PrePrint: Cite Final Version as: Giang, N. K., Leung, V. C. M., Kawano, M., Yonezawa, T., Nakazawa, J., Lea, R. & Broadbent, M., 2019, IEEE International Conference on Big Data and Smart Computing (BigComp). IEEE press

CityFlow: exploiting edge computing for large scale smart city applications

Nam Ky Giang, Victor C.M. Leung
Electrical and Computer Engineering
The University of British Columbia
Vancouver, Canada
{kyng, vleung}@ece.ubc.ca

Makoto Kawano, Takuro Yonezawa, Jin Nakazawa
Graduate School of Media and Governance
Keio University
Fujisawa, Japan
{makora, takuro, jin}@ht.sfc.keio.ac.jp

Rodger Lea, Matt Broadbent
Computing and Communications
Lancaster University
Lancaster, U.K.
{ r.lea1, m.broadbent}@lancaster.ac.uk

***Abstract*—This paper presents an approach to supporting the development process for large-scale smart city applications that leverage edge computing resources. A smart city testbed called CitiFlow is developed, which uses Distributed Node-RED as the underlying middleware to facilitate the decomposition and communication among sub components of large scale smart city applications. For the evaluation, a lab-based setup and a real world deployment were executed and are presented.**

Keywords—Fog, Edge computing, Smart Cities, IoT

I. INTRODUCTION

The development of Smart Cities, and the applications that provide services has benefited from a number of new technologies including the growing use of Fog/Edge computing technologies [1]. However, while these technologies offer significant benefits it is still the case that developing applications to exploit them is complex and time consuming and that truly large-scale smart city applications are still in their infancy. In this paper, we highlight some of these technical challenges and then describe our research efforts to develop a platform, CityFlow, that is designed to address these challenges and to encourage the development and deployment of large-scale city applications. We validate the approach using a lab-based trial and a real-world deployment.

II. ISSUES AFFECTING THE DESIGN OF LARGE SCALE SMART CITY APPLICATIONS

There are a number of technical challenges that must be addressed when developing and deploying large-scale city wide applications. These include:

A. Geographic Distribution of Smart City Computing Systems

Large scale smart city applications may range from smart buildings to smart city or regions spanning metro areas of larger geographical groupings. This large-scale geographic distribution has several consequences that influence the way applications are developed.

Firstly, the computing resources are generally communicating over a heterogeneous network that involves 1) different communication mediums (e.g Wi-Fi, LTE, Wired, etc) and 2) a mix of static and dynamic endpoints with different reachability (e.g direct IP addresses vs behind NAT). This heterogeneity makes it more difficult for inter-device communication. In large scale city applications, these communication details should not hinder the development of the application in general. Theefore, the developers should be provided with programming tools and primitives that allows them to focus only on the application logic.

Second, due to the large scale distribution of computing resources, their physical location, or in general their physical context, becomes an important factor in the city computing application model. For instance, if there are many instances of smart vehicles with a dash camera, and the city is interested in understanding the road conditions in a specific location, only the dash cameras in that area should upload images for analysis.

B. Dynamic Nature of Edge Devices

Due to the close bonding with the physical world, edge devices often exhibit a highly dynamic nature, in terms of both load fluctuation and context changes (e.g. location changes when edge nodes are mobile). While load balancing and dynamic scaling is commonly used in cloud computing to cope with the fluctuation in application load, it is more difficult to do the same in the edge. This is partly because edge resources are not as centralized and readily available as cloud computing resources so that it is harder to scale horizontally. The other reason is that the heterogeneous networking environment makes it difficult to locate resources for the load balancing task.

In addition to load fluctuation, changes in physical context such as location, which as described in the previous section plays an important role in IoT applications, requires a certain level of context monitoring and situational re-evaluation. The

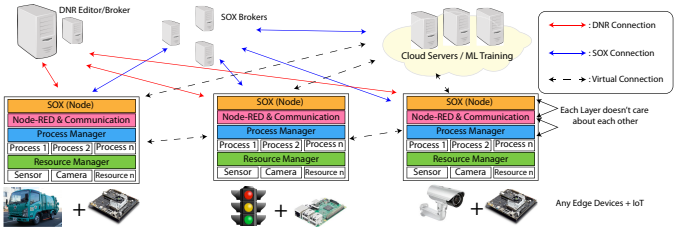

Figure 1. Overview of CityFlow.

involvement of dynamic physical context also leads to the question of how to expose the context information to programming primitives or constructs that developers can use to build the application. Returning to our previous example of cars and dash cameras, a city wide monitoring application, distributed across all cars in the city would use the contextual data - their locations - to coordinate the communication, i.e. to route the sensing streams to the appropriate processing and analysis nodes.

