer, service designers can easily compose different operators to form their service templates in just a few minutes. After that, during the runtime data processing, flows can be automatically orchestrated by FogFlow based on the high level data usage intent defined by service users.

Service users can be either data producers or result consumers. From the producer perspective, they can request which type of service logics to be applied to their data; while from the consumer perspective, they can request which type of results to be generated. What FogFlow does is to translate their high level data usage intents into concrete data processing flows and then deploy and maintain them seamlessly over cloud and edge nodes. More importantly, with this data-centric programming model and based on standard data model,

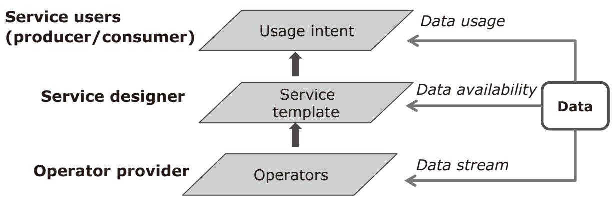


Fig. 4 Different roles in FogFlow.

the underlying data processing flows can be shared and optimized across multiple services and users. FogFlow provides a unique basis for infrastructure providers to move towards a data-driven ecosystem and economy.

Autonomous Management: FogFlow can carry out IoT service orchestration decisions in a decentralized and autonomous manner. This means each FogFlow edge node can make its own decisions only based on a local context view. This way the majority of workloads can be directly handled at edges without always relying on the central cloud. With this "cloudless" approach, FogFlow can not only provide fast response time, but also achieve high scalability and reliability.

Optimized Deployment: In FogFlow the task configuration and deployment of data processing flows is optimized for the cloud-edge environment to meet certain goals, for example, minimizing the internal data traffic between cloud and edge nodes, minimizing the latency to produce expected actionable results from raw data, or maximizing the accuracy of the produced results. The optimization of data processing flows not only happens at the beginning of launching a service, but also continues during the entire service lifetime. One of such optimization behaviors is dynamically migrating tasks from one edge to another edge in order to maintain the minimal response time for mobile objects like connected cars or flying drones.

3. Edge Programming Models

Currently the following two programing models are provided by FogFlow to support different types of workload patterns.

3.1 Service Topology

The first workload pattern is to trigger necessary processing flows to produce some output data only when the output data are requested by consumers. To define an IoT service based on this pattern, the service provider needs to define a service topology, which consists of a set of linked operators and each operator is annotated with a specific granularity. The granularity of an operator will be taken into account by FogFlow to decide how many task instances of such an operator should be instantiated based on the available data. A service topology must be triggered explicitly by a requirement object issued by a consumer or any application. The requirement object defines which part of processing logic in the defined service topology needs to be triggered and it can also optionally define a specific geo-scope to filter out data sources for applying the triggered processing logic. More details can be seen in our previous paper[2)] and the online tutorial[3)].

3.2 Fog Function

The second workload pattern is designed for the scenario in which service designers do not a-priori know the exact sequence of stream processing steps. Instead they can define a fog function to include a specific operator for handling a given type of information. FogFlow can then create the graph of processing flows based on this description of all fog functions.

Different from service topology, a fog function is a very simple topology with only one operator and it is triggered when its input data become available. As FogFlow can automatically chain different fog functions as well as allow more than one fog functions to handle a new data item, a constantly changing execution graph can be automatically triggered and managed by the FogFlow runtime as data arrive and disappear.

From the design perspective, fog function is more flexible than service topology, because the overall processing logic of an IoT service can be easily changed over time by adding or removing fog functions when the service processing logic needs to be modified for new business requirements. With the fog function programming model, FogFlow can support serverless computing for a cloud-edge based environment.

4. Use Cases

This section explains how FogFlow programming models can be used by service providers to realize different smart IoT services. Two concrete use case examples are introduced: the first one is based on service topology while the second one is based on fog function.

4.1 Lost Child Finder

This use case is to find a lost child as soon as possible by taking advantage of edge computing enabled by FogFlow. Assume that there are lots of cameras deployed in a smart city and also inside stadiums. As part of the city infrastructure, some edge nodes like IoT Gateway are deployed in many different fields of the city to be managed by FogFlow for supporting various IoT services.

