Manisha Luthra, Boris Koldehofe. *ProgCEP: A Programming Model for Complex Event Processing over Fog Infrastructure.* To appear in the Proceedings of 2nd International Workshop on Distributed Fog Services Design, December 2019, ISBN 978-1-4503-7031-8/19/12.

ProgCEP: A Programming Model for Complex Event Processing over Fog Infrastructure

Manisha Luthra, Boris Koldehofe
Technical University of Darmstadt
Darmstadt, Germany
[firstname.lastname]@kom.tu-darmstadt.de

ABSTRACT

Complex Event Processing (CEP) is a powerful paradigm that can derive meaningful insights by correlating multiple data sources, e.g., in the Internet-of-Things applications. However, these applications often require deployment across a wide variety of devices ranging from mobile devices to edge and cloud or simply put: *fog infrastructure*. This is not easily possible using existing programming models because of missing (i) support for deployment on heterogeneous devices and (ii) important interfaces for the deployment of CEP, e.g., for developing *operator placement* algorithms.

In this paper, we present ProgCEP: a programming model that facilitates the development of the operator placement algorithm and its deployment in a fog computing setting. In addition, it is portable and deployable on any kind of fog infrastructure and provides dynamic scaling of resources and deployment using the operator placement algorithm. We evaluate ProgCEP on its *applicability* and *realizability* on a publicly available fog testbed involving on-site, GENI and CloudLab resources using Docker tools. To this end, we enable (i) deployment of CEP using our dockerized implementation on the aforementioned fog infrastructure in less than 50 secs (on 25 distributed resources) and (ii) easy development of operator placement algorithms in terms of minimum lines of code.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

KEYWORDS

Complex Event Processing; Programming Model; Fog Computing

ACM Reference Format:
Manisha Luthra, Boris Koldehofe. 2019. ProgCEP: A Programming Model for Complex Event Processing over Fog Infrastructure. In *Proceedings of DFSD '19: 2nd International Workshop on Distributed Fog Services Design (DFSD '19).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3366613.3368121

1 INTRODUCTION

Complex Event Processing (CEP) is a powerful paradigm to derive meaningful insights from raw data streams generated by multiple data sources. It is widely used in different applications including network monitoring, traffic management, financial trading, etc. Another emerging class of applications Internet-of-Things (IoT) can highly benefit from CEP by distributed processing and deployment of so-called *operators* e.g., filters or joins on top of processing devices. For instance, for traffic congestion detection, sensor streams from multiple vehicles including vehicle speed and density can be correlated and distributed for efficiently processing a traffic congestion query [11]. The processing of the events could happen at different devices, e.g., road side units, vehicles, road side cameras, cloud or simply: *fog infrastructure*. However, this is not easily possible using existing CEP programming models due to limitations in (i) flexibly deploying CEP operators on distributed heterogeneous devices and (ii) important interfaces for developing operator placement algorithm, which is a key technique for deployment in CEP. Additionally, a real world deployment of CEP on distributed fog infrastructure is missing or only limitedly explored [2, 15].

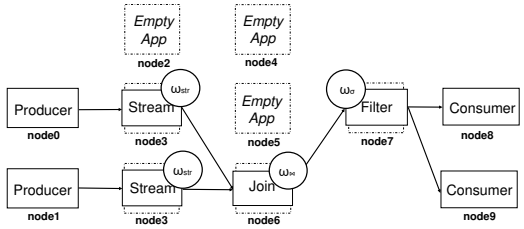


Figure 1: ProgCEP node model. The solid contour indicates pinned operators while dotted indicates unpinned operators.

In this paper, we present a programming model, ProgCEP, for CEP deployment on fog infrastructure. Using the proposed programming model, we make it easier for developers to design and deploy operator placement algorithms as well as to dynamic scale fog resources. An operator placement algorithm dictates deployment of CEP operators, e.g., filter, join, etc., on physical hosts such as fog infrastructure for distributed query processing. In Figure 1, an example on operator placement is shown, which is further explained in the system model of ProgCEP. We enable automated and distributed deployment of our programming model using Docker tools on fog infrastructure. We further demonstrate the deployment on an open fog testbed including on-site, GENI[1] and CloudLab[2] resources. To this end, we can perform (i) distributed CEP deployment on a wide range of fog infrastructure in a couple of seconds and (ii) the development of an operator placement in minimum effort in terms of lines of code.