C. *Ease of development*

Developing applications that are able to scale and adapt to the dynamic nature of city environments is in itself an issue. Programming languages and toolchains that have evolved to facilitate the development of more traditional applications often lack the features needed to address, and in some cases exploit, the scale and dynamic nature of smart city computing infrastructure. In particular, we believe that the following factors are critical:

- Easy component reuse
- Abstract communication among components
- Scaling via automatic replication
- Adaptability to run-time changes
- Suitability for a range of skills

III. CITY FLOW

CityFlow is a new Smart City development environment that attempts to address the issues outlined in the previous section. It combines an edge process framework, called Distributed Node-RED, which is an extension of the popular visual programming tool for the IoT Node-RED, with a city scale messaging and communication framework SOXfire used as a universal sensor data exchange mechanism. CityFlow is the result of ongoing research as part of the BigClouT Smart City research project[1].

A. *Distributed Node-RED.*

Node-RED is a dataflow-based visual programming tool and language for IoT applications. Dataflow is a natural programming model represented as a directed graph of processing nodes, each of which representing a resource. The applications are developed by dragging and dropping processing nodes onto a canvas, and 'wiring' the nodes together. The wires represent communication paths between nodes. The resulting application is referred to as a "flow", which can be deployed to a single device. However, Node-RED has no support for distributed edge environments.

Distributed Node-RED (DNR) is an extension of Node-RED for distributed environments [3, 4]. There are three notable extensions, which are described below. The first extension is the notion of a *device* within the dataflow language. This notion enables individual hardware devices to be uniquely identified in a distributed environment. The device also carries a set of characteristics. This allows the user to determine which device(s) a node should be deployed to and executed on; for example, mobile devices, embedded computers, and cloud servers.

The second extension is the notion of *remote wires*. Remote wires make it possible to support inter-device communication so that the nodes running on distributed separate devices can send the data. These wires are implemented by using a publish/subscribe communication mechanism. For instance, assuming that node A sends the data to node B. Then node A publishes the data to the communication broker so that node A does not need to know where node B is. Similarly, node B subscribes to the communication broker so that node B can receive the data without needing to know its source node. Additionally, using remote wires logically separates the process so that highly computational processes can be conducted on the multiple edge devices without using cloud computing.

The last extension is the *constraint* primitive, used as a broader abstraction that specifies how a node is deployed and run in a distributed setting. Accordingly, every node in a dataflow is attached with a constraint, property that defines how the deployment is conducted. This constraint indicates requirements on device identification, computing resources such as CPU and memory, and physical location. For example, by using the constraint, we could restrict a sensor processing flow so this it only runs on a device mounted on a moving vehicle, located within a certain geographic area.

By using DNR for CityFlow, developers can concentrate on developing ML applications without knowing which devices and data resources are used. For more details or other extensions of DNR, see [4].

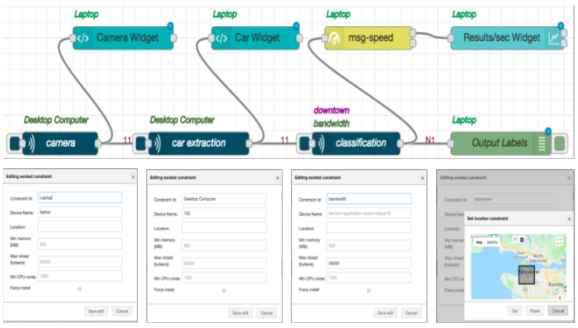

Figure 2. Simple DNR application designed to analyse traffic using image recognition

[1] BigClout project: http://bigclout.eu/

B. *Sensor-Over-XMPP Node*

CityFlow offers a node implemented in DNR that handles Sensor-Over-XMPP (SOX). SOX is the specification of SOXFire [9] which is a universal sensor data exchange system. This utilizes the Internet protocol of the open XML format (XMPP), typically used for chat communications, to represent the meta information. By using SOX nodes, data from physical sensors and virtual sensors can be treated uniformly allowing application developers to focus on the algorithms rather than issues of data gathering. Furthermore, because our focus is on distributed applications/algorithms, communication transparency is necessary to support highly distributed processing, such as Machine Learning (ML), where the model is distributed to the edge devices all over a city to conduct prediction locally.

