An Extension to iFogSim to Enable the Design of Data Placement Strategies

Mohammed Islam Naas*†, Jalil Boukhobza†, Philippe Raipin Parvedy*, Laurent Lemarchand†
* Orange, Rennes, France
†Univ. Bretagne Occidentale UMR 6285, Lab-STICC F-29200, Brest, France
*{mohammedislam.naas, philippe.raipin}@orange.com
†{laurent.lemarchand, boukhobza}@univ-brest.fr

***Abstract*—Fog computing consists in extending Cloud services down to the network edge by using resources such as base stations, routers and switches. It presents a dense, heterogeneous and geo-distributed infrastructure which pushes to investigate how data are placed within this infrastructure in order to minimize service latency, network utilization and energy consumption. *iFogSim* is a Fog and IoT environments simulator dedicated to manage IoT services in a Fog infrastructure. In this paper, we present an extension to *iFogSim* to be able to model and simulate scenarios with strategies aiming to optimize data placement in Fog and IoT contexts. Data placement problem is NP-Hard due to the large number of Fog nodes and the high amount of data to be placed. Thus, we added a support to divide and conquer strategies to subdivide the issued infrastructure into several parts hence reducing the data placement computing time. Moreover, the extension involves a generic smart city scenario with different workloads making it possible for the users to investigate the behavior of their strategies using various workloads. In order to optimize the execution time of the simulations, we parallelized the Floyd-Warshall algorithm. This algorithm is used in *iFogSim* to compute all shortest paths between nodes in order to simulate data transmission. We have evaluated this extension using the proposed smart city scenario with various infrastructure configurations. The experiments show that our extension has a small overhead in terms of simulation time and memory utilization.**

***Index Terms*—Internet of Things, Fog computing, Data Placement, Data Storage, iFogSim, Graph partition, Generalized Assignment Problem, Floyd Warshall Algorithm.**

I. Introduction

The Internet of Things (IoT) considers a global and dynamic network of smart and virtual objects [5]. As predicted by CISCO, the number of connected objects will reach 50 billion in 2020 [11]. As a matter of fact, these objects will generate a huge amount of data; from 145 Zetta Bytes (ZB) in 2015 up to 600 ZB by 2020 [19]. Due to the exponential increase in the number of objects and the high data traffic that they generate, relying exclusively on the Cloud for processing and storage purposes would cause high access latencies for application instances spread over the whole network. Furthermore, applications for healthcare, security and Industry 4.0 are too sensitive to latency to be deployed on the Cloud [23].

Fog is a computing paradigm that consists in extending traditional Cloud services down to the network edge. It uses the network, compute and storage resources of components such as routers and switches located between IoT objects and Cloud datacenters [9]. Fog computing allows data processing, filtering and aggregation, either at the core or at the edge of the network offering latency reduction, network bandwidth preservation and Quality of Service (QoS) enhancement [6].

Fog presents a dense and geo-distributed infrastructure. Fog components, also called Fog nodes, are heterogeneous in terms of processing capabilities, storage capacities and access latencies. In fact, resource-limited nodes (e.g. set-top boxes) are located at the edge of the network and provide a low access latency for the objects, whereas resource-rich nodes (e.g. point of presence) are located in the core of the network and provide a higher access latency [17]. Since data locality is crucial for processing in such infrastructure (dense, geo-distributed and heterogeneous), it is important to investigate how data should be placed in order to reduce service access latencies [17].

Fog infrastructures may involve thousands of Fog nodes and IoT service instances and serve millions of smart objects. Solutions for such complex systems should be explored and evaluated based on simulation methods hence reducing the evaluation cost and time before deployment in real systems.

iFogSim [13] is the most popular Fog and IoT environments simulator based on *CloudSim* [7]. It can model and simulate infrastructures with millions of heterogeneous Fog nodes, IoT services, sensors and actuators. *iFogSim* requires the user to define the Fog infrastructure, mapping and scheduling policies related to IoT services, and makes it possible to evaluate different metrics related to latency, network, energy consumption and operational cost [13]. It was used in several studies [14, 18, 20] to model and evaluate policies for service allocation in order to minimize energy consumption, data traffic and to meet application QoS requirements.

