IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON CYBER-PHYSICAL-SOCIAL COMPUTING AND NETWORKING

Received April 9, 2018, accepted May 22, 2018, date of publication June 6, 2018, date of current version June 29, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2844336

Exogenous Coordination for Building Fog-Based Cyber Physical Social Computing and Networking Systems

NAM KY GIANG[1], RODGER LEA[2], AND VICTOR C. M. LEUNG[1], (Fellow, IEEE)
[1]Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada
[2]School of Computing and Communications, Lancaster University, Lancaster LA1 4YW, U.K.

Corresponding author: Victor C. M. Leung (vleung@ece.ubc.ca)

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.

ABSTRACT With the proliferation of smart embedded devices, cyber physical social computing and networking systems (CPSCN) are emerging as a next generation of social networks. Unlike traditional social networks that run on cloud-based infrastructure, CPSCN systems usually depend on a large number of distributed, heterogeneous devices, such as mobile phones, smart vehicles, or network access points. These computing resources, which are often referred to as fog computing systems, provide a gateway to the physical world, and thus offer new possibilities for social applications. Unfortunately, building CPSCN systems that leverage fog computing infrastructure is not straightforward. Significant challenges arise from the large scale distribution of computing resources over a wide area, and the dynamic nature of multiple, possibly mobile, hosts. In this paper, we extend our previous work on a distributed dataflow programming model and propose an application platform for realizing CPSCN systems. A key aspect of our work is the development of an exogenous coordination model, which exhibits a separation of concern between computation and communication activities, and helps resolve some of the challenges brought about by the dynamic and large scale nature of CPSCN systems.

INDEX TERMS Cyber physical social computing and networking, fog computing, Internet of Things, exogenous coordination, data-flow.

I. INTRODUCTION

As the number of smart embedded devices increases, computing infrastructure is more intimately tied to the physical world, often capturing high volumes of real world sensor data. This not only gives us more insight into our surroundings but also opens a communication channel between humans and physical objects, thus enabling a new class of systems that we called Cyber Physical Social Computing and Networking systems (CPSCN). These systems, where the presence of physical entities plays a significant role, often depend on fog computing - a distributed computing platform that encompasses devices and servers from the edge, through the network, to the cloud.

Thanks to its widely distributed nature, fog computing can support the large numbers of participating smart devices in CPSCN applications in a scalable and efficient way. In particular, by distributing workloads across network edges and the cloud it offers scalability and flexibility. However, while fog computing enables a more efficient and time sensitive model for executing many tasks, it comes with its own challenges that require some new approaches to the development process. Particularly, these challenges stem from the intrinsic characteristics of fog computing systems and the CPSCN: widely distributed computing resources and rapidly changing contexts across a large number of compute nodes.

Since the computing resources in CPSCN systems are often widely geographically distributed, it requires a different application model and software engineering process as compared to traditional social networks. While existing networks also consist of distributed components, the distribution of these components on the computing infrastructure is usually transparent to the developer. This is because in a centralized data center, application developers do not worry about which machine hosts computation. In contrast, in fog computing systems where the computing and communication infrastructure is not centralized, or homogeneous, developers need to specifically cater to these differences. In CPSCN

systems, we argue that the application model needs to take into consideration the underlying computing infrastructure and its associated contextual information as an integral part of an application. As a simple example, it should allow the developers to choose a particular physical location where a computation activity should run.

Second, the large scale characteristic of fog computing infrastructure and the complexity of CPSCN systems suggest that we should rethink the *separation of concerns* between communication and computation activities. Traditionally, these activities are often mixed together in the applications' codes. With more complex CPSCN applications, an exogenous coordination model, which explicitly separates communication and computation, could provide a more complete solution [1]. In an exogenous coordination model [2], the process of developing application components is completely separated from the communications or co-operations among them. Every component is modeled as a computation activity with its own input and output ports. At development, a computation activity does not have knowledge about the peers it will interact with. All it does is to take the data from its inputs, do the computation and place the result on its outputs. The developers design the application by describing how these independent computation activities co-operate with one another. At runtime, the co-operation among these activities is then coordinated by an external coordinator. That is, depending on the application design, the coordinator takes data from the outputs of one activity and place them on the inputs of other activities. In other words, the coordinator ''wires'' or ''routes'' data among the computation activities so that the developer's application design or requirements are met. When the system is large and composed of many devices, an exogenous coordination model relieves the developers from the communication details and helps them to focus on the abstract application model.

In summary, we propose a new approach to building CPSCN systems that embraces all these fog-based characteristics. Our key contributions are as follows:

- We show that contextual information of a computation activity plays an important role in designing an application platform to support CPSCN systems.
- We demonstrate that an exogenous coordination model can help to maximize the reusability of *off-the-shelf* software components in building complex CPSCN applications in a fog computing infrastructure.
- We propose our design of a scalable context-dependent exogenous coordination platform and show that it can greatly facilitate the development of large scale CPSCN applications.

II. CPSCN SYSTEMS USING FOG COMPUTING INFRASTRUCTURE

Unlike traditional social networks running in the cloud, CPSCN systems using a fog computing infrastructure can leverage many advantages provided by the distributed and *close-to-the-edge* nature of the computing platform.