DFSD '19, December 9–13, 2019, Davis, CA, USA

ACM ISBN 978-1-4503-7031-8/19/12…$15.00
https://doi.org/10.1145/3366613.3368121

[1] GENI: an open infrastructure for at-scale networking and distributed systems. Available at https://www.geni.net/. [Accessed 10.10.2019].
[2] CloudLab: flexible, scientific infrastructure for research on the future of cloud computing. Available at https://www.cloudlab.us/. [Accessed 10.10.2019].

Manisha Luthra, Boris Koldehofe. *ProgCEP: A Programming Model for Complex Event Processing over Fog Infrastructure.* To appear in the Proceedings of 2nd International Workshop on Distributed Fog Services Design, December 2019, ISBN 978-1-4503-7031-8/19/12.

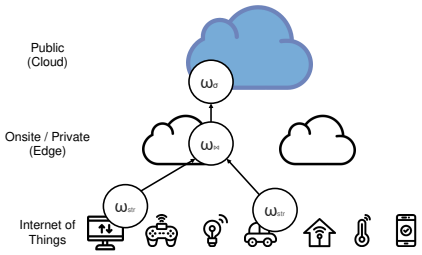


Figure 2: Mapping of CEP operators on Fog infrastructure model.

The paper is structured as follows. In Section 2, we provide ProgCEP model, in Section 3, we present the design including the programming model and distributed deployment using a workflow. In Section 4, we evaluate in terms of the application areas and its deployment on a publicly available fog testbed. In Section 5, we review the related work and finally we conclude and provide outlook in Section 6.

2 PROGCEP SYSTEM MODEL

ProgCEP comprises of event producers that generate continuous and ordered data streams. The event consumers specify a complex event that must be detected, e.g., a traffic congestion by means of a query language. A query comprises of a set of operators $\omega \in \Omega$ such as stream (ω_{str}), filter (ω_{σ}), join ($\omega_{\bowtie}$), etc., which are compute units that collectively define a complex event. ProgCEP inherits standard set of CEP operators included in AdaptiveCEP query language [18]. AdaptiveCEP is a query language that is embedded in the mainstream of ProgCEP system. Using AdaptiveCEP, ProgCEP is able to specify complex events with Quality of Service (QoS) demands that must be fulfilled by the system. Such QoS demands can be used to specify requirements, e.g., in terms of latency that must be fulfilled by the underlying fog infrastructure.

2.1 Node Model

ProgCEP employs *Empty Apps* that serves as event brokers to host CEP operators for distributed event processing. An Empty App is used to encapsulate any kind of CEP operator. The basic ProgCEP system model is illustrated in Figure 1. We refer to the hosts of producers, consumers and brokers as *nodes*. The nodes communicate in ProgCEP by forming an overlay network that can be operated on kind of fog infrastructure. Each of these system nodes are encapsulated in separate containers. In CEP systems, the query is collaboratively processed in the form of a directed acyclic graph called an operator graph (as seen in the figure above). ProgCEP differentiates between pinned entities, i.e., producers and consumers from unpinned entities, i.e., CEP operators by means of what we call *static* and *dynamic* containers, respectively. As the name suggest, the static containers are pinned to one node, while the dynamic containers are unpinned meaning these can exchange operators on different nodes at runtime. In this way, we enable a flexible operator deployment on fog infrastructure.

2.2 Placement Model

In CEP systems, the deployment of operators on infrastructure is dictated by means of an operator placement algorithm. It allows placement of the operators optimally or sub-optimally on physical hosts, while minimizing for a cost function of one or more QoS metrics such as end-to-end latency. On the one hand, ProgCEP, contributes a generic system model that allows placement of CEP operators on any kind of fog infrastructure. On the other hand, novel operator placement algorithms can be developed and integrated easily using ProgCEP placement interfaces.