Native data storage management is very simple in *iFogSim*. The generated data, called *Tuples*, are transmitted to each IoT service requesting them. This simple policy may work in case where the tuple is consumed only by one service/Fog node. However, in case where the same tuple is used by several IoT services, it is transmitted as many times as the number of requesting services. This adds a considerable overhead to the service latency, network utilization and energy consumption.

In this paper, we propose an extension to *iFogSim* that makes it possible to evaluate data (tuple) placement policies in a Fog infrastructure in order to reduce IoT service data access latency. This extension involves the followings contributions:

- The support for data storage and mapping in *iFogSim*: we

978-1-5386-6488-9/18/$31.00 ©2018 IEEE

have developed a set of classes and functions that makes it possible to keep track of data placement in the Fog.
- The problem formulation of data placement as a Generalized Assignment Problem (GAP): one of the ways to model data placement strategies is to use the GAP formulation as in [17]. An interface has been designed to allow for such a modeling.
- An interface for Mixed Integer Linear Programming (MILP) methods has been designed to solve a GAP data placement problem with the *CPLEX MILP* tool [1].
- As the data placement problem is NP-Hard [17], we developed a partitioning facility to subdivide the simulated infrastructure in subparts on which the data placement problem is solved independently and possibly in parallel, hence decreasing the solving time and complexity.
- Since a data placement strategy efficiency may depend on the characteristics of data flows and data sharing between IoT services, we made it possible to define different workloads for the simulation scenarios.
- In order to accelerate the simulation time in *iFogSim*, we implemented a parallel version of Floyd Warshall algorithm [22]. This algorithm is used to compute the shortest network path and the related latency between two nodes for data placement strategies.

We have evaluated our extension using 3 infrastructure configurations on a generic scenario of a smart city; small, medium and large infrastructure encompassing 1000, 2000 and 3000 Fog nodes, respectively. For each one, we studied the overhead added by our extension to *iFogSim* in terms of simulation time and memory utilization. The experiments have shown that our extension has a small overheads.

The rest of this paper is structured as follows. Section II presents some related work and gives some background knowledge about *iFogSim*. Section III details our contribution toward data placement integration in *iFogSim*, and Section IV describes the evaluation part. Section V concludes this work and discusses perspectives for future work.

II. Background and Related Work

In this section, we first discuss some related work and then we describe the simulator *iFogSim*.

A. Related Work

In this subsection, we summarize state-of-art work about data placement management in the context of Fog computing and IoT. Table I shows a classification of state-of-the-art work on Fog infrastructure simulators according to the following three criteria; (i) data placement management, (ii) Fog environment consideration and (iii) IoT objects modeling.

GridSim is a toolkit simulator [21] for peer-to-peer and Grid computing context. It makes it possible to model and simulate grid resources and network, and evaluate solutions for task allocation and scheduling, and finally manage data placement in the simulated infrastructure. However, there is no support dedicated to Fog and IoT environments. Thus, users have to define Fog nodes and smart objects in order to implement their data placement strategies.

TABLE I: A simple state-of-the-art work classification

Work	Data placement	Fog environment	IoT objects
[21]	+	-	-
[10]	-	+	-
[3]	-	-	+
[15]	-	+	-
[13]	-	+	+

Etemad et al. proposed a Fog environment simulator based on Discrete Event System Specification (DEVS) [10]. However, it neither simulates IoT objects nor offers data placement management in the simulated infrastructure.

In [3], Baccelli et al. introduced *FIT-IoT-LAB*, a large-scale federated experimental platform to design and benchmark IoT protocols, applications and services. It includes 2700 physical wireless sensors whose 117 mobile robot nodes are deployed across six sites in France. However, this platform does not involve neither Fog nodes nor data placement strategies.

Ruben et al. proposed *EmuFog* in [15], a Fog environment emulator. It can emulate Fog infrastructures that encompass switches and routers. However, it offers neither data placement management nor IoT objects emulation. Moreover, due to the large density of Fog infrastructures, emulation-based solutions can be very expensive in terms of resources and time.

In [13], Gupta et al. proposed *iFogSim*, a simulator for large-scale Fog and IoT environments. It can model infrastructures with millions of IoT objects and thousands of Fog nodes (and datacenters). It makes it possible for users to define and implement their own solutions to manage resources to allocate and schedule IoT services across the whole infrastructure and measure the impact in terms of access latency, network congestion, energy consumption and cost. However, data placement management is not offered yet with this simulator.

