Received ; | Revised ; | Accepted

DOI: xxx/xxxx

PREPRINT. FINAL VERSION: HTTPS://DOI.ORG/10.1002/SPE.2695

Developing Applications in Large Scale, Dynamic Fog Computing: a Case Study.

Cite this paper as N. K. Giang, R. Lea, and V. Leung (2019), Developing Applications in Large Scale, Dynamic Fog Computing: a Case Study, Software: Practice and Experience, 2019; pp 1-14. https://doi.org/10.1002/spe.2695

Nam Ky Giang[1] | Rodger Lea[2] | Victor C.M. Leung*[1]

[1]Department of Electrical and Computing Engineering, The University of British Columbia, Vancouver, Canada
[2]School of Computing and Communication, Lancaster University, Lancaster, United Kingdom

Correspondence
*Victor C.M. Leung Email: vleung@ece.ubc.ca

Present Address
The University of British Columbia, Vancouver, BC

Summary

In recent years, fog computing has emerged as a new distributed system model for a large class of applications that are data-intensive or delay-sensitive. By exploiting widely distributed computing infrastructure that is located closer to the network edge, communication cost and service response time can be significantly reduced. However, developing this class of applications is not straightforward, and requires addressing three key challenges. Supporting the dynamic nature of the edge network, managing the context-dependent characteristics of application logic and dealing with the large scale of the system. In this paper, we present a case study in building fog computing applications using our open source platform Distributed Node-RED (DNR). In particular, we show how applications can be decomposed and deployed to a geographically distributed infrastructure using DNR, and how existing software components can be adapted and reused to participate in fog applications. We present a lab based implementation of a fog application built using DNR that addresses the first two of the issues highlighted above. To validate that our approach also deals with large scale, we augment our live trial with a large scale simulation of the application model, conducted in Omnet++, that shows the scalability of the model and how it supports the dynamic nature of fog applications.

KEYWORDS:
edge/fog computing, exogenous, dataflow, dynamic, large scale, coordination

1 | INTRODUCTION

As the number of Internet of Things (IoT) devices increases, the amount of data generated by these devices is putting pressure on the traditional centralised cloud computing infrastructure. Fog computing, and its relation, edge computing, where computing resources are distributed closer to the edge network, has emerged as a system model to support many data-intensive or delay-sensitive applications[1].

While this is promising, building applications to leverage the computing resources across the edge network, fog and cloud, is not an easy task. Interesting research questions are, to name a few, what does an application model in fog computing look like? How do we decompose the application and deploy its constituents to the widely distributed infrastructure? How do we support the characteristics that are inherently relevant in the edge network, such as the large-scale, the dynamic nature and the context-dependent nature of computation?

To start experimenting with building this class of applications, we have developed Distributed Node-RED (DNR), a distributed application platform based on the open source project Node-RED[2], which provides a visual dataflow programming paradigm for building IoT applications. During the course of our research, our DNR project has gone through several iterations - from being a data flow based programming tool for building distributed IoT applications to a dynamic coordination platform for large scale fog/edge computing[3].

DNR embraces a dynamic, context-dependent and exogenous coordination model designed to support fog applications. In exogenous coordination[4], applications consist of software constituent components that can be independently developed. The cooperation of these components (i.e. when and in which order should they exchange data) is controlled/coordinated by an external coordinator. This coordination model explicitly separates the communication and computation aspect of an application, thus providing a loosely coupled distributed application model, an important requirement for scalability.

Exogenous coordination is not a new idea, and although theoretically applicable to large scale fog computing, our recent work[5], showed that applying exogenous coordination to fog computing is not straightforward. For example, due to the dynamic nature of the systems, the coordinator has to periodically monitor the system and communicate the most up to date coordinated results to all the constituents. This proves to be costly.

In this paper, we present the process of using DNR to develop a real-world fog application that involves video processing and neural network image classification of CCTV camera footage using fog-based load balancing capability. In particular, we show how existing software components such as vision processing libraries or deep learning neural networks are modified to participate in a larger scale fog application. We also show, through the fog-based load balancing capability, the influence of the dynamic characteristic of fog computing systems on the design of our coordination platform. Finally, we show that a stateful or incremental coordination model is necessary to fully support the development of large scale and dynamic fog applications.

Our DNR project, as well as our example fog application are fully open source and available online for evaluation[6]. It is worth noting that, the class of fog applications addressed in this paper refers to large scale applications that leverage fog computing resources in a large area such as smart cities[7]. Smaller scale problems, such as developing applications for smart homes or smart environments have their own unique challenges, therefore, are out of the scope of this paper.

In the next section we start reviewing the unique characteristics of our target class of fog applications.

2 | LARGE SCALE, DYNAMIC FOG COMPUTING APPLICATIONS

Fog Computing infrastructure generally refers to the computing resources provided by the access networks such as routers and cell towers[8]. In our work however, we consider fog computing infrastructure to include various distributed computing resources ranging from edge network devices to the cloud and everything in between. Thus, our fog applications can collect data, do physical actuation at the edge network, have some delay-sensitive processing done in the access network and also can store data in the cloud for further analysis.

We identify our class of fog applications as large scale and dynamic distributed applications. This is because it takes into account the dynamic nature of edge devices (e.g. mobile phones, smart cars) and involves a large number (e.g. hundreds or thousands) of devices that are scattered across a large physical area. Due to these characteristics, our class of fog applications has a very close bonding with the physical context of the computing infrastructure. That is, unlike traditional distributed applications running on the cloud with minimal knowledge of the underlying infrastructure, our fog applications are intimately connected to the device they are deployed on. Figure. 1 illustrates the positioning of our application class with regard to three attributes: the deployment scale of the system, the level of bonding with physical context and the dynamic nature of the system.

The deployment scale of the system can range from smart home, spanning smart offices, factories to smart cities. We argue that at each deployment scale, there is particular challenge pertaining to it, thus has its own solution. For example, at small scale such as smart home, device naming and identification could be a problem of interest. However, this problem becomes less important when it comes to the larger deployment scale of a smart city.

The level of bonding with physical context can range from no bonding at all, such as traditional Internet applications, to hardware capability aware applications such as some applications that require a certain hardware capability (e.g. number of CPU cores, has Bluetooth connectivity), to physical context aware applications such as the ones that only run in a certain location (e.g. only collect sensing data in downtown area, provided that the sensing component is deployed in a running vehicle).