2.3 Fog Infrastructure Model

We consider a hierarchical fog infrastructure as shown in Figure 2, comprising of three layers: IoT, edge and cloud layer. The IoT layer comprises of IoT devices interconnected over a wireless communication link. The edge layer offers resources at the edge, which offer a low latency link to the IoT devices in physical proximity. Lastly, a fixed network cloud layer comprising distributed resources in data centers. It is important to note that cloud and edge resources are assumed to communicate via a fixed IP infrastructure, while IoT devices and edge resources can form different wireless network topologies including device-to-device communication between IoT devices.

In the IoT scenario, IoT layer represents producers and consumers while operators can be placed on any of the three layers. The end-to-end latency for this resource model is influenced by the physical proximity of resources, but also the computational power of resources. In general, we assume higher resource availability and processing power at the cloud. In contrast, IoT devices are considered to be resource-constrained because they are battery-powered. Edge nodes are considered more powerful than mobile nodes. They are however constrained in their availability.

3 THE PROGCEP DESIGN

ProgCEP provides a programming model to realize flexible deployment of CEP applications on fog infrastructure. To do this, ProgCEP heavily benefits from akka[3] and Docker[4] tools. We model the set of producers (P), consumers (C) and *Empty Apps* as akka actors. The actor based communication model of akka allows ProgCEP to form a cluster and communicates using Gossip protocol. Hence, enabling the fog infrastructure including the Internet of Things (IoT) devices to communicate in a device-to-device fashion. The monitoring and placement related information is also propagated via gossip protocol among the Empty Apps. In the below sections, we first describe the programming model by explaining the placement interface and *Empty Apps* abstraction and then we describe the overall workflow of ProgCEP.

3.1 ProgCEP Programming Model

We aim for three main design goals: (i) easy deployment of operators on fog resources using state-of-the-art mechanisms while giving developers to extend it in order to uncover further opportunities (Section 3.1.1) (ii) allow applications to easily scale up (within same node) and out (adding more nodes) as and when required (Section 3.1.2) and (iii) provide a portable and lightweight model (with minimum footprint) that supports a wide range of heterogeneous fog resources distributed over a wide area (Section 3.1.3).

[3] akka: a distributed systems toolkit. Available at https://akka.io/. [Accessed 10.10.2019].
[4] Docker available at https://www.docker.com/. [Accessed 10.10.2019].

Manisha Luthra, Boris Koldehofe. *ProgCEP: A Programming Model for Complex Event Processing over Fog Infrastructure.* To appear in the Proceedings of 2nd International Workshop on Distributed Fog Services Design, December 2019, ISBN 978-1-4503-7031-8/19/12.

API	Description
`mechanismName`	Sets the placement mechanism name
`getPlacementMetrics`	Determines the QoS metrics that must be optimized
`initializePlacement`	Resets placement parameters. (called initially and on reconfiguration)
`findHost`	Finds physical hosts for operator placement determined based on placement metrics
`findPossibleNodesToDeploy`	Retrieves all nodes that can host operators

Table 1: ProgCEP placement interface for developing novel placement algorithms.

3.1.1 Placement Programming Model. ProgCEP placement programming model provides a means for developers to implement novel operator placement algorithm while utilizing fog infrastructure. An operator placement is defined as a suitable mapping of CEP operator graph to available infrastructure while satisfying a cost objective function. The infrastructure as studied in the existing work is commodity hardware including bare metal servers, while utilizing some form of virtualization, e.g., virtual machines. Only a few solutions [2, 15] have been proposed that consider fog computing infrastructure. They focus on proposing novel placement algorithms, while we focus on facilitating development of novel placement algorithm.

In particular, we provide a programming model for operator placement as shown in Table 1 such that CEP developers can (i) design novel placement mechanisms as well as (ii) experiment and compare against existing ones. Although, there exists a wide range of popular streaming systems, e.g., Apache Flink [3] and Storm [8], which allows to write CEP applications, however, to the best of our knowledge this is the first effort in facilitating development of novel operator placement algorithms.

```
def findHost(cluster: Cluster) {
    val candidates = findPossibleNodesToDeploy(cluster
        )
    applyRandomAlgorithm(candidates)
}

def applyRandomAlgorithm(candidates: Vector[Member]) {
    val random: Int = scala.util.Random.nextInt(
        candidates.size)
    val randomMember: Member = candidates(random)
    HostInfo(randomMember, this.getPlacementMetrics())
}
```