For example, consider a hypothetical CPSCN scenario where traffic information, which is computed from sensor streams (e.g., video streams from dashcams) generated by vehicles, is shared back to other vehicles based on their proximity to one another. This CPSCN scenario involves a large number of smart, connected motor vehicles, many data processing units that are distributed over a large area such as the whole city and human social networks where the computed traffic information is shared. The data processing units could be mounted on the street lamp posts, as well as be installed in communication access points (APs), such as Wi-Fi APs or cell towers. The proximity to data sources where data are generated allows these processing units to deliver fast responses or notifications while saving network bandwidth. Thus, whenever a traffic event occurs, a notification can be sent to all the vehicles in close proximity to the incident and at the same time, posted on traditional social networks (e.g., Twitter).

In this class of CPSCN applications, resource management and application development are still very challenging as it is unclear how to coordinate the operation of such a large number of smart devices in a large scale deployment [3]. Two distinguishing characteristics of fog computing systems that very much influence the process, are the geographic distribution and dynamic nature of the participating entities.

A. GEOGRAPHIC DISTRIBUTION OF FOG COMPUTING INFRASTRUCTURE

Large scale CPSCN applications may range from industrial applications spanning factories to smart cities spanning metro areas or larger geographical groupings. This large scale geographic distribution has several consequences that influence the way applications are developed.

First, fog computing resources generally communicate over heterogeneous networks that involve 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 (Internet Protocol) addresses vs. behind NAT (Network Address Translation)). This heterogeneity makes it more difficult for inter-device communication. In large scale fog applications, these communication details should not hinder the development of the application in general. Therefore, the developers should be provided with enough programming tools and primitives that allow them to focus only on the application logic.

Second, due to the large scale distribution of computing resources, their physical locations, or in general their physical context becomes an important factor in the fog computing application model. Returning to our smart city vehicle example, although there are many instances of the application running in vehicles, a particular traffic update might only be sent to a subset of vehicles that are directly affected by the incident. That is, in addition to specifying the location where an application component should be deployed, a developer might want to restrict the interaction between two components based on their locations or other context information.

B. DYNAMIC NATURE OF FOG COMPUTING SYSTEMS

Due to the close bonding with the physical world, fog computing systems often exhibit a highly dynamic nature, in terms of both load fluctuation and context changes (e.g., location changes when fog 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 fog computing environment. This is partly because fog computing resources are not as centralized and readily available as cloud computing resources so that it is harder to scale horizontally, and partly because the heterogeneous networking environment makes it difficult to locate resources for the load balancing task.

In addition to load fluctuation, changes in physical context, 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 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. Furthermore, the application model should explicitly distinguish contextual data and application data so that the coordination of computation activities is driven by the contextual data and ensuring application data are communicated properly. Returning to our previous vehicle example, the application data in this case is the sensing data generated by vehicle and road side sensors. In order to communicate or send this data to the appropriate recipients, the contextual data - vehicle locations - are used to coordinate the communication, that is, to route the sensing streams to the appropriate end-points.

III. CONTEXT-DEPENDENT EXOGENOUS COORDINATION AS AN INTEGRAL DEVELOPMENT MODEL

To develop complex CPSCN systems like the one illustrated in SectionII, we need an appropriate application model that eases the development process from design, through development to deployment. Traditionally, in many existing Internet-based social networks, the whole application is developed and run in the cloud. In fog computing systems, the same process becomes more complex as the computing resources are both dynamic and unpredictable, leading to two important design requirements; the ability to decompose the applications into independent components and the ability to compose and coordinate those components.

Although modular application design has been well studied, as well as the associated need of inter-process communication, to date there has been less study on how to coordinate the participating components when they are part of a fog based CPSCN system. In particular, since the components run on a dynamic fog computing platform, important research questions arise as to the correctness of the application which no longer depends solely on the correctness of the computation activities, but also the physical context they are running within.

As indicated, our approach to answering these research questions is based on the notion of an exogenous coordination model. In [1], we suggested that this was a promising application model for fog applications. In this work, we develop that idea and explore the details of such an approach and its implications for the underlying fog platform.

With an exogenous coordination model, a distributed application in the fog consists of independent software components that cooperate with each other in fulfilling the application's requirements. The communications amongst them is externally managed by a coordination entity (coordinator) and explicitly separated from the computation activities. That is, each software component is modeled as a computation activity with well-defined communication ports for inputs and outputs. Computation proceeds by taking incoming data on the input port, carrying out processing and placing the results on the output port. Communications between components are realized by the external coordinator which arranges for data from one output to be routed to another input. In other words, the coordinator *wires* the components together using the underlying networking infrastructure.

Our experiences with the exogenous coordination concept in CPSCN, has highlighted a second, related issue. Coordinating distributed computation requires an understanding of the physical context that computation operates within, and this physical context needs to be separated from the sensor data generated by said computation.

For example, in our hypothetical CPSCN application described in Section II, it is common practice to mix the physical location data with the sensor streams so that the applications can take advantages of the location information. However, by mixing these two, it is difficult to maintain the reusability of the application design. This is because the developers have to rewrite the code to extract the context information from the data whenever they want to derive new context-dependent application logics.

In our context-dependent exogenous coordination model, we treat the contextual data separately from the actual application data. While the application data flow between software components' ports, the contextual data is used to coordinate how these components communicate with one another. For instance, based on the location of the sensor streams, we can coordinate the communication among two software components so that they only communicate when they are *nearby*.

IV. SYSTEM DESIGN

To validate our approach, we have developed a system model that supports context-dependent exogenous coordination and have developed a design (and implementation) for a system platform supporting the model.