The dynamic nature can range from static environment such as applications running in a data centre, with minimal load fluctuations or on immobile devices such as desktop computers, to semi-dynamic where the dynamic characteristic could be modelled or predicted, and to fully-dynamic systems such as where the mobility of devices or load fluctuations cannot be

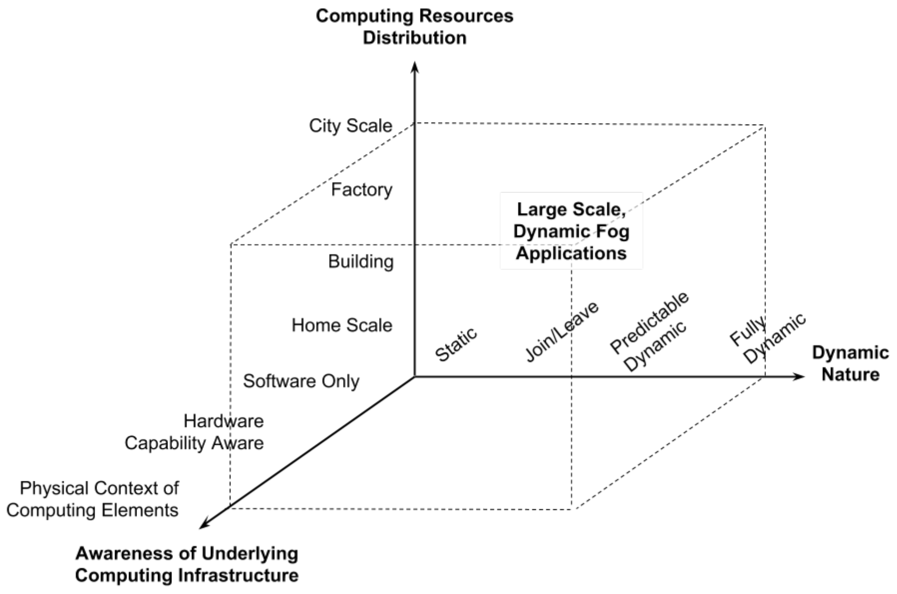

Computing Resources Distribution
City Scale
Factory
Building
Home Scale
Software Only
Large Scale, Dynamic Fog Applications
Static
Join/Leave
Predictable Dynamic
Fully Dynamic
Dynamic Nature
Hardware Capability Aware
Physical Context of Computing Elements
Awareness of Underlying Computing Infrastructure

FIGURE 1 Our class of fog computing applications.

predicted. A periodic context acquisition and evaluation is usually required in a fully-dynamic system so that the application's correctness can be verified and maintained.

3 | MOTIVATING SCENARIO

In this section, we introduce our example fog application that is built using our DNR platform and lay out the requirements on the application platform in order to support our example application.

3.1 | Theoretical application modelling and coordination requirements

The application takes advantages of cameras mounted on cars or lamp posts and the computing infrastructure across the city to carry out vision processing and image classification tasks. For example, to identify locations of lane markers on the road network that need to be repainted, or looking for a suspect car to help the police. The cameras could be dashcams mounted on city buses, or CCTV cameras at traffic junctions. The video streams might go through several processing steps such as sizing, recolouring, or background subtraction. The processed image streams are then pushed through neural network components to identify good or bad lane markers, or to classify and recognise cars.

Due to the heavy nature of video data, it is best to leverage the fog computing infrastructure to have data processing tasks closer to the edge network, where video streams are generated. It is also unreasonable to have all the processing tasks running inside a single edge device due to the resource constraints. Thus, a distributed deployment where multiple devices cooperate together is necessary. It is also assumed that the city has some of these computing resources available at the edge network, such as in road-side units, in traffic control elements or in partnering cell towers.

From the software development perspective, there are several high level software components involved in the application: a camera capturing component, several vision processing and neural network components. When these software components are deployed into the fog infrastructure, this can be seen in Figure. 2 .

Due to the geographic constraints, there are locations that do not have traffic components, or have overlapping communication coverage. This means pre-configuring the communication among devices does not work. Also where there is no infrastructure available, a particular car might be selected to host the data processing component if it can fulfil the application's constraints. Furthermore, the communication among software component is highly dynamic as one fog device (e.g. in one cell tower) might have a different reception and data rate than the other. This will effectively affect the application's performance and behaviour.

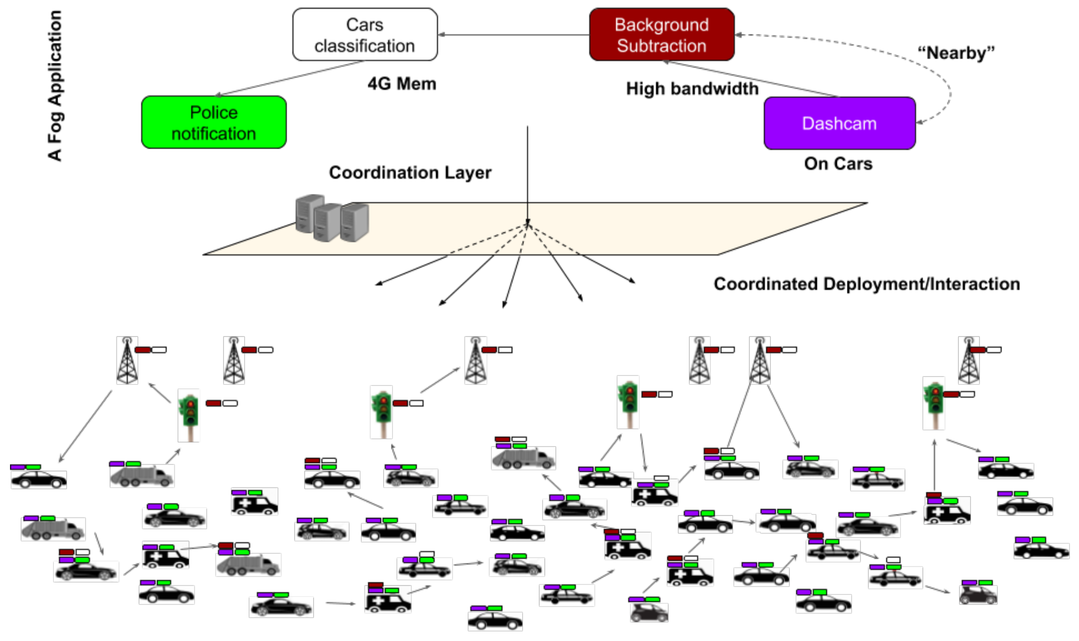


FIGURE 2 Example fog application.

Because of these physical characteristics that are specifically related to the large scale, dynamic fog infrastructure, an external coordination layer is required to coordinate the cooperation among software components dynamically.

The task of the coordination layer is then to coordinate the communication among each participating instances of the software components so that the application's requirements are met. For example, choosing the right instances of the Background Subtraction component for the instances of the Dashcam component to send data to (e.g the arrows between devices). Due to the mobile nature of the participant, this coordination task has to be done periodically to keep up with the system's dynamic nature. This proves to be costly in real deployment so to be more efficient, it should refer to past assignments and make incremental changes to avoid reassigning the communication peers if not necessary. Thus, stateful or incremental coordination is needed.