Listing 1: Excerpt of a Random operator placement

As prominently discussed in the literature [9], we divide the existing operator placement algorithms into two main categories: (i) centralized and (ii) decentralized. A centralized placement algorithm assumes global knowledge on the network (specific QoS metrics) to assign an operator to a physical host. In contrast, a decentralized algorithm deals with only partial knowledge on the network or cluster of hosts. It is known that finding an optimal placement is an NP-hard problem [4]. Hence, there exists many heuristics and approximate solutions to this problem. Both kind of placement heuristics require monitoring knowledge on the QoS metrics. In ProgCEP, we provide explicit extensible monitors for commonly used QoS metrics such as latency, bandwidth, CPU load, etc. Each of these metrics are measured from end-to-end, meaning the entire path from producer to consumer. The measurements are accumulated step by step and hence individual measurements can also be fetched easily. For centralized placement algorithms, the QoS monitors transfer the observed metric to a centralized broker. While for decentralized algorithms, we provide access to well-known decentralized monitoring algorithms like Vivaldi [5], which is used in prominent operator placement algorithms [13, 14]. Next, we explain how easy it is to implement a placement heuristic using our APIs defined in Table 1. Next, as mentioned before, each mechanism has a cost objective function of a QoS metric. First, each placement mechanism needs to define a name, this is a simple `String`, e.g., Random. The cost objective function is composed of one or more QoS metrics, e.g., latency, CPU load, bandwidth utilization, etc., that determines suitability of physical hosts for operator placement. This is determined by the `getPlacementMetrics` API, which connects to the respective QoS monitor. Consequently, this helps in formulating the cost function. Thereafter, the placement parameters are initialized using `initializePlacement`, which is invoked in the beginning and each reconfiguration, e.g., during periodic updates. Lastly, `findPossibleNodesToDeploy` and `findHost` determines the physical host where the operator can be deployed. In Listing 1, we provide an excerpt of random algorithm using our APIs in Scala.

The `findHost` method is recursively called to deploy operator on random candidate host returned by `applyRandomAlgorithm` method. The returned host contains the information on the placement metrics, e.g., latency for operator placement algorithm that minimizes for latency. For the random algorithm it is empty.

3.1.2 Dynamic scaling. To achieve our second design goal, we looked into two kinds of deployment modes: (i) centralized and (ii) distributed deployment. In the first design, we allow launching ProgCEP, as Docker containers on a single physical machine. Here, each system entity, namely producer, consumer and Empty Apps are encapsulated in a Docker container listening on different ports and communicating using a Docker bridge network. Although, this design provides distribution, yet it is restricted to a single physical machine and can only scale up (adding more resources to the same node, example by utilizing all the cores). Yet, users can benefit from this deployment mode for scaling up of resources.

```
VehicleSensor:
    image: mluthra/progCEP
    environment:
      - MAIN=progcep.machinenodes.Publisher
      - ARGS=--port 3300 --name VehicleSensor --ip
          VehicleSensor --host node1
    ports:
      - 3300:3300
    networks:
      - vehicularNetwork
    deploy:
      replicas: 2
      restart_policy:
        condition: on-failure
      placement:
        constraints: [node.hostname == node1]
    privileged: true
```

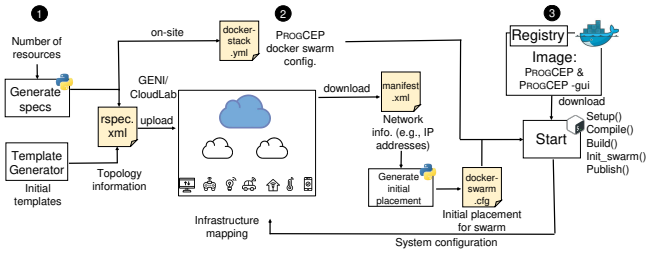
Listing 2: Sample listing in ProgCEP Docker YAML configuration

Manisha Luthra, Boris Koldehofe. *ProgCEP: A Programming Model for Complex Event Processing over Fog Infrastructure.* To appear in the Proceedings of 2nd International Workshop on Distributed Fog Services Design, December 2019, ISBN 978-1-4503-7031-8/19/12.