One service that can be easily realized based on FogFlow is to find a lost child using the service topology illustrated by **Fig. 5**. The designed service topology consists of three different operators: Face Extraction that can recognize and extract face images from camera

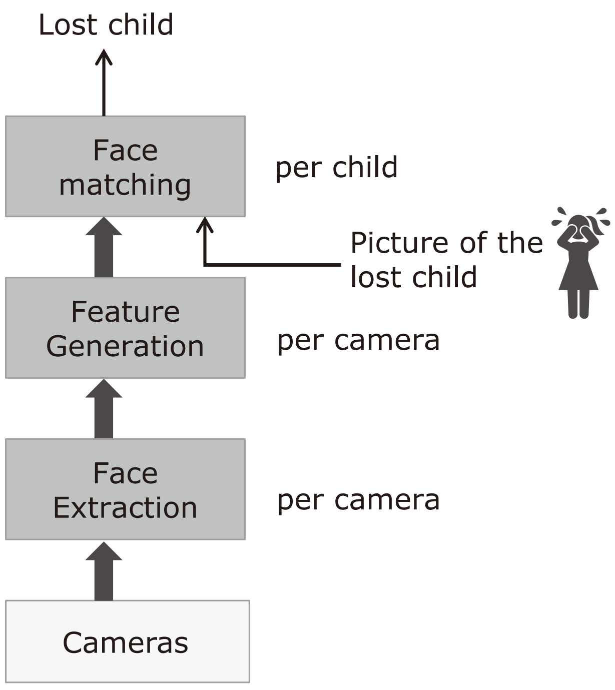


Fig. 5 Lost child finder.

video streams, Feature Generation that calculates the feature vector for each detected face, and Face Matching that compares detected faces with the face of the lost child in terms of their feature vectors. Different granularity is defined with each operator in the service topology, for example, Face Matching is instantiated per child while the other two operators are instantiated per camera.

To trigger this service topology, an external application issues a requirement and also subscribes to the output result of the Face Matching operator. By changing the geo-scope defined in the requirement, the external application can control FogFlow to orchestrate this service topology for a changing geo-scope, so that we can first search for the child in a small scope and then expand the searching scope step by step if the child is not found.

In this use case the external application as a consumer is very simply because all the complexity of how to dynamically orchestrate the data processing flows for a changing geo-scope is handled by FogFlow. In addition, according to our experiment, we can also reduce bandwidth consumption by 95% as compared with a cloud-based approach.

4.2 Smart Parking

We implemented this use case together with our European project partner, University of Murcia, based on the real scenario of Murcia City. In Murcia, there are

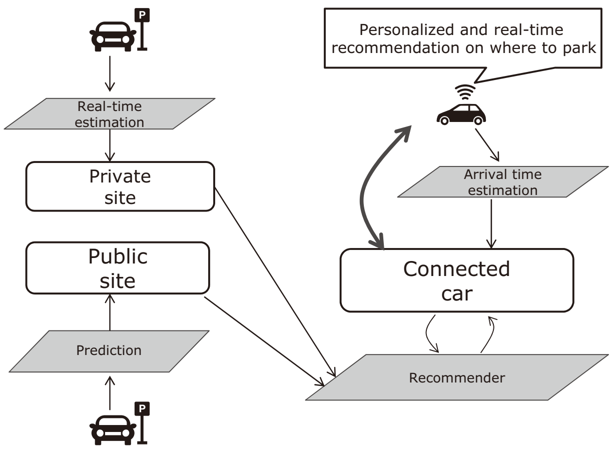


Fig. 6 Smart parking.

two types of parking sites, regulated parking zones that are operated by the city government and can provide historical information of how parking slots are used per day, and private parking sites that are operated by private companies and can provide real-time availability of parking spots. By utilizing these two types of data sources and other public transportation information, our Smart Parking service can provide real-time and personalized parking recommendation for each driver.

In this use case, it is not easy to apply service topology, but using fog function to realize the required data processing logic is straightforward. As illustrated by **Fig. 6**, we just need to design and implement dedicated fog functions for each physical object involved in the use case. For example, one fog function for each public site to predict how many parking spots are available per 10 minutes based on their historical information; two fog functions for each connected car, one to estimate its arrival time according to the traffic situation on the way and the other to calculate at which park site the driver can get a parking spot on arrival. The deployment of those fog function instances are placed on the edge node close to their input data sources so that FogFlow can help to reduce more than 50% bandwidth consumption and also provide real-time parking recommendation for each driver.