Note that there is a significant difference between our large scale fog application and smaller scale distributed applications or traditional Internet applications. In those types of applications, there usually exists only one combination of the software components' instances to fulfil the service. For example, in a typical smart home application, one combination of a temperature sensor instance, a regulator instance and a heater instance would adequately fulfil an automated climate control application. In our class of fog applications, due to the large scale deployment of the computing devices, there are many possible combinations of the software components' instances. Each combination might serve a particular location of the city and each component instance sees one another as an independent peer. Therefore, the combinations have to be non-overlapping (no shared instances between combinations) and finding as many of these non-overlapping combinations as possible is an important task of the coordination layer.

Further details of these coordination requirements and implementation techniques can be found in[5].

3.2 | Actual experimental methodology

Since realising such an application in a real world setting is difficult, a lab-based implementation has been developed that mimics the core application scenario. While the number of participating devices are significantly reduced, the number of software components involved stays the same. That is, in our setup, 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.

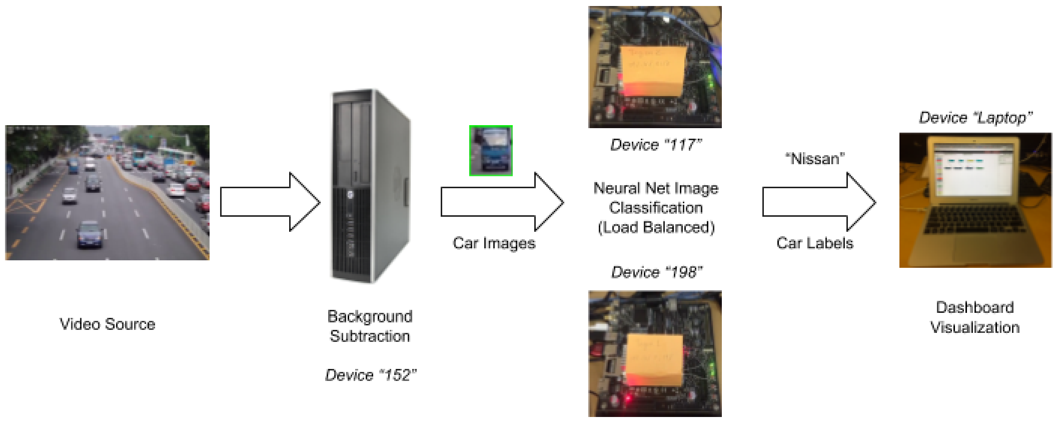


FIGURE 3 Lab setup for our example fog application.

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

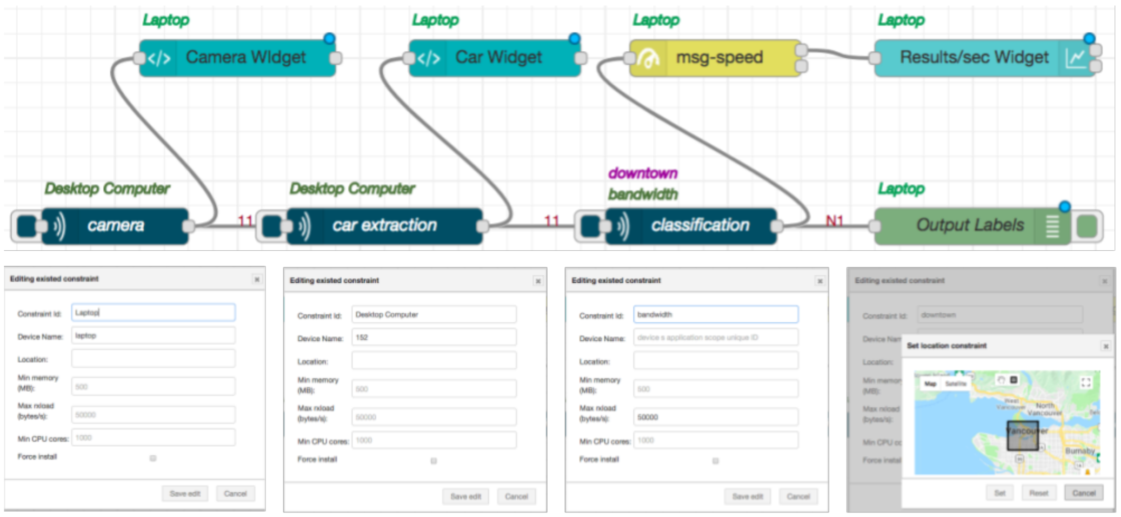
Since these naming becomes irrelevant in large scale deployment, our DNR platform can constrain the execution of software components on devices using their physical context such as the level of memory usage, or their location. To simulate this, device *117* and *198* are configured with a predefined latitude and longitude (this is a common practice in case of lamp posts deployment). Our example application is then designed such that, the video source and background subtraction components are constrained to run on a desktop computer while the neural network image classification component is constrained to run at a certain location (e.g downtown). On deployment, the desktop computer resolves to the device *152* and the "downtown" devices resolve to the device *117* and *198* (two embedded boards).

In our lab-based experiment, it is difficult to simulate the mobility behaviour of the participating devices (i.e dashcams mounted on cars). While we can certainly replay car traces from available open data sets, or use traffic simulation suite such as SUMO[9], the small number of participating devices make it irrelevant to have moving devices while still keeping the data streams continuous. In a real world scenario, with a large number of participating devices (i.e. cars, lamp posts, traffic elements), maintaining the data streams among them is easier. When a car moves out of the coverage of a fog node, e.g. a wireless enables street light, another one could move in immediately, allowing the fog node to continuously get input for its processing task. In our lab-based experiment with only a limited number of devices available, the data streams will be less stable, thus the outcome will be intermittent.

Since the mobility of devices in the studied scenario is used to show the dynamic nature of the system, we opt to look for a different approach to study this dynamic characteristic. We realised that in fog computing, since the computing activities are closely bound to the physical context of the devices, the system load for each device also frequently fluctuates. For example, when we process camera footage of a street scene, the number of cars extracted from the footage varies from second to second. Thus, we use the system load factor to illustrate the dynamic nature of the system in our lab-based experiment and incorporate a load balancing feature into our lab-based setup. To do this, a device capability constraint that is based on the network load is created and applied to the image classification component. Thus, this component is constrained so that it only runs on devices with a required amount of network bandwidth.

When the number of extracted cars is small, the input to the image classification component is small, therefore, network load of the underlying device is low. The device continues to do the job. When this number increases, the underlying device experiences higher load and, through context synchronisation, the external coordinator starts to route the data stream to a second device that also satisfies the constraint to run the image classification component.

This essentially demonstrates the load-balancing feature of the system, which in turn, illustrates its dynamic nature. It also illustrates that to support the dynamic nature of fog computing applications, the coordination platform has to periodically monitor the devices' network bandwidth and other resources to make the coordination decision as to which device to send the car image stream to.