To provide further scalability (scale out), we extended the architecture to distributed deployment mode, where we utilized resource orchestrators like Docker Swarm and Kubernetes for initial deployment. In this design, we launch ProgCEP system entities as Docker services, which can be managed by orchestrators like Docker Swarm or Kubernetes. For this, a compose configuration that specifies the placement constraints as shown in the Listing 2 is maintained.

Here, `VehicleSensor` is a producer service with the main definition in `Publisher` class and corresponding name and port defined under configuration options `environment` and `ports`, respectively. The arguments (`ARGS`) option defines the akka actor name which is used by the akka remoting to communicate across distributed nodes. The Docker services connect in an overlay network named `vehicularNetwork`. A major difference between a bridge and an overlay network is that the former are isolated networks on a single engine, while the latter spans over multiple engines. The deployment can be made more reliable using the configuration option `replicas` and `restart_policy`. The former defines the replication factor of the service while the latter defines the counter action for failures. Lastly, the placement constraint gives the initial placement of the service on the physical host. The given example in the listing is a *static container* service, which always remain on `node1`. In addition to these, *dynamic containers*, the Empty Apps are created and initially assigned to a node. However, these are reconfigured using operator placement algorithms as defined in the above section. We utilize such configuration to scale to multiple nodes defined in a node specification (containing metadata such as host info) as stated in Section 3.2 by renaming the hostnames to node0, node1, etc. These hostnames are mapped to container service names using the above compose configuration. Consequently, the overlay network allows us to use container names to enable the communication in the network. Docker Swarm generates routing tables which map the container names to specific IP and port addresses. As placement module reconfigures the placement, Docker Swarm automatically rewrites these routing tables, which is a major advantage of using it.

3.1.3 Container-based Virtualization. Virtualization in ProgCEP plays an important role to encapsulate applications with their dependencies on any infrastructure and hence provide platform independence, but also resource isolation which is important as multiple IoT applications might run simultaneously on the fog infrastructure. A widely used approach towards virtualization is using hypervisor like Xen, VMWare, etc., which allows multiple guest operating systems on a single physical machine. These guest operating systems that are executed are called virtual machines (VMs). Although, VMs provide complete isolation from host it require large infrastructure and higher compute power. Another way of providing virtualization is using Linux containers, e.g., Docker containers. These provide a lightweight and faster boot times while still providing resource isolation from the host operating system. Another emerging kind of lightweight virtualization technique is unikernel, that provides complete isolation of resources from the host since it builds on the concept called library operating system. This is a method for constructing an operating system where the kernel and application runs in the same address space. However, this results in poor portability as the kernel needs to be compiled for drivers written



(a) ProgCEP workflow

Figure 3: ProgCEP distributed deployment on fog infrastructure illustrating the workflow.

for specific hardware. While unikernels, provide complete isolation they are bad in terms of portability. Linux containers provide ProgCEP a right level of abstraction to enable wide area portability in a lightweight way on heterogeneous fog resources. For this reason, we choose containers, specifically Docker (since it is well established) as an abstraction to build ProgCEP images.

3.2 ProgCEP Workflow

In Figure 3, we illustrate the workflow of utilizing the ProgCEP programming model to deploy CEP operators on fog infrastructure. We explain the workflow using the hierarchical fog infrastructure model as follows. First, ❶ we generate the input specification for the specified number of resources to be acquired for deployment. This number is fixed, yet, not all the resources should be used for deployment. This is to acquire knowledge on the number of resources participating in the deployment. The automatic specification generator outputs a spec file, e.g., `rspec.xml` for GENI and CloudLab resource acquisition. This specification includes the limitations of the resources, e.g., in terms of number of cores, RAM, etc. Other commercial cloud providers e.g., AWS can also be easily integrated in this process. To acquire edge and IoT resources, similar template configuration can be utilized where, e.g., host IP addresses and user information needs to be known.