IV. Evaluating CityFlow

A. *Lab Study: car recognition for traffic flow analysis*

Since realizing large scale application in a real world setting is difficult, an initial lab-based implementation has been developed that mimics the core application scenario.

The application aims to analyse traffic flows at busy city intersections by using image recognition to determine the type and number of cars and trucks flowing through the intersection. Figure. 2 shows the development of our example application using DNR. As can be seen, the application model consists of a data flow graph of the involved software components and the context-dependent constraints that are applied onto each software components. When the application is deployed onto the pool of participating devices, each device will process the application and all the associated context-dependent constraints. It then reasons about whether or not to enable certain components, as well as to fetch/redirect data from/to any external devices.

In our lab setup we deploy this application to a simple set of devices. Specifically, we have two embedded boards, one desktop computer and one laptop that participate in the distributed application. With regard to the software components involved, we still have one component that captures the camera feed (through playing a video footage), the data is sent to a background subtraction component to extract the foreground running cars, which are subsequently classified by a neural network classification model.

The lab-based experimental setup is seen in Figure. 3 . In our setup, we name the desktop computer *152*, the two

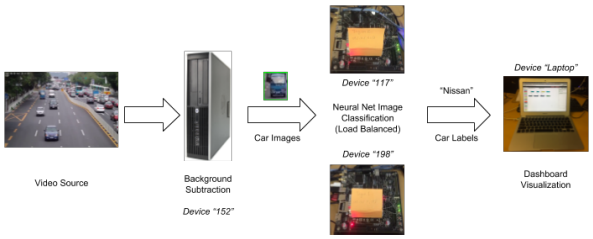

Figure 3. Lab setup for our example fog application.

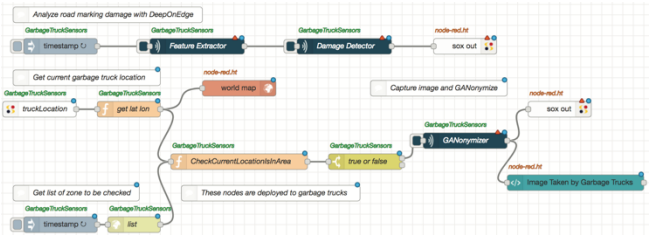

Figure 4. Road damage detection application in CityFlow.

embedded boards *117* and *198*. For the sake of simplicity, these numerical identification is manually named after their IP addresses in our local network. For example, if one's IP address is *192.168.1.117*, it is named *117*. Since these naming becomes irrelevant in large scale deployment, our DNR platform can constraint the execution of software components on devices using their physical context such as the level of memory usage, or their locations. To accomplish this, device *117* and *198* are configured with a predefined latitude and longitude. Our application is developed so that the video source and background subtraction components are constrained to run on a desktop computer, thus the device *152* while the neural network image classification component is constrained to run at a certain location (e.g downtown).

On deployment, this constraint resolves to the two embedded boards with pre-configured coordinates. There are also device capability constraints applied to the image classification component so that it only runs on devices with a minimal amount of network bandwidth.

This essentially demonstrates the load-balancing capability of the system, which in turn, highlights its dynamic nature. That is, our coordination platform has to periodically monitor the devices' network bandwidth to make the coordination decision as to which device to send the car images stream to. Since we let the developer to explicitly specify the network bandwidth threshold at which the load-balancing coordination is triggered, the example also shows how physical status of the underlying computing infrastructure (e.g current network consumption of devices) can affect the correctness of the application logic (e.g only run this component on devices that have minimum available bandwidth).

Our lab-based setup allows us to verify the behaviour of our example application when it is distributed to a number of devices. It also shows that the application model itself is scalable as it does not rely on any specific deployment details. These could be what types of devices are involved, where are they located, which network connection they are on, etc. In large scale applications, this is an important characteristic of the application model, which is also known as a *programming-in-the-large* approach.

B. *Real World deployment: Road Damage Detection*

Following on from our lab study, we use a road damage detection application as a first case real-world study, with a focus on road line markings. This application has been deployed to garbage trucks in the city of Fujisawa, Japan, and runs live, gathering information on the status of the road network for city officials.