Editing existed constraint

Constraint Id: Laptop
Device Name: laptop
Location:
Min memory (MB): 500
Max nxload (bytes/s): 50000
Min CPU cores: 1000
Force install

Editing existed constraint

Constraint Id: Desktop Computer
Device Name: 152
Location:
Min memory (MB): 500
Max nxload (bytes/s): 50000
Min CPU cores: 1000
Force install

Editing existed constraint

Constraint Id: bandwidth
Device Name: device s application scope unique ID
Location:
Min memory (MB): 500
Max nxload (bytes/s): 50000
Min CPU cores: 1000
Force install

Editing existed constraint

Constraint Id: downtown
Device Nam
Location:
Min memor (MB):
Max nxload (bytes/s):
Min CPU co
Force instal

Set location constraint

FIGURE 4 Fog applications design and development in DNR.

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 the 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 is participatory-based and does not rely on any specific deployment details such as what types of devices are involved, where are they located, which network connection they are on, etc. Thus if a device decides that it cannot run a certain component, it just needs to move on with other components. Meanwhile, the external coordinator always has the collective component instances contributed by the devices, it can coordinate the communication among those available instances so that the application's requirements are met. In large scale applications, this is an important characteristic of the application model, which is also known as a *macro programming* approach.

4 | DEVELOPING FOG APPLICATIONS WITH DNR

This section describes in details how we implement our lab-based fog application scenario using our DNR framework. Our DNR framework is an extension of the open source project Node-RED, a data flow based, visual programming tool for composing applications from individual software components.

Our early work[10] has shown that by leveraging the intuitive data flow and visual programming model, large scale and complex applications can be built easily without the need to worry about a class of problems related to the underlying computing infrastructure. Some of these problems are device heterogeneity or network fragmentation. It also reduces the gap between design and implementation, thus facilitating the prototyping and tryouts of complex applications. This is a particularly important requirement for building applications targeted at large scale fog computing infrastructure due to the complexity involved.

Besides supporting a compositional and scalable application model based on data flow, DNR extends the capability of the Node-RED project by allowing developers to distribute the application flow onto multiple devices as well as to express context-dependent constraints in their applications. These are important capabilities as they now have the tool needed to deploy complex applications that take advantages of the large scale, dynamic and distributed fog computing environment.

Figure. 4 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 (top of Figure) and the context-dependent constraints that are applied onto each software components (bottom of figure). 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.

5 | SOFTWARE COMPONENT REUSE

The exogenous coordination approach requires that each participating software component be modelled as a pure computing function with input and output connections. This high level of abstraction makes it very easy to reuse existing software components to build large scale applications, thus, reduces the effort in building the applications themselves.

This also means that existing software components have to be rewritten to conform to these abstraction. However, from our experience, we show that only a few lines of code need to be added/changed in order to allow a complex software component to conform. We tested with a background subtraction module from the OpenCV[11] library and an image classification neural network using the Caffe framework[12].

The main idea is to exploit Linux pipes as the input and output ports for these software components. Accordingly, instead of running a one-off command with input arguments to start the software components whenever there is data at their input, the software components are started upfront, awaiting data at their process' standard input pipe. Once they complete their data processing, they place the results in their standard output pipe (e.g. using the print function).

Since we use Node-RED as the underlying middleware for the components, we developed a special Node-RED node (*node-red-contrib-pythonshell*) that wraps around the actual component code to help communicate between these components and the Node-RED runtime. The images data passing through the components are then serialised using base64 encoding. The following listings show the code of these software components and how they are adapted to work with our exogenous coordination platform.

Listing 1: Code for Video Source Component (in Python)

```
...
# video source
cap = cv2.VideoCapture('vid.mp4')
while(True):
    ret, frame = cap.read()

    # serialization
    ret, frameBuf = cv2.imencode('.png', frame)
    frameStrBase64 = base64.b64encode(frameBuf)

    # send data to stdout
    print frameStrBase64
```

Listing 2: Code for Background Subtraction Component (in Python)

```
...
# read from stdin
while True:
    frameStrBase64 = sys.stdin.readline()

    # deserialization
    frameStr = base64.b64decode(frameStrBase64)
    frameNp = numpy.frombuffer(frameStr, dtype=numpy.uint8);
    frame = cv2.imdecode(frameNp, flags=1)

    # get foreground, thresholded, eroded, dilated, finding contours steps are omitted
    ...
    for car in carContours:
        ret, frameBuf = cv2.imencode('.png', car)
        frameStrBase64 = base64.b64encode(frameBuf)

        # send data to stdout
        print frameStrBase64
```

Listing 3: Code for Neural Network Image Classification (in Python)

```
# caffe network initializations are omitted
...
# read from stdin
while True:
    frameStrBase64 = sys.stdin.readline()

    # deserialization
    frameStr = base64.b64decode(frameStrBase64)
    frameNp = numpy.frombuffer(frameStr, dtype=numpy.uint8);
    car = cv2.imdecode(frameNp, flags=1)

    # caffe image class classification
    net.blobs['data'].data[...] = transformer.preprocess('data', car)
    output = net.forward()

    # send data to stdout
    print 'predicted class is:', labels[output['prob'][0].argmax()]
```

6 | DYNAMIC COORDINATION

The dynamic nature of the edge network requires the coordination layer to be also dynamic. Since there is no prior knowledge of how the system will behave and it is also unpredictable, a periodic coordination execution is required to ensure the current combinations of software instances still meet the application's context-dependent constraints. In our example application and its' implementation using DNR, the developer imposed an application constraint on the neural network image classification node that limits its execution to devices that have enough network bandwidth. To ensure this requirement is met, the coordination layer has to periodically monitor the system and acquire the devices' current physical condition, such as network consumption. Once the threshold is met, the coordination layer finds another appropriate instance of this node and reroutes the data stream to it. This naturally becomes a load-balancing system where the data stream is routed to different devices according to their network consumption. It should be noted that the mechanism works on every dynamic application scenarios and is not limited to the load-balancing application (e.g. constrain a software component to run in a certain location, let two components communicate when they are nearby, etc.).

The result of the load-balancing capability can be seen in Figure. 5 where a simple dashboard is shown to illustrate the whole working system in our example application. The Memory Usage column captures the memory consumption of the desktop computer (*152*) and the two embedded boards (*117* and *198*). The metric is reported in kilobytes. The Network Load Factor shows the input bandwidth consumption of the embedded boards in bytes per second. In those widgets to the right, camera footage (derived from a video source) is shown with individual car images extracted from the scene as well as the number of classified result per second. Recall that our application has two constraints on the neural network image classification component. One constraint limits its execution to the devices within a certain location (i.e. downtown) and the other specifies the threshold of network consumption to the maximum of 50,000 bytes per second. In the result, it is clearly seen that once the network load of device *198* rises to the 50,000 mark, the second device, *117* starts to receive the data stream (the sudden rise in the memory consumption of *117*). The number of results per second is dropped when the transition occurred but quickly rises back again.

Our lab-based application experiment presents the need for dynamic, periodic coordination executions in our class of applications. The coordination layer has to constantly observe the physical context of participating devices and delivers the appropriate combination of the software component instances so that the data streams are sent to the right destination, fulfilling the application's context-dependent constraints.

In large scale, real deployment, this proves to be costly. This is where our simulation network plays its role in studying the coordination performance in large scale setting. In the next section, we report our simulation results using Omnet++ network simulator.