In this paper, we propose an extension of the simulator *iFogSim* that allows to design data placement management strategies in the context of Fog computing and IoT.

B. iFogSim simulator

iFogSim is a discrete event simulator for Fog and IoT environments. It is dedicated to simulate IoT application scenarios based on the *Sense-Process-Actuate* model. In this model, sensors first collect data and publish them either periodically or in an event-based manner. After that, IoT services running on Fog nodes retrieve and process those data. Finally, according to the obtained results, actions are sent (back) to actuators.

As shown in Figure 1, *iFogSim* is composed of a set of physical entities that may be of different types: *FogDevice* (i.e. a Fog node such as a router), *Sensor*, and *Actuator*.

iFogSim is also composed of a set of logical entities to model the application use-case. For example, as shown in Figure 1, *AppModule* models IoT services, *AppEdge* defines the data dependency between a pair of IoT services, and *Tuple* models the fundamental unit of communication between entities that are the data (and meta data).

In *iFogSim*, Fog nodes are considered as processing and storage resources on which IoT services (*AppModule*) can be placed and then scheduled for execution. IoT services can be placed statically or dynamically through migration mechanisms to be defined by the user.

IoT service placement policies determine how IoT services are allocated in the simulated infrastructure driven by some objectives related to end-to-end latency minimization, network utilization, energy consumption or operational cost reduction. The *AppModulePlacement* in Figure 1 is the abstraction of an IoT service placement policy, and the *AppModuleMapping* holds, for each IoT service, its hosting Fog node.

IoT service scheduling determines how IoT services are scheduled within a *FogDevice*. The default resource scheduler allocates device resources among all active IoT services in a given *FogDevice*. *AppModuleScheduler* is the abstraction of scheduling policies implemented within a Fog node.

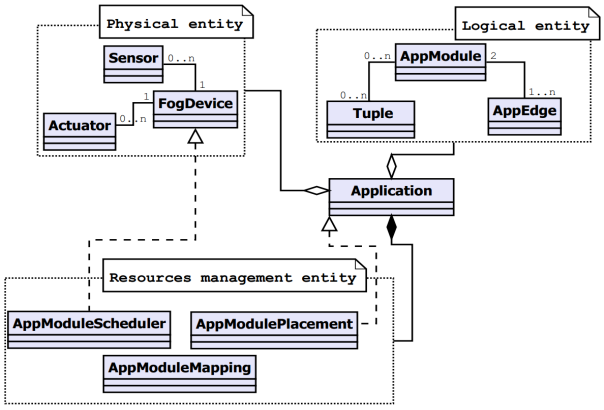


Fig. 1: UML Class diagram of iFogSim main components.

III. Contribution

In this section, we start by giving an overview of our contribution toward enabling data placement in *iFogSim* then we detail the main functionalities added, and finally, we describe the *Floyd Warshall* algorithm parallelization.

A. Overview

In this work, we provide an extension to *iFogSim* that allows users to manage data placement in the simulated scenarios in order to minimize the service latency, the network congestion, or energy consumption. As shown in Figure 2, we added three functionalities to *iFogSim* described below.

- **Data Placement**: it is the main component in our extension, it offers to users a set of functionalities to compute the data placement using linear programming methods. We also provide a set of data placement strategies which can be used for comparison.
- **Infrastructure Partitioning**: in order to reduce the data placement problem complexity, we added the *Infrastructure Partitioning* functionality. It subdivides the simulated infrastructure into several parts and runs a data placement subproblem on each one of them [16].

TABLE II: Data sizes.

Data Item	Size (in MB)
d_1	5
d_2	6

TABLE III: Available storage.

Fog Node	Capacity (in MB)
sn_1	9
sn_2	8

TABLE IV: Placement cost.

Placement Pair	Cost
d_1,sn_1	4
d_1,sn_2	2
d_2,sn_1	3
d_2,sn_2	1

- **Workload Repartition**: our extension includes a generic scenario of smart city that encompasses a set of datacenters, Fog nodes, IoT objects and IoT services. As the used workload may have an impact on the quality of a data placement policy, we added the *Workload Repartition* functionality. It allows to generate different configurations of data flows within the smart city scenario.