5. Interworking with FIWARE

Since November 2017 FogFlow has been promoted as an incubated open source Generic Enabler (GE) in the FIWARE community[4]. Within this community, FogFlow holds a unique position as Cloud-Edge Orchestrator to launch and manage dynamic data processing flows seamlessly over cloud and edges for data ingestion, transformation, and also advanced analytics. As illustrated by **Fig. 7**, FogFlow can interwork with other FIWARE GEs to power FIWARE-based IoT Platforms at the following two layers.

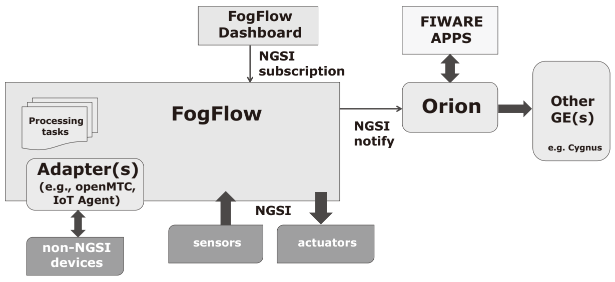


Fig. 7 Integration with other FIWARE GEs.

At the upper layer, FogFlow can interwork with Orion Broker via the standardized NGSI interface in the following two ways. The first one is to consider Orion Broker as the destination of any context information generated by a FogFlow IoT service. In this case a NGSI subscription must be issued by an external application or FogFlow Dashboard to ask FogFlow to forward the requested context updates to a specified Orion Broker. The second one is to consider Orion Broker as a data source to provide additional information. In this case we can implement a simple fog function to fetch any necessary information into the FogFlow system. In either way there is no need to make any change to the existing Orion-based FIWARE system. Therefore, this type of integration can be done fast with nearly zero effort.

At the low layer, for the integration with any Non-NGSI supported devices like MQTT, COAP, OneM2M, OPC-UA, LoRaWAN, FogFlow can reuse the modules of existing IoT agents and transform them into FogFlow adapters based on the fog function programming model. With these adapters FogFlow can dynamically launch necessary adapters for device integration directly at edges. This way, FogFlow can and is able to talk with a wide range of IoT devices.

6. Conclusion

FogFlow is a distributed execution framework to orchestrate IoT services by managing their dynamic data processing flows over cloud and edges in a seamless and scalable manner. This article introduces its design goals, key technology features, and business value propositions. We also briefly explain its programming models, including service topology for on-demand data processing and fog function for serverless edge computing. Two use cases are presented to illustrate how these two programming models can be used to realize city-scale IoT services.

FogFlow has been promoted as an open source FIWARE GE and it can easily interwork with other FIWARE GE. As a standard-based and data-driven edge computing framework, FogFlow provides a unique basis for infrastructure providers to make their infrastructure open towards a data-driven ecosystem and economy.

Reference

1) NGSI data model
http://fiware.github.io/specifications/ngsiv2/stable/
2) B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa and A. Kitazawa: FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities, IEEE Internet of Things Journal, Volume 5, Issue 2,pp.696-707, April 2018
3) FogFlow Tutorial
http://fogflow.readthedocs.io
4) NEC Press release: NEC Develops a FIWARE-based Fog Computing Framework for Edge-based IoT Services, November 2017
https://uk.nec.com/en_GB/press/201711/20171127_01.html

Authors' Profiles

CHENG Bin
Senior Researcher
NEC Laboratories Europe

KOVACS Ernoe
Senior Manager
NEC Laboratories Europe

KITAZAWA Atsushi
Professional Fellow
NEC Solution Innovators

TERASAWA Kazuyuki
Senior Manager
Future City Development Division

HADA Tooru
Executive Expert
IoT Platform Solutions Division
NEC Solution Innovators

TAKEUCHI Mamoru
Manager
IoT Platform Solutions Division
NEC Solution Innovators

Information about the NEC Technical Journal

Thank you for reading the paper.
If you are interested in the NEC Technical Journal, you can also read other papers on our website.

Link to NEC Technical Journal website

Japanese   English

Vol.13 No.1 Sustainable Data-driven City Management

Orchestrating a brighter world
NEC
NEC Technical Journal
Sustainable Data-driven City Management
Vol.13 No.1

Vol.13 No.1
November 2018

Special Issue TOP