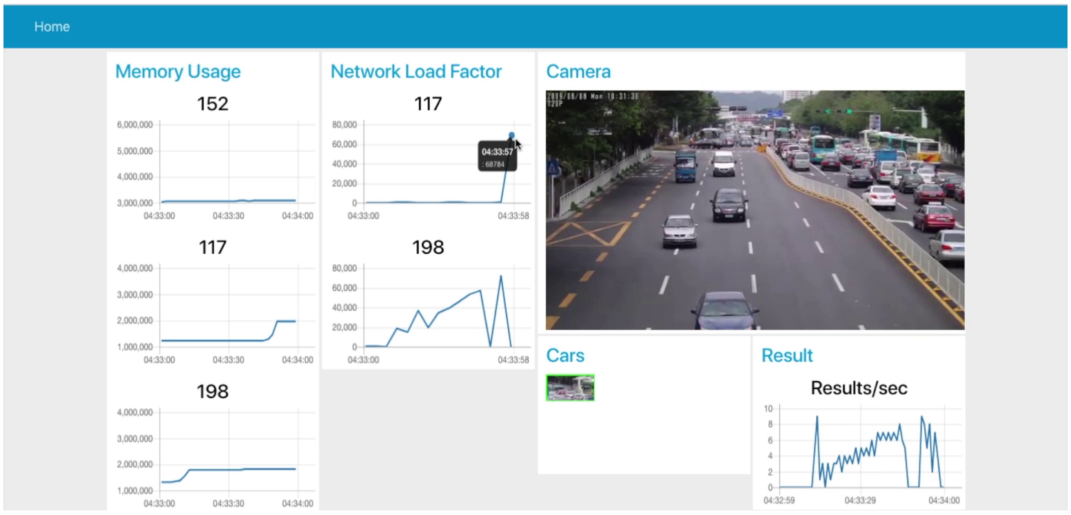


FIGURE 5 Dashboard.

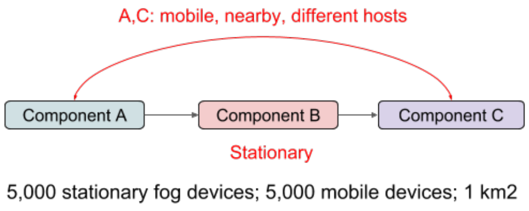

A,C: mobile, nearby, different hosts
Component A
Component B
Component C
Stationary
5,000 stationary fog devices; 5,000 mobile devices; 1 km2

FIGURE 6 Large scale simulated application model and configuration.

7 | LARGE SCALE SIMULATION

7.1 | Overview

While the lab-based setup has shown the scalability of the application model itself, a large scale experiment is still necessary as we would like to know how our coordination platform scales and supports the dynamic nature of the devices. To do this, a simulation has been independently developed using the Omnet++ network simulator. We use the simulation to study how our coordination model can support a large number of devices and how the level of system dynamics affects the coordination results.

The simulation network (Figure. 6) consists of 5,000 static fog devices and 5,000 mobile devices with variable speed. The simulated area is set to 1 kilometre square. We use a simplified fog application that consists of three component, a data source, an intermediate and a data sink component. The data source and sink components are constrained so that any combination of their instances are deployed on different devices and have to be close to one another. The closeness is defined to be within 300m of one another. The three software components are then distributed onto the 10,000 participating devices (both fog and mobile nodes), yielding a total of 30,000 component instances.

The application's performance metric (i.e. fulfilment level) is defined as the ratio between all the messages received by the sink component instances and all the messages sent by the source component instances, that is, the package delivery ratio. We collect this data in a fast moving scenario and a static immobile scenario.

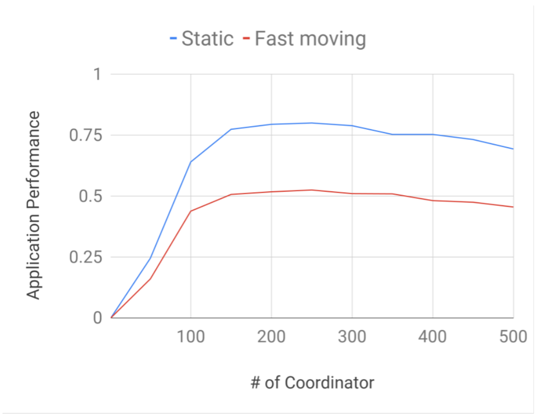


FIGURE 7 Effect of system dynamic to large scale coordination

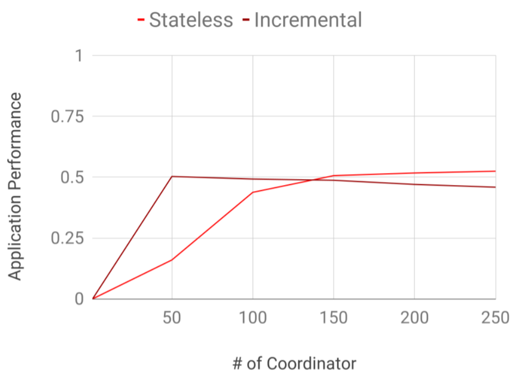


FIGURE 8 Stateless vs incremental coordination in fast moving scenario

7.2 | Simulation results

As described above, the application's performance metric is set to be the ratio between all the messages received by the sink components' instances and all the messages sent by the source components' instances. The two scenarios deployed in the simulation is static coordination where the devices do not move, and fast moving where the devices randomly change their speed and bearing.

We first study the effect of adding more coordinators to the coordination layer. Obviously, to support the increasing number of devices, more and more coordinators have to be added to the coordination layer. However, when more coordinators are added to the platform, the number of devices per coordinator is decreased. Up to a certain point, the individual coordinators might not have enough input devices in order to derive the optimal results, which are the combinations of the instances of the involved software components. Recall that each instance of a software component is equivalent to a single device that hosts it.

Figure. 7 illustrates this interesting findings. As the figure shows, when the number of coordinators increases, the application performance increases as well, but only to a certain point where it starts to degrade even if more coordinators are added.

The figure also shows the performance difference between a static and a dynamic setting. Again, it is obvious that when the system is static, the application performance is much higher than when it is dynamic. However, the performance trend stays the same whether it is static or dynamic. That is, the performance increases, peaks and degrades at around the same point as far as the number of coordinators is concerned. This shows that the optimal number of coordinators is almost orthogonal to the level of system dynamic. This is an interesting and notable observation because based on this fact, the system operator can roughly estimate the optimum number of coordinators required for a certain system without having to worry about the system's dynamic characteristic.

Secondly, we study how our incremental coordination strategy can support the system dynamic. Recall that our coordination platform periodically monitors the overall system context and calculates the combinations of software component instances. Whenever a new coordination result is produced, it is broadcast to all the participating devices so that their software component instances can update their peers. Sometimes this could be redundant as the existing combinations might still satisfy all the application's constraints. Whenever this happens, the existing communication links are disconnected and new links are formed. This leads to messages being dropped during their routes. To overcome this, our coordination platform keeps the previous coordination results and incrementally calculates the subsequent results so that if existing combinations still meet the application's constraints, they are excluded from the coordinator's input.