A. CONTEXT-DEPENDENT COORDINATION PRIMITIVES

In our exogenous coordination model, a CPSCN application comprises a large number of participating devices, denoted as D. Each participating device $Di \in D$ has a set of properties P that defines its context, such as location, computing

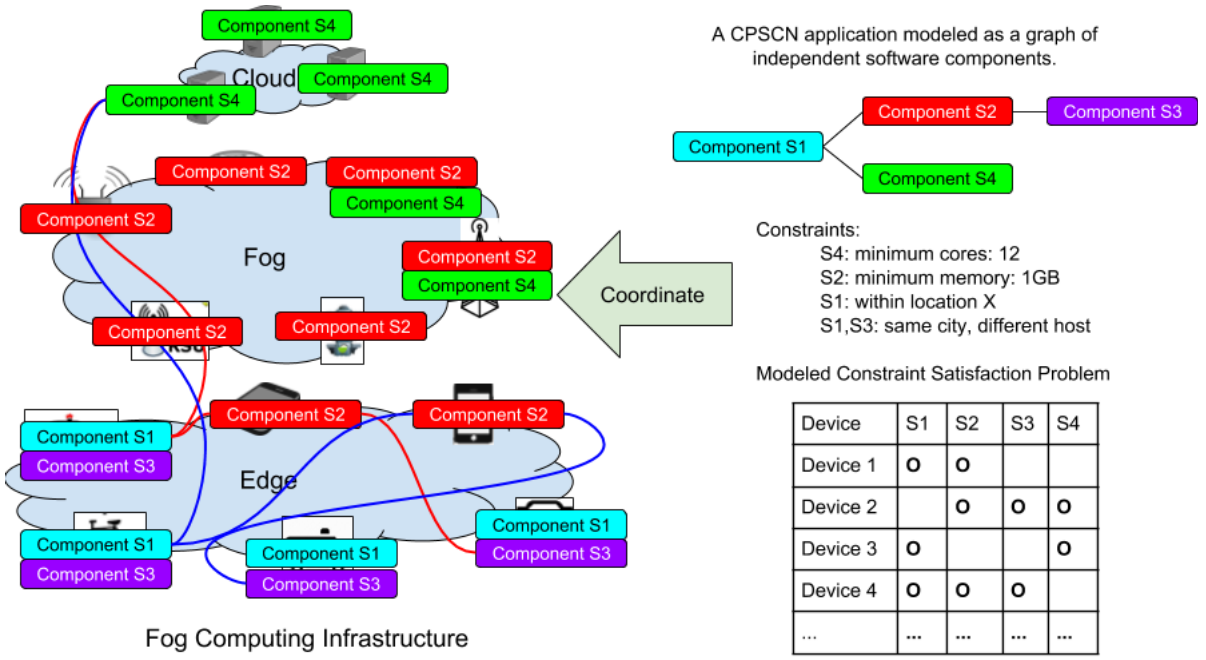


FIGURE 1. **Coordinating software components in fog-based CPSCN applications.**

resources, etc. Each of these properties contributes to the dynamic nature of the system in general.

The CPSCN application also involves a set of software components S, each of which is modeled as a computation activity with input and output ports. Each application in this CPSCN is then represented by a dependency graph of S, showing how data are originated, processed and consumed.

We define function $V : D \rightarrow P$ that captures the value of property set P of a device D, which normally represent the context of D. $Vx : V(D) \rightarrow Px$ is another function that extracts a single property x from the set of device properties $V(D)$. Function $F : S \rightarrow D$ to select all the devices that are capable of running a given software component.

The CPSCN application also includes a set of constraints C, represents the application-level requirements on the software components set with regard the devices' properties P.

We propose a context-dependent coordination primitive $\gamma(Si)_P$ that defines a subset of C called *context-dependent* constraints. In essence, *context-dependent* constraints are restrictions on the execution of a particular software component $Si \in S$ with regard property set P. $\gamma(Si)_P$ is formulated as follow:

$$\begin{aligned}&\gamma(Si)_P\\ &\quad\Longleftrightarrow \exists Dm \in F(Si),\\ &\quad satisfied(V_P(Dm))\end{aligned}$$

where *satisfied* is a function provided by the developers to define the required device properties for a software component to run. An example of such constraint is:

$$\begin{aligned}&onServer(S1)_{\{numberOfCores,freeMem\}}\\ &\quad\Longleftrightarrow \exists Di \in D,\\ &\quad(V_{numberOfCores}(Di) > 8)\\ &\quad\wedge(V_{freeMem}(Di) > 2)\end{aligned}$$

Here *onServer* is the constraint name, *numberOfCores* is a property of the participating devices D where the application can be deployed on. *assign* is a function that enables the execution of $S1$ on the device Di. This constraint restricts the execution of software component $S1$ on only the devices where the number of CPU cores is greater than 8 and the available memory is greater than 2GB.

It is the definition of $\gamma(Si)_P$ that makes our platform a context-dependent coordination platform because it takes the physical context of the device where the software components run into account.