❷ Next, an integration process, regenerates the global node configuration that contains aforementioned knowledge on the acquired resources. The initial resource management is performed using Docker Swarm as detailed in the programming model. Additionally, we support other resource orchestrators like Kubernetes and Apache Mesos, which can be easily integrated. ❸ This process is followed by download of ProgCEP and ProgCEP GUI images from the Docker hub. After successful compile and build, resource management takes place using ProgCEP placement programming model. It performs resource management and allocation using state-of-the-art placement techniques as explained in the above section. The execution environment assumes only Docker installation, however, it can be automatically installed using our setup configuration. Some instances of cloud provide readily available resources with Docker installed, e.g., CloudLab, where setup need not be performed. This decreases the deployment time. In addition, use of registry and the master-worker pattern in swarm orchestration further decreases the deployment time. This is because the Docker image has to be only downloaded once on the master. Furthermore, Swarm ensures replication and recovery of nodes and hence makes ProgCEP fault-tolerant.

Manisha Luthra, Boris Koldehofe. *ProgCEP: A Programming Model for Complex Event Processing over Fog Infrastructure.* To appear in the Proceedings of 2nd International Workshop on Distributed Fog Services Design, December 2019, ISBN 978-1-4503-7031-8/19/12.

Testbed	Type	Location	# of resources
Onsite	Intel(R) Core(TM) i5-6200U	Darmstadt	1
GENI	MAKI protogeni	Darmstadt	5
GENI	Instageni	Illinois	9
CloudLab	Emulab	Wisconsin	10

Table 2: Number of resources acquired for the fog infrastructure.

4 EVALUATION

We evaluated ProgCEP in two aspects, the applicability and realizability. In the first evaluation, by presenting the application areas we show that it is utilized to build CEP applications and develop novel operator placement algorithms in practice [10, 11]. In the second evaluation, we show the realization of ProgCEP on fog testbed and evaluate its performance. Lastly, we show the simplicity of our programming model by evaluating the lines of code needed to write state-of-the-art operator placement algorithms.

4.1 Application Areas

TCEP system. TCEP [11] is a transition-capable CEP system that develops a concept of transition i.e., to dynamically exchange operator placement mechanisms at runtime to deal with the changes in the environment and QoS demands. The main idea behind is that there is no *one-size-fits-all* placement heuristic which is suitable for each environmental condition. For this, two transition modes that defines the strategy for transition were proposed. For facilitating transitions between operator placement, such a system needs to have an interface for development of an operator placement mechanism and underlying infrastructure for deployment of operators. TCEP benefits from ProgCEP programming model and placement interfaces to develop the concept of transitions and to evaluate different strategies.

Transitions in Fog Infrastructure. In [10], authors explain how flexibility in utilizing fog infrastructure can be utilized for the concept of transitions. In this paper, first, the authors presented an extensive case study on multiple use cases in a smart city environment, where transitions in fog infrastructure can be helpful. Secondly, they introduced transitions in different CEP mechanisms including operator placement. Lastly, they evaluated two placement mechanisms Relaxation [13] and Mobile DCEP [17] to show there is no *one-size-fits-all* operator placement mechanism. To achieve this, the authors utilized ProgCEP in order to develop different operator placement mechanisms using our simple interfaces.

4.2 Realizability

To show the benefit of ProgCEP over real fog infrastructure, we deploy it over a cluster of cloud and edge resources. For the cloud part, we utilize a mix of GENI[1] and CloudLab[2] resources and for the edge part, we utilize some on-site resources available in our lab and GENI (MAKI protogeni), in consistent to the hierarchy of our infrastructure model in Figure 2. This gives us some resources that are near to the user at edge, while some far away resources, which comprise our fog testbed. We limit the resources at the edge and on-site in terms of available memory to 2 - 4 GiB, while at cloud in GENI and CloudLab it ranges between 8GiB - 32GiB. This introduces additional heterogeneity in the available resources in terms of memory in addition to location and network latency. This is easily done using the aforementioned `rspec.xml` in GENI and CloudLab. GENI and CloudLab are open infrastructure for research in networking and distributed systems at large scale. We leverage from these two test-beds to acquire resources for our experimental fog infrastructure as explained in Table 2. Using the above resources, we have shown the applicability of ProgCEP for two big areas in existing work: TCEP [11] and Transitions in Fog infrastructure [10].

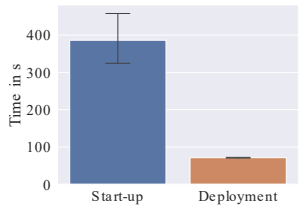


Figure 4: Start-up and deployment time on fog infrastructure.