Figure. 8 shows the positive result from doing incremental coordination. As shown, the number of required coordinators can be reduced roughly from 150 to 50 while still achieve a performance peak. So only a third of the original number of coordinators is required to achieve the same peak in application's performance. Note that in this experiment, it is unnecessary to record the results for the static scenario as the coordination results are never changed. So it is dropped from the experiment.

8 | DISCUSSION

The case study presented in this paper allows us to extract several insights into the process of developing application in large scale fog computing, as well as some limitations of our application model.

8.1 | Component-level vs System-level, Application Scoping

Our first insight is, there is a clear distinction between component-level application vs system-level application. For example, the image classification component itself could be seen as a complete, independent application, or it could also be seen as a sub component of a larger application. It could also be further decomposed into smaller sub components that in turn, could become other complete, independent applications. Thus, we found that it is necessary to clearly define the scope and granularity of the application decomposition process. In our work, we take the developer's point of view as the reference point to define this distinction. That is, an independent, complete application is defined by whether it is created by an independent developer. Using this definition, if one developer is responsible for building the image classification module, such module becomes one complete application under that developer's point of view. However, when another developer is tasked with combining multiple components together, she is responsible for the interconnection of the components as the whole and therefore, this interconnection, cooperation among components becomes one complete application from her point of view.

Thus, when developing any large scale fog applications, the system-level developers have to ask questions such as; which components are available, and how do these components interconnect with one another. Consequently, the programming details pertaining to individual computing activities become irrelevant. For example, how to collect sensing data from one device is completely irrelevant to system-level developers. Rather, the developers should be given a complete sensing module with different configuration that can be dynamically applied based on the host devices. This is when the coordination aspect of an application plays its important role.

8.2 | Task Assignment Model vs Participatory Model

Our second insight is that there is a clear distinction between the task assignment model and the participatory model in fulfilling the application's requirement. In the task assignment model, the application can be represented by a set of tasks to be executed. The tasks could depend on one another. However, the important characteristic of the task assignment model is that the assignment of tasks to the hosting devices is generally static, except for failure cases where tasks can be reassigned. This means the system and its devices should be static. In a more dynamic system, the task assignment model needs to constantly keep track of available resources in the system, therefore the coordination overhead is generally very high. In turn, the task assignment model has an advantage when task completion need to be guaranteed.

In the participatory model, the system can be more dynamic as it depends on the resource contribution of participating devices from time to time without a fixed commitment. This is somewhat similar to the crowd-sourcing application model where people (or their devices) are recruited to solve a particular problem. However, in this type of application, people (or their devices) often connect to a centralised crowd-sourcing platform to solve advertised tasks (or aggregate devices' data)[13], this eliminates the need to coordinate the execution and cooperation of individual participants. In our work, devices contribute the application's sub components that they can run and the coordinator connects and coordinates the contributed components together to fulfil the application's logic. In large scale, dynamic fog computing applications, we believe this is the appropriate model for the application development process.

8.3 | Limitations

There are a number of limitation with our current approach. Currently our application model does not fully address the configuration side of the software components. Each independent application comes with its own set of configurations that dictates the application specific behaviour under certain circumstances. This configuration aspect has not been studied in this work. We envision that the coordination platform has to coordinate not only the cooperation of the sub components but also their individual configuration in a dynamic way that is suitable for the run-time condition. For example, in our lab-based set up, the camera component usually comes with a set of configurations to adjust its execution, such as the captured frame rate or resolution. This configuration data, while pertaining to the component-level application, also affects the composition of large scale application.

Thus, the application platform should allow the developers to also dynamically coordinate the configuration of individual sub components based on the underlying hosting devices. One possible solution is to let developers define a set of configurations for certain sub components so that at run-time, the system can selectively choose the right configuration for their execution. Since the study into this subject can easily grow out of the scope of this paper, we leave this for our future work.

Another limitation of the case study is the lack of a real world test bed to test drive the application platform. This is particularly difficult to achieve in an academic setting and we acknowledge that there will be more work to be done to deploy the model in the real world scenario. Currently we are working closely with Keio University in Japan under the EU-Japan joint project Bigclout[14] to deploy the DNR platform to Keio's smart city infrastructure in Fujisawa city. In our current trial, we employ a fleet of garbage trucks in Fujisawa city as our fog computing platform. Each garbage truck has a camera onboard to capture video footage of the road segments they travel to. These video streams are analysed to classify the lane markers on the road surface to see if a repaint needs to be done. Our DNR platform allows the city developers to quickly build and deploy such application on top of the fleet and associated city infrastructure. While we are still in the data collection and experimentation phase, the details of the application and system architecture have been published in[15].

Lastly, while our work addresses a number of critical issues in the design and development of large scale applications, there are several that are outwith the scope of our work. One of these issues is the physical communication layer that interconnects devices and software components together. In this paper one simplifying assumption we make is that the communication layer is provided so that inter-device connection can be established between any pair of devices efficiently and at low cost. With the advent of new communication technologies such as 5G, we believe this is soon to be within reach for most fog computing applications.

9 | RELATED WORKS

While research into fog computing has gathered momentum in recent years, application modelling and development techniques are still the subject of ongoing research. Issues to be addressed include: how fog applications are modelled and built, how to run and maintain them in the widely distributed fog computing infrastructure. A popular research approach is to reuse existing techniques from traditional Internet applications for fog applications such as the use of containers or micro-services following the Service Oriented Architecture (SOA).

For example, the authors in[16] [17] [18] discussed the problem of orchestrating and management of containers in a fog computing infrastructure. Interesting observations were that container orchestration in fog computing has to take into account specific characteristics of fog infrastructure such as device join/leave, mobility support, or dynamic workloads. In our work, we address the dynamic nature of the system, which is the broader case of those requirements. The other common idea was that the orchestration has to take into account the device's physical capabilities. We abstract this idea into a more general notion of physical context that encapsulates the device's physical properties such as current location or memory usage level.

Exploiting the micro-services architecture in fog computing is another approach, notably the idea of Osmotic Computing[19] in service orchestration and deployment. Micro-services and SOA could be seen as a way to decompose application into constituents that can be distributed to the infrastructure. Osmotic computing incorporates the notion of device capabilities into the micro-service definitions, deriving the notion of MicroElements as the abstraction for the computing units.

While these existing works about application decomposition aligns well with our design choice where we incorporate device's capabilities into the application modelling, they do not provide a unified programming model for the fog application developers. Thus, a *programming-in-the-large* mindset[20] is still relevant to build and deploy applications in large scale fog computing environment.