B. Data Placement Component

This component offers a set of tools to manage data placement in *iFogSim* (see Figure 2).

1) Problem Formulation Engine: this engine gives users the possibility to formulate the data placement as a GAP-like problem using linear programming methods. The GAP-like formulation consists in allocating for each pair of *(data item, storage Fog node)* a placement cost which can be expressed in terms of latency, energy consumption, or some other metrics. The data placement problem formulation may involve a set of constraints, for instance one may limit the data amount that can be stored in a given Fog node. The linear programming formulation of the data placement problem is given as:

$$\begin{cases} Min & Overall\_cost = \sum\sum cost_{i,j}.x_{i,j} \\ S.T & \\ & constraints \\ & \vdots \\ & x_{i,j} \in \{0,1\} \end{cases}$$

With:

- $cost_{i,j}$ is the cost of storing data item i in Fog node j
- $x_{i,j} = 1$ if data item i is stored in Fog node j, 0 otherwise
- $constraints$ are related to the placement. For instance, one may verify that all data items should be placed.

This problem consists in finding the set of $x_{i,j}$ that minimizes the $Overall\_cost$, respecting all the formulated constraints. This engine describes this problem formulation in a file with linear programming compliant format (LP file) which is to be used as an input for linear problem solvers. A simple example for this formulation is given next.

Let us assume that we are looking to minimize the global data placement cost of two data items d_1 and d_2 across two storage Fog nodes sn_1 and sn_2. Each data item has a given size (in MB) and each storage node has a storage capacity (in MB). Note that, for each pair (data item, storage node) there

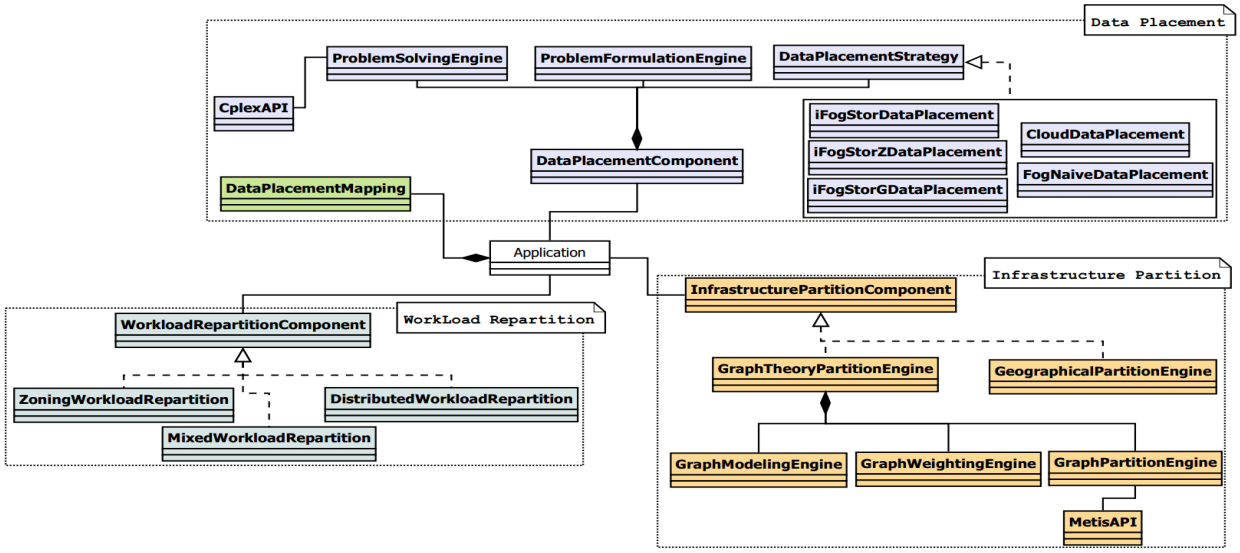


Fig. 2: UML Class diagram of the main added functionalities in iFogSim

is a placement cost (e.g. access latency). Let Table II, Table III and Table IV contain data sizes, storage capacities and all pairs (data item, storage node) placement cost, respectively. The linear formulation of this problem is described as follows:

$$\begin{cases} Min & Z = 4 * x_{1,1} + 2 * x_{1,2} + 3 * x_{2,1} + 1 * x_{2,2} \\ S.T & \\ 1) & 5 * x_{1,1} + 6 * x_{2,1} \leq 9 \\ 2) & 5 * x_{1,2} + 6 * x_{2,2} \leq 8 \\ 3) & x_{1,1} + x_{1,2} = 1 \\ 4) & x_{2,1} + x_{2,2} = 1 \\ 5) & x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \in \{0, 1\} \end{cases}$$