More importantly, we propose a second novel type of context-dependent primitive $\Gamma(Si, Sj)_P$ that defines the other subset in C called *inter-component*. *Inter-component* constraints are restrictions of the cooperation between any two software components $Si, Sj \in S$ with regard property set P. $\Gamma(Si, Sj)_P$ is formulated as:

$$\begin{aligned}&\Gamma(Si, Sj)_P\\ &\quad\Longleftrightarrow \exists Dm \in F(Si), \quad \exists Dn \in F(Sj),\\ &\quad related(V_P(Dm), V_P(Dn))\end{aligned}$$

where *related* is a function provided by the developers to define the required context-dependent relationship between

any two or more software components. Some examples of such constraint are:

$$
\begin{aligned}
&nearby(S1, S2)_{\{lat,lon\}} \\
&\quad \Longleftrightarrow \exists Dm \in F(S1), \exists Dn \in F(S2), \\
&\quad |V_{lat}(Dm) - V_{lat}(Dn)| < M \\
&\quad \wedge |V_{lon}(Dm) - V_{lon}(Dn)| < N \\
&\quad sameHost(S1, S2)_{\{macAddr\}} \\
&\quad \Longleftrightarrow \exists Dm \in F(S1), \exists Dn \in F(S2), \\
&\quad V_{macAddr}(Dm) == V_{macAddr}(Dn)
\end{aligned}
$$

Here M, N denote the range that defines the *nearby* situation in one application. *lat* and *lon* are the latitude and longitude properties of the participating devices.

B. COORDINATION PROBLEM MODELING

Using our context-dependent primitives, the developers are now able to express different coordination constraints for their fog-based CPSCN applications. In order to fulfill those constraints so that the application can run correctly given all the contextual data, we model our coordination problem as a Constraint Satisfaction Problem as follow:

- Variable: the involved set of software components - $\{S1, S2, \ldots Sn\}$
- Domain: the participating devices - $\{D1, D2, \ldots Dm\}$
- Constraint: application-level requirements on the execution of S - $\{C1, C2, \ldots Ck\}$
- Goal: deliver the right assignments of S to D that satisfy all constraints C.

An illustration of CPSCN applications modeling is shown in Figure 1

Another important definition in our model is the system dynamic *SD* with regard to a property $Pi \in P$. *SD* is a variable that captures the change of *Pi* in time. Thus, *SD* can be seen as a measure to quantify the dynamic level of our system.

Since the system is dynamic, the process of fulfilling the constraints has to be re-executed periodically to keep up with *SD*. That is, the coordinator has to periodically reevaluate to deliver up-to-date assignments. Whenever a new assignment is published, all participating devices update their running software components and establish communication with external components as needed. It is worthwhile noting that in contrast to many existing constraint satisfaction problems, our approach requires the coordinator maintains a notion of past events. That is, to reduce the system level overhead of communication establishment, the coordinator needs to maintain its history and use it to minimize the number of reconfigurations so as to avoid unnecessary communication reconfiguration and associated overhead.

Further, the constraint solver has to deliver as many independent solutions as possible in a scalable way. This is because the large scale deployment of CPSCN, which usually encompasses a large number of devices and software components, makes it possible to have different assignments serving at different locations.

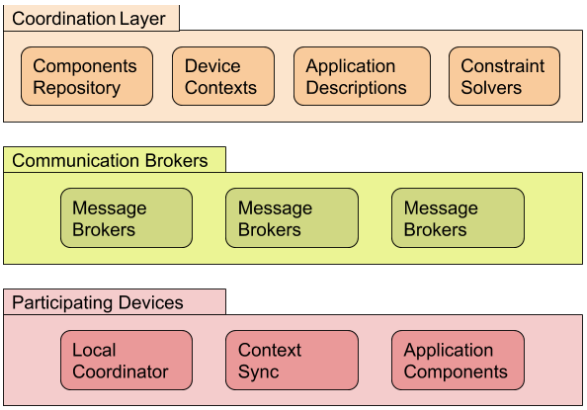


FIGURE 2. System architecture.

C. SYSTEM DESIGN

Figure 2 presents our system architecture design. Our system consists of three layers: a coordination layer, a layer of communication brokers and a layer of participating devices.

The use of communication brokers is necessary to facilitate the communication among devices which reside, sometime behind firewalls, or on a heterogeneous set of networks. Briefly the system leverages a publish/subscribe mechanism to glue software components together based on their names or identifications.

All participating devices are provisioned to take part in the CPSCN, e.g., by end users signing up to the network. While we do not explicitly study the incentive aspect of participants, we believe that a typical crowd-sourcing incentive model could be easily employed.

As seen in the previous Section, there can be many participating devices that are capable of running a particular software component. Thus, there are usually more than one instance of each software component running at the same time. These multiple instances of an software component all participate in the application's execution in a large scale setting. Periodically, participating devices "contribute" their software component instances that they are capable of running to a component repository, which is maintained at the coordination layer.

From time to time, based on this repository and the application descriptions (the dependency graph of components and the constraints), the coordination layer uses a distributed network of constraint solvers to calculate the assignment of software components to devices. Each solver is then responsible for a subset of the input. The results, which are the non-overlapping combinations of software component instances, are then synchronized back to the participating devices so that they can update their communication peers.

To fulfill the *inter-component* constraints, the coordination layer also needs to know the context of the involved software components. In order to do this, participating devices also periodically synchronize their contexts to the coordination layer. Thus, these device contexts are also fed into the constraint solvers to calculate the coordination result.

Again, each constraint solver (coordinator) is responsible for capturing the context of a subset of all the participating devices.