Some classes of applications are inherently distributed, such as IoT and Wireless Sensor Network (WSN) applications. For example, Hiesgen et al.[21] and Sivieri et al.[22] followed an actor-based approach in modelling IoT applications. Elgamal et al.[23] studied the assignment of tasks within an application on IoT devices. Riliskis et al.[24] developed Ravel, an application model for the IoT based on the Model-View-Controller design pattern. Cherrier et al.[25] proposed a finite state machine-based approach in modelling IoT applications.

While these works addressed the distributed nature of IoT applications, most of them target small scale settings where the deployment of components onto participating devices does not produce a multitude of the component's instances. As a result, many unique characteristics of large scale fog computing system such as its dynamic nature or the involvement of physical context in the application model were not addressed.

Research into WSN has also generated a large number of programming models and languages for distributed embedded sensor devices[26]. However, a key different between these research bodies and our target class of applications is that the software components within WSN applications tend to be homogeneous while our fog applications usually involve a variety of software components. Moreover, most of existing WSN applications are deployed to static infrastructure where the context fluctuation (e.g. mobility) of nodes is minimum.

Apart from application decomposition and orchestration, dynamic coordination among participating devices and software components is another research challenge in the dynamic fog infrastructure[27]. Ha et al.[28] presented a distributed computing framework for fog computing, notably the Fog Tracker implementation. Fog Tracker provides a similar functionality to our dynamic coordination platform where device's physical context is periodically acquired for the dynamic coordination purpose. However, the work does not target large scale deployment where multiple instances of each software component are deployed and the physical context of the underlying fog devices plays an important role.

10 | CONCLUSION

In this paper, we present a case study of applications development and deployment in fog computing using the open source project DNR. We implement our case-study scenario and build a prototype fog application using DNR. We further validate our approach by building and running a large scale fog computing study using the network simulator Omnet++ where we address the problem of scalability and dynamic nature of the fog computing environment at large.

Our lab-based experiment with DNR validates our belief that large scale fog applications require a composition based approach in which exogenous co-ordination takes on the system level task of managing large scale coordination of software components to meet application level constraints. Our work has also highlighted the need to augment off-the-shelve components to allow them to participate in large scale fog applications and we have shown an initial technique to address this issue.

The large scale simulation triggers some interesting observations, notably the effect of dynamic characteristic on the application performance and its relationship with the number of coordinators employed. Briefly, more coordinators in the system does not always guarantee better application performance and the optimum number of coordinators is not affected by the system dynamics.

We plan to study further issues related to co-ordination scalability, as well as those highlighted in the limitations section such as handling component constraints in a system level composition.

11 | ACKNOWLEDGEMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 61671088, in part by Moovee Innovations Inc., in part by the Canadian Natural Sciences and Engineering Research Council through the Industrial Postgraduate Scholarship (Applicant No. 486401), and in part by the H2020-EUJ-2016 EU-Japan joint research project, BigClouT, under Grant 723139.

References

1. Lin Jie, Yu Wei, Zhang Nan, Yang Xinyu, Zhang Hanlin, Zhao Wei. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. *IEEE Internet of Things Journal.* 2017;4(5):1125–1142.

2. O'Leary Nick, Conway-Jones Dave. *Node-RED: Flow-based programming for the Internet of Things.* https://nodered.org. Accessed December 1, 2018; 2018.

3. Giang Nam Ky, Lea Rodger, Blackstock Michael, Leung Victor C. M.. Fog at the Edge: Experiences Building an Edge Computing Platform. In: 2018 IEEE International Conference on Edge Computing (EDGE):9–16; 2018.

4. Arbab Farhad. What do you mean, coordination. *Bulletin of the Dutch Association for Theoretical Computer Science (NVTI).* 1998;(March '98):11–22.

5. Giang Nam Ky, Lea Rodger, Leung Victor C. M.. Exogenous Coordination for Building Fog-Based Cyber Physical Social Computing and Networking Systems. *IEEE Access.* 2018;6:31740–31749.

6. Giang Nam Ky. *Distributed Data-Flow Coordination Platform Based on Node-RED.* https://github.com/namgk/dnr-editor. Accessed Jan 26, 2019; 2019.

7. Lea Rodger. Smart Cities: an overview of the technology trends driving smart cities: IEEE Technology Trend Paper. 2017;:1–16.

8. Bonomi Flavio, Milito Rodolfo, Zhu Jiang, Addepalli Sateesh. Fog Computing and Its Role in the Internet of Things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing:13–16; 2012; Helsinki.

9. Krajzewicz Daniel, Erdmann Jakob, Behrisch Michael, Bieker Laura. Recent Development and Applications of {SUMO - Simulation of Urban MObility}. *International Journal On Advances in Systems and Measurements.* 2012;5(3&4):128–138.

10. Giang Nam Ky, Blackstock Michael, Lea Rodger, Leung Victor C. M.. Developing IoT Applications in the Fog : a Distributed Dataflow Approach. In: Internet of Things (IOT), 2015 5th International Conference on the:155–162IEEE; 2015; Seoul.

11. Pulli Kari, Baksheev Anatoly, Kornyakov Kirill, Eruhimov Victor. Real-time Computer Vision with OpenCV. *Commun. ACM.* 2012;55(6):61–69.

12. Jia Yangqing, Shelhamer Evan, Donahue Jeff, et al. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093.* 2014;.

13. Bozzon Alessandro, Brambilla Marco, Ceri Stefano, Mauri Andrea, Volonterio Riccardo. Pattern-based specification of crowdsourcing applications. *Web Engineering.* 2014;8541:218–235.

14. Project BigClouT. *Bigclout: Big data meeting Cloud and IoT for empowering the citizen clout in smart cities.* http://bigclout.eu. Accessed Jan 26, 2019; 2019.

15. Kawano Makoto, Yonezawa Takuro, Tanimura Tomoki, et al. CityFlow: Supporting Spatial-Temporal Edge Computing for Urban Machine Learning Applications. In: Urb-IoT 2018 - 3rd EAI International Conference on IoT in Urban Space; 2018.

16. Wobker Cecil, Seitz Andreas, Mueller Harald, Bruegge Bernd. Fogernetes : Deployment and Management of Fog Computing Applications. In: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium:1–7; 2018.

17. Hoque Saiful, Brito Mathias Santos De, Willner Alexander, Keil Oliver, Magedanz Thomas. Towards Container Orchestration in Fog Computing Infrastructures. *Proceedings - International Computer Software and Applications Conference.* 2017;2:294–299.

18. Yigitoglu Emre, Mohamed Mohamed, Liu Ling, Ludwig Heiko. Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing. *Proceedings - 2017 IEEE 6th International Conference on AI and Mobile Services, AIMS 2017.* 2017;:38–45.

19. Carnevale Lorenzo, Celesti Antonio, Galletta Antonino, Dustdar Schahram, Villari Massimo. From the Cloud to Edge and IoT: a Smart Orchestration Architecture for Enabling Osmotic Computing. *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA).* 2018;:419–424.

20. DeRemer Frank, Kron Hans. Programming-in-the large versus programming-in-the-small. In: International conference on Reliable software, vol. 10: :114–121; 1975.