This problem consists in finding all $x_{i,j}$ that minimize Z (i.e. the overall placement cost) respecting all constraints. Note that, $x_{i,j} = 1$ means that the data d_i is placed in the storage node sn_j, thus the related placement cost is added to Z. Otherwise, $x_{i,j} = 0$. The first two constraints verify that the data amount affected to each storage node should be lower to its available storage capacity. Equalities 3) and 4) ensure that each data is placed once and only once. To solve this problem, we use the *Problem Solving Engine* which is detailed next.

2) Problem Solving Engine: once the data placement problem is formulated, one can use the *Problem Solving Engine* to solve it. This engine offers an interface to use the popular *CPLEX MILP* solver. It can be configured to call *CPLEX* with specific parameters such as the maximum solution search time. Users may also plug their own methods to solve the problem such as algorithms or other linear problem solvers.

3) Data Placement Strategy: this interface offers the possibility to use a set of existing strategies to place data. Users can reuse these strategies directly to compare with their solutions, or upgrade them to design some new ones:

- **Cloud Data Placement**: this strategy consists in placing all data items in the Cloud datacenters. Then, IoT services may request part of them for their own use. This case illustrates the traditional Cloud data placement method.
- **Naive Fog Data Placement**: in this strategy, data items are stored either in their sources or in the closest node (in term of latency) if there is not enough free space.
- **iFogStor Data Placement**: this strategy was proposed in [17], it consists in modeling data placement as a GAP-like problem and solves it using *CPLEX* [4]. We implemented it using the *Problem Formulation Engine* and the *Problem Solving Engine*. This strategy finds the optimal solution to place data minimizing the overall service latency. Due to the complexity of the data placement problem, this solution may lead to a combinatorial explosion.
- **iFogStorZ Data Placement**: this is a heuristic for *iFogStor* based on a spatial locality assumption proposed in [17] in order to reduce the data placement computing time. It consists in subdividing the infrastructure geographically into several zones. For each zone, a data placement sub-problem is modeled and solved separately. We implemented this strategy using the *Geographical Partition Engine* which will be presented thereafter.
- **iFogStorG Data Placement**: this is another heuristic for *iFogStor* proposed in [16]. It follows the same idea of subdividing the infrastructure. Here, the subdivision process is based on graph modeling and partition methods hence increasing the placement efficiency. We implemented this strategy using the *Graph Partition Engine*.

C. Infrastructure Partition Component

As mentioned earlier, managing IoT data placement in Fog infrastructures is a NP-Hard problem. On the other hand, as discussed in [16], data placement policies should be feasible at runtime to adapt to the dynamic nature of Fog infrastructures. To this purpose, in our previous work [16, 17], we proposed a solution to reduce the data placement computing time. Our idea is based on a divide and conquer strategy and

consists in subdividing the Fog infrastructure into several disjoint parts.Then, for each part, a new data placement with a lower complexity is modeled and solved separately and in parallel with the other ones. To implement this solution, we have developed two reusable techniques described below; (i) geographical subdivision and (ii) graph theory based subdivision (see Figure 2). Additionally to the available partitioning methods, users can develop their own ones.

1) Geographical Partition Engine: motivated by the assumption that for many IoT applications data are in general used in the same geographical area where they have been produced [16], this engine subdivides the Fog infrastructure geographically into several zones. Each zone groups all Fog nodes that are located within the same geographical area (e.g. cities, towns, quarters, etc.). Indeed, even though this technique is very simple to implement, it may suffer from several drawbacks such as producing unbalanced sub-parts in terms of complexity, reduced scalability, and placement quality degradation for large distributed IoT application workloads.