To carry out the constraint solving task, we adapt an off-the-shelf constraint solver, the Google OR Tools[1] for our coordination platform. Google OR Tools include an optimized implementation of constraint solvers that can be configured with a time-based deadline so that we can control the execution time of our coordinators. The tool also provides a solutions collector where multiple solutions can be collected after one execution instead of producing just one result. This feature meets one of the model needs identified above, i.e., a need to produce as many solutions as possible. To generate independent solutions, we add an extra step of filtering all the overlapped solutions from the tools' results. Each solution is a combination of software component instances where each one of them represents a physical device that can fulfill the application description.

When the participating devices receive the solutions, they extract the solution that involves them. From that they further extract the peer devices they should be connecting with. Finally they establish the communication with these peers to fully accomplish their roles.

As we described earlier, it is beneficial to retain some history of configuration events to minimize disruption as existing communications among devices might be rerouted due to new coordination decisions. In some cases this might result in unnecessary communication rerouting if existing communication paths still meet all the application constraints. In our current implementation, before reestablishing any communication path, the devices always check with their current peers to see if the existing combination is still valid. If it is still valid, the device simply ignores the coordination decision sent from the coordinator.

V. EXPERIMENTAL VALIDATION

We have conducted several experiments with our context-dependent exogenous coordination model using both a physical testbed and a simulation. Our physical testbed is based on the open source, Node-RED project from IBM[2] which we have extended with distribution support. Since we are interested in large scale deployments, which are obviously difficult to test with prototype research software, we have also developed a network model using the Omnet++ simulator in order to study our constraint solver at scale. This section describes our experiments with regard these two implementations.

A. DISTRIBUTED NODE-RED

In our previous work [4] we augmented Node-RED, an open source project that features a dataflow-based visual programming model, to create a distributed dataflow application development platform called Distributed Node-RED (DNR). The original Node-RED project itself does not have the capability to develop and execute distributed applications and runs as a single process on a single machine. Our DNR lets application developers specify the deployment constraints of an application flow so that once deployed, nodes in an application flow can be executed on multiple devices while still connecting to one another to deliver a fully functional application. Our original version of DNR was designed to be a deployment configuration model that exposes the underlying computing resources to the high-level application developers. Once deployed, the deployment configuration cannot be changed and thus, does not have the capability to execute a highly dynamic fog-based CPSCN, such as our example.

We have extended our DNR to add the coordination component and continuous context acquisition component described in this paper, so as to support the dynamic characteristics of fog computing systems. We have also implemented our context-dependent primitives into the platform so that the ability to express application constraints based on device's properties. These currently include resource-based constraints, communication constraints and temporal and spatial constraints.

The platform is deployed across a network of nodes ranging from hosted cloud infrastructure, through lab servers to edge devices such as Raspberry Pis or similar. A single server or cloud instance is designated as the coordinator and also serves as the component repository.

Since Node-RED comes with a comprehensive software component library, especially when there are existing components that can interact with traditional social networks (e.g., Facebook or Twitter), integrating these networks with physical devices is straightforward. Therefore, a CPSCN application can be easily provisioned. Furthermore, with its visual programming interface, the path from design to development has been shortened significantly. Based on this easy to use programming tool, we add our own coordination primitives that improves the seamlessness of the CPSCN application development.

Figure 3 shows the development canvas of our DNR platform. To build an application, the developer only needs to drag the software components from the left column to the canvas and *wire* them together. The application constraints can be added by using the Node Requirement button at the top right corner. The constraint definition dialog allows to define constraints based on physical location of the devices, or any of their computing resources, such as the number of CPU cores or the amount of free memory. For example, the developer can request that a certain node should only run within a particular geographical location. In cases where the node is deployed to a moving vehicle, it will only be activated if the vehicle passes through a certain area - defined by the geographical constraint.

B. OMNET++ SIMULATION

In addition to our DNR coordination platform, we created an Omnet++ network that simulates the coordination problem