21. Hiesgen Raphael, Charousset Dominik, Schmidt Thomas C. Embedded Actors âĂŞ Towards Distributed Programming in the IoT. In: Consumer Electronics Berlin (ICCE-Berlin), 2014 IEEE Fourth International Conference on:371–375; 2014.

22. Sivieri Alessandro, Mottola Luca, Cugola Gianpaolo. Building Internet of Things Software with ELIoT. *Computer Communications.* 2016;0:1–13.

23. Elgamal Tarek, Sandur Atul, Nguyen Phuong, Nahrstedt Klara, Agha Gul. DROPLET : Distributed Operator placement and Resource Provisioning for IoT applications spanning Edge and Cloud Resources. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD).* 2018;:1–9.

24. Riliskis Laurynas, Hong James, Levis Philip. Ravel: Programming IoT Applications As Distributed Models, Views, and Controllers. *Proceedings of the 2015 International Workshop on Internet of Things Towards Applications.* 2015;:1–6.

25. Cherrier Sylvain, Ghamri-Doudane Yacine M., Lohier Stephane, Roussel Gilles. D-LITe: Distributed Logic for Internet of Things Services. In: Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing:16–24IEEE; 2011.

26. Mottola Luca, Picco Gian Pietro. Programming wireless sensor networks: fundamental concepts and state of the art. *ACM Computing Surveys.* 2011;43(3):1–51.

27. Shi Weisong, Cao Jie, Zhang Quan, Li Youhuizi, Xu Lanyu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal.* 2016;3(5):637–646.

28. Jeong Taeyeol, Chung Jaeyoon, Hong James Won-Ki, Ha Sangtae. Towards a distributed computing framework for Fog. *2017 IEEE Fog World Congress (FWC).* 2017;:1–6.

AUTHOR BIOGRAPHY

nam.jpg

Nam Ky Giang is currently a PhD candidate at the Electrical and Computer Engineering department of the University of British Columbia. Prior to his PhD journey, he obtained a Master degree from Korea Advanced Institute of Science and Technology (KAIST) from 2011 to 2013, and a Bachelor of Engineering from Hanoi University of Science and Technology from 2005 to 2010. His research interests involve programming and coordination models and languages for large scale distributed systems such as edge or fog computing. He is also keen on software development and engineering, and is the author of Distributed Node-RED project, an open source platform for building and coordinating distributed applications.

rodger.png

Rodger Lea received the Ph.D. degree in computer science in 1989 from Lancaster University, U.K. Following work at Chorus Systems and HP Labs, Dr Lea moved to Sony's Tokyo labs in 1995 to lead work on multimedia operating systems and mixed reality TV. This work led to the creation of the VRML2.0 specification which subsequently contributed to the X3D and MPEG 4 BIFS standards. In 1997 Dr. Lea founded Sony's Distributed Systems Lab in San Jose, CA. Under his leadership this lab generated over 100 patents and delivered a number of key technologies to Sony product groups. Dr. Lea was the lead architect of the IEEE1394 based Home AV architecture (HAVi), created the HAVi consortium and led Sony's work on creating prototype media centric home network applications including a personalised home media server. Parts of this work were subsequently adopted by the European TV standard DVB MHP and the US OpenCable. As a Vice President, responsible for R&D activities, Dr Lea operated at the most senior level of the company and was involved in strategic planning and relationships including Sony's activities with Microsoft, AT&T and a variety of industry forums. He spun out two companies from Sony, supported in-house venturing and led technical due diligence for Sony's US venture capital fund. In 2006 Dr. Lea rejoined Lancaster University as an EPSRC research fellow and currently holds the title of Senior Research Fellow. In addition to his industrial work, Dr Lea has been an active member of the research community, publishing over 70 papers and a number of books. In addition to his standards activities (DVB MHP, ETSI VRML, ATSC DASE, W3C W3D, OSGi, OCAP, Java TV, and MPEG4), he helped setup conferences such as EuroSSC and UbiSys and been a PC member, reviewer and editorial board members on a number of international conferences and journals including IEEE PerCom, Pervasive, iThings, PerWare, IEE DSEJ (Editorial Board), ACM IMWT, USENIX OSDI, IFIP. Most recently he is a board member and co-chair ACM Middleware and vice chair of the IEEE ad hoc committee on Industry Engagement.

vleung.jpg

Victor C.M. Leung received the B.A.Sc. (Hons.) degree in electrical engineering from the University of British Columbia (UBC) in 1977, and was awarded the APEBC Gold Medal as the head of the graduating class in the Faculty of Applied Science. He attended graduate school at UBC on a Canadian Natural Sciences and Engineering Research Council Postgraduate Scholarship and received the Ph.D. degree in electrical engineering in 1982.

From 1981 to 1987, Dr. Leung was a Senior Member of Technical Staff and satellite system specialist at MPR Teltech Ltd., Canada. In 1988, he was a Lecturer in the Department of Electronics at the Chinese University of Hong Kong. He returned to UBC as a faculty member in 1989, and currently holds the positions of Professor and TELUS Mobility Research Chair in Advanced Telecommunications Engineering in the Department of Electrical and Computer Engineering. Dr. Leung has co-authored more than 1100 journal/conference papers, 40 book chapters, and co-edited 14 book titles. Several of his papers had been selected for best paper awards. His research interests are in the broad areas of wireless networks and mobile systems.

Dr. Leung is a registered Professional Engineer in the Province of British Columbia, Canada. He is a Fellow of IEEE, the Royal Society of Canada, the Engineering Institute of Canada, and the Canadian Academy of Engineering. He was a Distinguished Lecturer of the IEEE Communications Society. He is serving on the editorial boards of the IEEE Transactions on Green Communications and Networking, IEEE Transactions on Cloud Computing, IEEE Access, Computer Communications, and several other journals, and has previously served on the editorial boards of the IEEE Journal on Selected Areas in Communications âĂŞ Wireless Communications Series and Series on Green Communications and Networking, IEEE Transactions on Wireless Communications, IEEE Transactions on Vehicular Technology, IEEE Transactions on Computers, IEEE Wireless Communications Letters, and Journal of Communications and Networks. He has guest-edited many journal special issues, and provided leadership to the organizing committees and technical program committees of numerous conferences and workshops. He received the IEEE Vancouver Section Centennial Award, the 2011 UBC Killam Research Prize, the 2017 Canadian Award for

Telecommunications Research, and the 2018 IEEE ComSoc TGCC Distinguished Technical Achievement Recognition Award. He co-authored papers that won the 2017 IEEE ComSoc Fred W. Ellersick Prize, the 2017 IEEE Systems Journal Best Paper Award, and 2018 IEEE ComSoc CSIM Best Journal Paper Award.

How to cite this article: N. K. Giang, R. Lea, and V. Leung (2018), Developing Applications in Large Scale, Dynamic Fog Computing: a Case Study, *Software: Practice and Experience*, *2018;00:1–6*.