Although road damage is a common problem, in many cities, inspection is still conducted by sight. This manual visual inspection is resource intensive and expensive. Therefore, we have to explore ways to inspect the city infrastructures, especially roads, at low cost. Recent work by Kawano et al. [5] proposed a method for road inspection using recorded images from cameras mounted on garbage trucks. However their approach relied on central cloud processing which is inefficient and costly in terms of processing and communications and fails to handle local privacy issues. Simultaneously, since they adopted an object detection approach [6] which uses a large neural network, it is difficult to work on edge devices.

Moreover, we have to treat a covariate shift [6, 7] in the city. The covariate shift occurs because the relationship between the data and the desired output does not differ but the difference of the marginal distribution between a training dataset and a test dataset. We assume that the covariate shift exists in the city, as the distribution of the data from the city often varies significantly in spatial and in temporal domains, namely, a spatial-temporal covariate shift. Our application to address these problems, implemented using CityFlow, operates as follows Figure 4:

The flow of Figure 4 top. In order to deploy the networks to edge devices and cope with a spatial-temporal covariate shift, the neural network is separated into feature extractor node and damage detector node and deployed to different edge devices on the truck, respectively. Consequently, we can adopt a domain adaptation learning approach [2], so that we can exchange the damage detector in order to treat the spatial-temporal covariate shift. Then, when the damaged road is detected through these networks nodes, the location of it and the degree of the damage are published to SOXFire via SOX Node (sox-out in the flow).

The flow of Figure 4 bottom. When multiple garbage trucks confirm an area of road damage, one truck is designated to upload a partial video of the area. A SOX-in node receives the location and the damage information, and the next node compares the location of the truck with it. If it is true, the driving recorder mounted on the truck sends the videos after anonymizing to the cloud visualization application. Before uploading, information related to the person such as faces or car matriculation plates is removed if the video contains them. Upload and display the videos in a suitable application for confirmation by city staff and the staff will fix the road.

V. Conclusion and Future Work

While we are still at an early stage, it is clear that the CityFlow platform is a powerful tool for developing and deploying large scale smart city applications. In particular, our real-world trial using Fujisawa City garbage trucks, has validates the overall platform and the ability to easily develop and deploy sophisticated applications that combine advanced ML algorithms with distributed processing.

Our first area of ongoing research is focused on the issue of coordination among the distributed elements of the smart city applications and in particular on how we support exogenous co-ordination. More details of our initial efforts can be found in [3]. Secondly, we are exploring distributed ML algorithms and aim to extend CityFlow with explicit support for distributed learning. Finally, although we have deployed CityFlow in a real world context in Fujisawa, we are keen to extend both the scale of our applications, and the geographical reach and hope to deploy to other cities in the BigClouT project in the near future.

Acknowledgement

This work was supported in part by National Institute of Information and Communications Technology and in part by H2020-EUJ-2016 EU-Japan joint research project, BigClouT (Grant Agreement N723139)

References

[1] Lea, R.: Smart Cities: An overview of the technology trends driving Smart Cities. IEEE press, March 2017, http://doi.org/10.13140/RG.2.2.15303.39840

[2] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., Lempitsky, V.: Domain-adversarial training of neural networks. The Journal of Machine Learning Research 17(1), 2096–2030 (2016)

[3] Giang, N.K., Lea, R., Leung, V.C.M.: Exogenous coordination for building fog-based cyber physical social computing and networking systems. IEEE Access 6, 31740–31749 (2018). https://doi.org/10.1109/ACCESS.2018.2844336

[4] Giang, N.K., Lea, R., Blackstock, M., Leung, V.C.M.: Fog at the edge: Experiences building an edge computing platform. In: 2018 IEEE International Conference on Edge Computing (EDGE) (2018)

[5] Kawano, M., Mikami, K., Yokoyama, S., Yonezawa, T., Nakazawa, J.: Road marking blur detection with drive recorder. In: 2017 IEEE International Conference on Big Data (Big Data). pp. 4092–4097 (Dec 2017)

[6] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)

[7] Shimodaira, H.: Improving predictive inference under covariate shift by weighting the log-likelihood function. Journal of statistical planning and inference 90(2), 227–244 (2000)

[8] Sugiyama, M., Kawanabe, M.: Machine learning in non-stationary environments: Introduction to covariate shift adaptation. MIT press (2012)

[9] Yonezawa, T., Ito, T., Nakazawa, J., Tokuda, H.: Soxfire: A universal sensor network system for sharing social big sensor data in smart cities. In: Proceedings of the 2nd International Workshop on Smart. p. 2. ACM (2016)