[1] https://developers.google.com/optimization/
[2] http://nodered.org

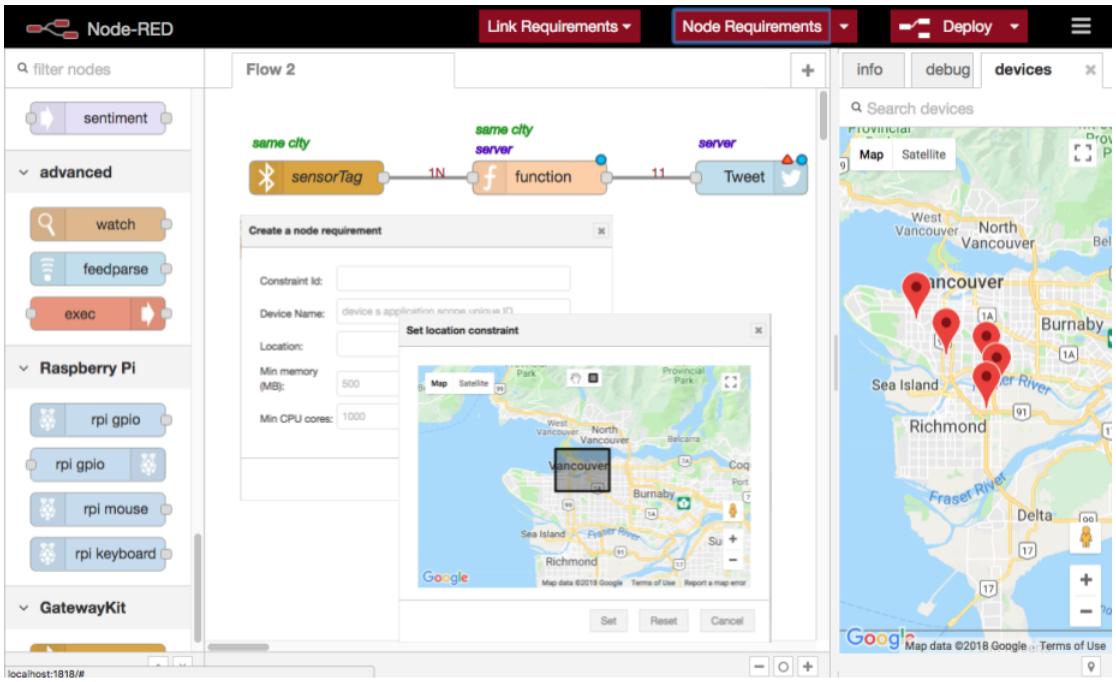


FIGURE 3. Building CPSCN applications using the DNR platform.

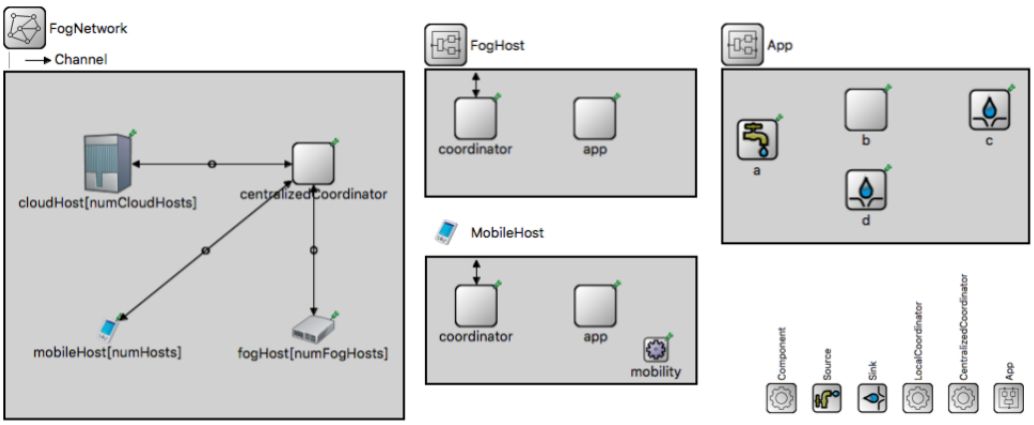


FIGURE 4. Omnet++ a network model for a fog-based CPSCN system.

in a large scale setting. Figure 4 shows our network design in the Omnet++ simulator. The network consists of one coordinator that oversees all network elements and a number of mobile and fog devices. The mobile devices are configured to move with certain speed and angle, which vary over time. Communication among devices are direct Omnet++ point-to-point channels with pre-configured communication delay. The simulation configuration can be seen in Table 1. The fog devices are configured to be stationary and randomly distributed over the simulation area. The simulation area is configured to be a large square of 1000 square meters.

The application is represented by four software components that are connected to one another. The application is constrained so that the data processing nodes (node b) can only be run by the stationary fog devices. It is also required that the data source and sink nodes cannot be located on the save device. Lastly, node a and node c are constrained to only participate in an application flow if they are 300 meters or closer to one another. Apart from the data

TABLE 1. Omnet++ simulation configuration.

Parameter	Value
moving speed	normal(mean: 10mps, std: 20mps)
change angle	normal(mean: 10deg, std: 20deg)
sensor stream rate	every 20s
nearby definition	within 300m in both axises
simulation area	1000 Sq m
simulation time	300s

processing node, node b, all other nodes run on mobile devices. Each device also has its own local coordinator which periodically updates the centralized coordinator with the device's location (i.e., x,y coordinates in Omnet++ simulation).

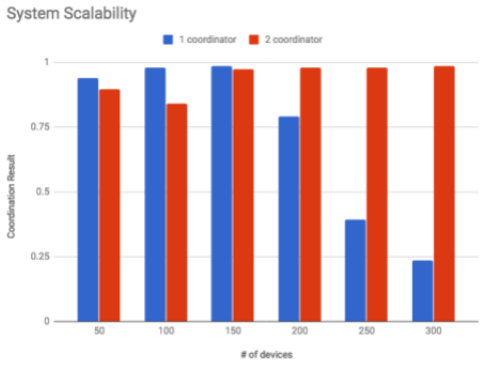


FIGURE 5. Scalability study.

We study how the coordination layer copes with the increase in the number of participating devices. Figure 5 shows the results of varying the number of devices from 50 to 300 for both categories (stationary fog devices and mobile devices). The horizontal axis shows the normalized ratio between the number of coordination solutions and the number of devices that need a solution. In this graph, 1 means all devices are getting their needed coordination solution. That is, they all know their peers to which they should send the data to. On the other end, 0 means none of the device knows where to send the data to. As the number of devices increases, the domain space of the constraint solver increases proportionally. When there is only one coordinator, the system is able to keep up with the increase in number of devices until 150 devices and started to quickly degrade. By adding a second coordinator to the coordination layer, the problem space of each coordinator is reduced in half, which results in a much better, scalable result. This result verifies that a distributed coordination layer is required to build a coordination platform for fog-based CPSCN applications.