Placement Algorithm	Kind of Placement	QoS metric (Cost)	Lines of Code (LOC)
Producer-Consumer	Centralized	-	20 LOC
Random	Centralized	-	28 LOC
Global Optimal	Centralized	Bandwidth ∗Latency	89 LOC
Relaxation [13]	Decentralized	Bandwidth ∗Latency^2	116 LOC
MOP-Algorithm [14]	Decentralized	Bandwidth ∗Latency	98 LOC
Mobile DCEP [17]	Decentralized	Message-Overhead	173 LOC

Table 3: Lines of code required to include a new placement algorithm using our interface.

Furthermore, we evaluated on the setup, start-up and deployment time of ProgCEP on the fog infrastructure described in Table 2. The setup time for ProgCEP on each host (using Docker registry) is around 30 seconds, thanks to our improved workflow (Section 3.2), in comparison to 180 seconds if Docker container is built on each host. In Figure 4, we present a bar plot with 95% confidence interval from 10 experiments for start-up and deployment times using our dockerized implementation. The start-up time taken by the given nodes joins the Docker swarm ranges between 300-400s for 25 nodes. The deployment time gives the amount of time required to publish the services i.e., the event producers, consumers and empty apps on the acquired cluster of resources which is done in around 70s.

To show the simplicity of our programming model we implemented simple operator placement algorithms in Scala, such as, Random (randomly chooses nodes for placement), Producer-Consumer (places on producers and consumers) and state-of-the-art Global Optimal (optimal placement), Relaxation [13] and Mobile DCEP [17]. In Table 3, we show the respective placement algorithms, the kind, the QoS metric they target and the lines of code in which we implemented these algorithms using our programming model. It is clear that our programming model minimizes the effort of operator placement implementation, with as low as 20 LOC for simple algorithms to less than 200 LOC for sophisticated ones.

Manisha Luthra, Boris Koldehofe. *ProgCEP: A Programming Model for Complex Event Processing over Fog Infrastructure.* To appear in the Proceedings of 2nd International Workshop on Distributed Fog Services Design, December 2019, ISBN 978-1-4503-7031-8/19/12.

5 RELATED WORK

We review the existing work in two directions: (i) distributed complex event processing programming models and (ii) fog programming models.

Complex event processing programming models. Current CEP systems are restricted in terms of developing novel operator placement algorithms in different ways. For instance, Apache Storm [8] in its classical form only provide a basic round robin operator placement. Although custom implementations for Apache Storm exists [4], none of them provide a uniform programming model for developing and deploying operator placement. DCEP-Sim [16], is a distributed CEP simulator that provides simulation of CEP applications on top of ns3. In contrast, we enable better evaluation of performance and accelerate development of such fog computing applications using our deployment architecture. Apache Flink [3] is another CEP engine that provides deployment of operators in a distributed fashion. However, it also relies on an integration with resource orchestrators like Kubernetes or Mesos for operator deployment. Resource orchestrators like Kubernetes [1] provides interface to develop and deploy placement algorithms, however, are still restricted in terms of deployment in a homogeneous infrastructure. While in this work, we provide (i) a programming model to develop CEP applications, (ii) an interface for developing operator placement and (iii) operator deployment on heterogeneous infrastructure such as *fog*.

Fog programming models. Fog infrastructure is getting big attention from academia as well as from industry due to its high performance and availability for a wide range of applications such as IoT. However, development using fog infrastructure is difficult due to limited availability of open source fog programming models. There are a few existing simulation [6] and emulation tools [7, 12] for development on applications for fog computing, however, a real world deployment of such applications using open source tools is limited. For instance, recently Mayer et al., proposed EmuFog [12], which is an extensible and scalable emulation tool for development on fog computing infrastructures. The framework utilizes Maxinet that is a multi-node extension of network emulator mininet. Although, the authors have made easy to emulate applications on top of this open source tool using Docker, yet, emulation is still a virtual representation of the actual deployment, which still gives limited insights of real world deployment that is explored in this work. Similarly fog simulation tools such as iFogSim [6], simulate fog computing infrastructure to measure the performance, yet again, only approximately close to real world deployment.