VI. RELATED WORK

A. EXOGENOUS COORDINATION RESEARCH IN OPEN DISTRIBUTED SYSTEMS

Exogenous coordination [2] is itself not a new idea and a number of researchers have applied an exogenous coordination approach in distributed systems. One of the closest lines of work to ours is to coordinate Open Distributed Systems, which according to Agha [5], are systems whose functionality is not solely defined by the result of evaluating an expression. In these systems, the relative state of components, their locality and distribution of the computation, etc., also play an important role in the correctness of the application itself.

In the current state of the art in coordination systems however, the dynamic characteristics of the systems are often limited to the fluctuations in the availability of computation activities (software components). Many existing coordination models and languages only support this level of dynamic nature without taking into consideration constant changes in the physical context of the computation activities. These works also mainly focus on coordinating different communication patterns among the computation activities rather than coordinating the dynamic nature of the system. For example, context-dependent connectors have received much attention over the past ten years [6]–[8]. However, the notion of context referred to in these works is limited to the pending activities on each of the connector's ports. The goal of the coordination model is then to react to the presence of components on each port and to propagate this event further into the connector's composing channels. In [9], Lombide Carreton and D'Hondt addressed the problem of coordinating mobile distributed systems that have frequently changing network topology. While this is the closest work to our research as far as we are aware, it only addresses the intermittent connectivity of participating devices without taken into account their actual movement and changes in context. Moreover, it focused on a small scale mobile distributed systems with local network broadcasting used as the coordination medium among participating components.

B. APPLICATION DEVELOPMENT IN FOG COMPUTING

As a new computing paradigm, fog computing with its own characteristics requires an appropriate application model. Some recent attempts to address these problems have been demonstrated by Hong *et al.* [10], Skarlat *et al.* [11] and Cheng *et al.* [12]. Hong *et al.* [10] proposed a high-level programming model that allows the developers to specify how the application can be deployed to many fog computing nodes and supports dynamic scaling of the computing resources. This work demonstrated that the application model itself should be aware of the underlying computing node on which the application is running. On the down side, the proposed model does not explicitly break the application into sub components, which makes it impossible to have different deployment strategies without changing the application logic. Skarlat *et al.* [11] proposed that fog applications consisting of many services that have their own computation demands and are placed into the fog computing system in a way that satisfies the deadline requirements of the applications. While this work considered the componentization of the application code and studied the placement of those components, it does not take into account the contextual information of data. Thus, it does not greatly differ from the scheduling problem in traditional distributed systems. Cheng *et al.* [12] proposed a containerized dataflow-oriented application models with external configurations that specify how the data flows through the fog computing nodes. This could be seen as an important contribution toward exogenous coordination of fog-based applications as the execution of the individual software components is totally separated from the communications among them. However, the coordination mechanism is still simple and based on only the

deployment scope. The inter-node communication is also limited at either broadcast or unicast fashion.

While we specifically studied the task assignment problem in fog computing, code offloading and task scheduling are the other two closely related problems to our work. For example, Cheng *et al*. [13] proposed a optimization strategy for offloading code and scheduling task in wearable computing setting in order to maximize the user experience. The different between these and our addressed problem is that we let the developers to explicitly specify the application constraints rather than predefine the objective function. Based on the defined constraints, we derive as many non-overlapping assignments as possible rather than looking for the best one.

Dynamic characteristic of distributed systems has also been well studied. For example, Zhang *et al*. [14] proposed a deep learning model to predict cloud workload, so as to cope with the dynamic nature of load in cloud computing. Despite of existing efforts, supporting dynamic characteristic of fog computing infrastructure is still a challenging task as fog devices are less homogeneous, geographically distributed, and not controlled by a single entity.

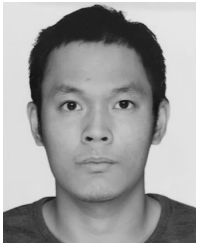
In our work, contextual information is incorporated into the application model, letting the developers to express application-level constraints based on the physical context of the computation activities (i.e., software components, dataflow nodes). In addition to that, our coordination model explicitly separate contextual data and application data, allowing for the separation of concern between communication and computation. That is, application data are used in computation activities while contextual data are used to coordinate the communication among those activities.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have explored the the requirements of fog computing application development as it relates to CPSCN systems and highlighted the issues raised by the scale and the changing context of the computing infrastructure. To address these issues, we have developed an exogenous coordination model that supports a reusable and scalable application model. We also show that coordination in a fog computing setting needs to incorporate the contextual data of the participating computation activities, and that the explicit separation of contextual data and application data makes it easier to separate communication and computation activities. To validate our approach, we have developed a fully functional research platform, DNR, which has been used for a number of small scale prototypes. In addition, to validate that our approach scales adequately, we have simulated a larger scale system using the Omnet++ system. In future work, we will address two areas; firstly, we plan to deploy larger scale real world CPSCN applications to validate our simulations and ensure the exogenous model meets the needs of developers; secondly we will extend the capabilities of our constraint solver, both in terms of constraints supported and in terms of scale and adaptability to system changes.

REFERENCES

[1] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. Leung, "On building smart city IoT applications: A coordination-based perspective," in *Proc. 2nd Int. Workshop Smart Cities, People, Technol. Data*, Trento, Italy, 2016, p. 7.
[2] F. Arbab, "What do you mean, coordination?" *Bull. Dutch Assoc. Theor. Comput. Sci.*, pp. 11–22, 1998.
[3] N. K. Giang, V. Leung, and R. Lea, "On developing smart transportation applications in fog computing paradigm," in *Proc. 6th ACM Symp. Develop. Anal. Intell. Veh. Netw. Appl. (MSWiM)*, 2016, pp. 91–98.
[4] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. Leung, "Developing IoT applications in the fog: A distributed dataflow approach," in *Proc. 5th Int. Conf. Internet Things (IOT)*, Oct. 2015, pp. 155–162.
[5] G. A. Agha, "Abstracting interaction patterns: A programming paradigm for open distributed systems," in *Formal Methods for Open Object-Based Distributed Systems*. Boston, MA, USA: Springer, 1997, pp. 135–153.
[6] S.-S. T. Q. Jongmans, C. Krause, and F. Arbab, "Encoding context-sensitivity in Reo into non-context-sensitive semantic models," in *Coordination Models and Languages* (Lecture Notes in Computer Science), vol. 6721. Berlin, Germany: Springer, 2011, pp. 31–48.
[7] M. Bonsangue, D. Clarke, and A. Silva, "Automata for context-dependent connectors," in *Coordination Models and Languages* (Lecture Notes in Computer Science), vol. 5521. Berlin, Germany: Springer, 2009, pp. 184–203.
[8] D. Clarke, D. Costa, and F. Arbab, "Connector colouring I: Synchronisation and context dependency," *Electron. Notes Theor. Comput. Sci.*, vol. 154, no. 1, pp. 101–119, 2006.
[9] A. L. Carreton and T. D'Hondt, "A hybrid visual dataflow language for coordination in mobile ad hoc networks," in *Coordination Models and Languages* (Lecture Notes in Computer Science), vol. 6116, D. Clarke and G. Agha, Eds. Berlin, Germany: Springer, 2010, pp. 76–91.
[10] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the Internet of Things," in *Proc. 2nd ACM SIGCOMM MCC*, 2013, pp. 15–20.
[11] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware fog service placement," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput. (ICFEC)*, May 2017, pp. 89–96.
[12] 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 Things J.*, vol. 5, no. 2, pp. 696–707, Apr. 2018.
[13] Z. Cheng, P. Li, J. Wang, and S. Guo, "Just-in-time code offloading for wearable computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 1, pp. 74–83, Mar. 2015.
[14] Q. Zhang, L. T. Yang, Z. Yan, Z. Chen, and P. Li, "An efficient deep learning model to predict cloud workload for industry informatics," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2018.2808910.



NAM KY GIANG received the B.E. degree from the Hanoi University of Science and Technology in 2010 and the master's degree from the Korea Advanced Institute of Science and Technology in 2013. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, The University of British Columbia. 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 LEA received the Ph.D. degree in computer science from Lancaster University, U.K., in 1989. He was with Chorus Systems and HP Labs, and then he moved to Sony's Tokyo Laboratories 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, he founded Sony's Distributed Systems Laboratory, San Jose, CA, USA. Under his leadership this laboratory generated over 100 patents and delivered a number of key technologies to Sony product groups. He 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 personalized home media server. Parts of this work were subsequently adopted by the European TV standard DVB MHP and the U.S. OpenCable. As a Vice President, responsible for research and development activities, he 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 U.S. venture capital fund. In 2006, he rejoined Lancaster University as an EPSRC Research Fellow and currently holds the title of Senior Research Fellow. In addition to his industrial work, he 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 has 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, and IFIP. He is currently a Board Member and a Co-Chair of ACM Middleware and a Vice Chair of the IEEE Ad Hoc Committee on Industry Engagement.

VICTOR C. M. LEUNG (S'75–M'89–SM'97–F'03) received the B.A.Sc. degree (Hons.) in electrical engineering from The University of British Columbia (UBC) in 1977 and the Ph.D. degree in electrical engineering from the Graduate School, UBC, in 1982. He received the APEBC Gold Medal as the Head of the Graduating Class with the Faculty of Applied Science. He received the Canadian Natural Sciences and Engineering Research Council Postgraduate Scholarship for his postgraduate studies.

From 1981 to 1987, he was a Senior Member of Technical Staff and a Satellite System Specialist with MPR Teltech Ltd., Canada. In 1988, he was a Lecturer with the Department of Electronics, The Chinese University of Hong Kong. He returned to UBC as a Faculty Member in 1989, where he is currently a Professor and the TELUS Mobility Research Chair in Advanced Telecommunications Engineering with the Department of Electrical and Computer Engineering. He has co-authored over 1100 journal/conference papers, 40 book chapters, and co-edited 14 book titles. Several of his papers have 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 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 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 received the 2017 IEEE ComSoc Fred W. Ellersick Prize, the 2017 IEEE Systems Journal Best Paper Award, and the 2018 IEEE ComSoc CSIM Best Journal Paper Award. He served on the Editorial Boards for the IEEE Journal on Selected Areas in Communications-Wireless Communications Series and Series on Green Communications and Networking, the IEEE Transactions on Wireless Communications, the IEEE Transactions on Vehicular Technology, the IEEE Transactions on Computers, the IEEE Wireless Communications Letters, and the *Journal of Communications and Networks*. He is serving on the Editorial Boards for the IEEE Transactions on Green Communications and Networking, the IEEE Transactions on Cloud Computing, the IEEE Access, *Computer Communications*, and several other journals. He has guest-edited many journal special issues, and provided leadership to the organizing committees and technical program committees of numerous conferences and workshops.