6 CONCLUSION

In this work we presented ProgCEP that enables easy deployment and development of complex event processing applications on fog infrastructure. We proposed a programming model for developing operator placement algorithms – a key component – for functionality of a CEP system. Using our programming model and dockerized implementation, we have shown the simplicity of deployment on fog infrastructure and its applicability on two existing systems [10, 11]. We have evaluated the deployment effort of our programming model on a real world deployment on fog infrastructure involving two large testbeds GENI and CloudLab. In future, we plan to evaluate operator placement algorithms using our programming model on this testbed to study the performance and enable wide range development and deployment of these mechanisms on fog infrastructure.

Acknowledgement. This work has been co-funded by the German Research Foundation (DFG) as part of the project C2 within the Collaborative Research Center (CRC) 1053 – MAKI.

REFERENCES

[1] 2019. Kubernetes. https://kubernetes.io/. Accessed 10.10.2019.
[2] Hamid Reza Arkian, Abolfazl Diyanat, and Atefe Pourkhalili. 2017. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. *Journal of Network and Computer Applications* 82 (2017), 152 – 165. https://doi.org/10.1016/j.jnca.2017.01.012
[3] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin* 38 (2015), 28–38.
[4] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. 2016. Optimal operator placement for distributed stream processing applications. In *Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems, DEBS 2016*. 69–80.
[5] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. 2004. Vivaldi: A Decentralized Network Coordinate System. *SIGCOMM Computer Communication Review* 34, 4 (Aug. 2004), 15–26.
[6] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. 2016. iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments. *Softwares for Practical Experience* 47 (2016), 1275–1296.
[7] Kirak Hong and David Lillethun. 2013. Mobile fog: a programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM Workshop on Mobile Cloud Computing*. 15–20.
[8] Sanjeev Kulkarni, Nikunj Bhagat, Masong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. 2015. Twitter Heron. *Proceedings of the ACM SIGMOD International Conference on Management of Data - SIGMOD '15* (2015), 239–250. https://doi.org/10.1145/2723372.2742788
[9] Geetika T. Lakshmanan, Ying Li, and Rob Strom. 2008. Placement strategies for internet-scale data stream systems. *IEEE Internet Computing* 12, 6 (2008), 50–60.
[10] Manisha Luthra, Boris Koldehofe, and Ralf Steinmetz. 2019. Transitions for Increased Flexibility in Fog Computing: A Case Study on Complex Event Processing. *In Informatik Spektrum, Special Issue on Fog Computing Reality* (August 2019).
[11] Manisha Luthra, Boris Koldehofe, Pascal Weisenburger, Guido Salvaneschi, and Raheel Arif. 2018. TCEP: Adapting to Dynamic User Environments by Enabling Transitions Between Operator Placement Mechanisms. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems (DEBS '18)*. ACM, 136–147. https://doi.org/10.1145/3210284.3210292
[12] Ruben Mayer, Leon Graser, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. 2017. EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures. *2017 IEEE Fog World Congress (FWC)* (2017), 1–6.
[13] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. 2006. Network-Aware Operator Placement for Stream-Processing Systems. In *22nd International Conference on Data Engineering (ICDE'06)*. 49–49.
[14] S. Rizou, F. Durr, and K. Rothermel. 2010. Solving the Multi-Operator Placement Problem in Large-Scale Operator Networks. In *2010 Proceedings of 19th International Conference on Computer Communications and Networks*. 1–6.
[15] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov. 2016. SpanEdge: Towards Unifying Stream Processing over Central and Near-the-Edge Data Centers. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*. 168–178. https://doi.org/10.1109/SEC.2016.17
[16] Fabrice Starks, Stein Kristiansen, and Thomas Plagemann. 2018. DCEP-Sim: An Open Simulation Framework for Distributed CEP: Introduction for Users and Prospective Developers. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems (DEBS '18)*. ACM, New York, NY, USA, 183–186. https://doi.org/10.1145/3210284.3219501
[17] F. Starks and T. P. Plagemann. 2015. Operator placement for efficient distributed complex event processing in MANETs. In *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 83–90.
[18] Pascal Weisenburger, Manisha Luthra, Boris Koldehofe, and Guido Salvaneschi. 2017. Quality-aware Runtime Adaptation in Complex Event Processing. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '17)*. 140–151.