2) Graph Partition Engine: to avoid the drawbacks of the previous technique, we proposed subdividing the infrastructure using graph methods based on three components:

- **Graph Modeling Engine**: it models the physical infrastructure as an undirected graph in which vertices represent Fog nodes and edges model physical links.
- **Graph Weighting Engine**: once the graph is generated, this engine allocates weights for both vertices and edges. Vertex weights are allocated by counting the number of data items produced in each Fog node. Edges weights are allocated by counting the number of data flows passing through these links.
- **Graph Partition Engine**: once the graph has been weighted, this engine applies a k-way partitioning method to divide it into k sub-graphs using Metis [2]. The partition process respects two criteria: (i) balancing the sum of vertex weights among sub-graphs making balanced sub-parts, and (ii) minimizing the sum of cut edges weights which reduces the number of cut data flows and thus enhances the data placement quality. Indeed, cutting data flows induces a loss of information about data consumers location causing a loss in the data placement optimality.

D. Workload Repartition Component

As discussed in Section III. A, our extension includes a generic smart city scenario that users may use to deploy and test their data placement strategies. As shown in Figure 3, in this scenario, a set of sensors collect environmental data and send them to a set of IoT services for processing purpose. These IoT services are deployed within an infrastructure involving a set of datacenters and a set of Fog nodes; gateways (GW), Local PoPs (LPOP) and Regional PoPs (RPOP).

As discussed in [16], the nature of the used workload repartition (in terms of Data Flow Distribution) may have a strong impact on the efficiency of data placement policies. Thus, we added the *Workload Repartition Component* to

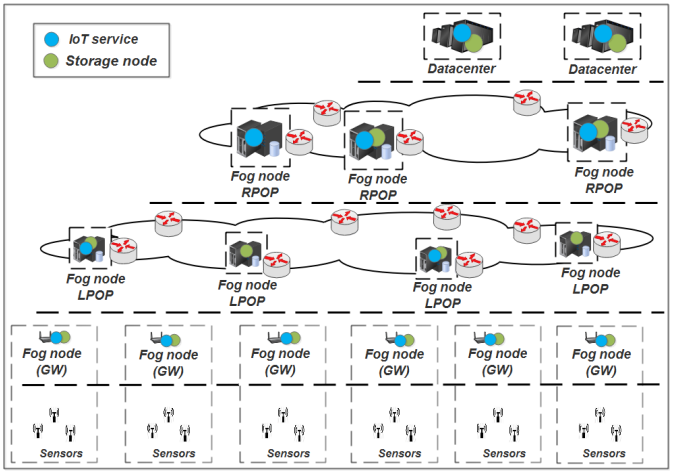


Fig. 3: Use case infrastructure.

iFogSim. It gives to users the ability to generate different workload configurations for the proposed scenario specifying their repartition rate. As shown in Figure 2, this engine also offers an interface to reuse three existing workload configurations; (i) *Zoned*, *Distributed* and *Mixed* which are described as follows.

- **Zoned workload**: here, data are used by IoT services located in the same geographical zone as the producers
- **Distributed workload**: here, data are used by randomly selecting IoT services from the whole infrastructure
- **Mixed workload**: in this case, 50% of the data are *zoned* and the other 50% are *distributed*.

E. Data placement simulation

In this section, we show how data are managed in *iFogSim* with and without our extension (see Figure 4).

A sensor first generates a tuple and sends it to the associated Fog node (ISP gateway). Once the tuple reaches this node, it is handled as follows:

- **iFogSim without data placement**: in the current version of *iFogSim* data are sent from sources to each consumer Fog node. As shown in the sequence diagram, the receiver Fog node (FogDevice1) processes the incoming tuple. Upon completion, if the precessing results in some new data which are usable for other Fog nodes (FogDevice2 and FogDevice3), the resulting data are sent to each one of them for processing purpose.
- **iFogSim with data placement**: there are two options for the incoming tuple; (i) the tuple is to be processed by the receiver Fog node or (ii) the tuple is to be processed by other Fog nodes. For the first option, the tuple is submitted for execution. Upon completion, if the processing results in some new data, the receiver node requests the *Data Placement Mapping* to retrieve information about the storage location for the resulting data. Then, data are sent as a new tuple to the specified Fog node (FogDevice2) for storage purpose. For the second option, the receiver Fog node requests the *Data Placement Mapping* to retrieve the related storage location for the received

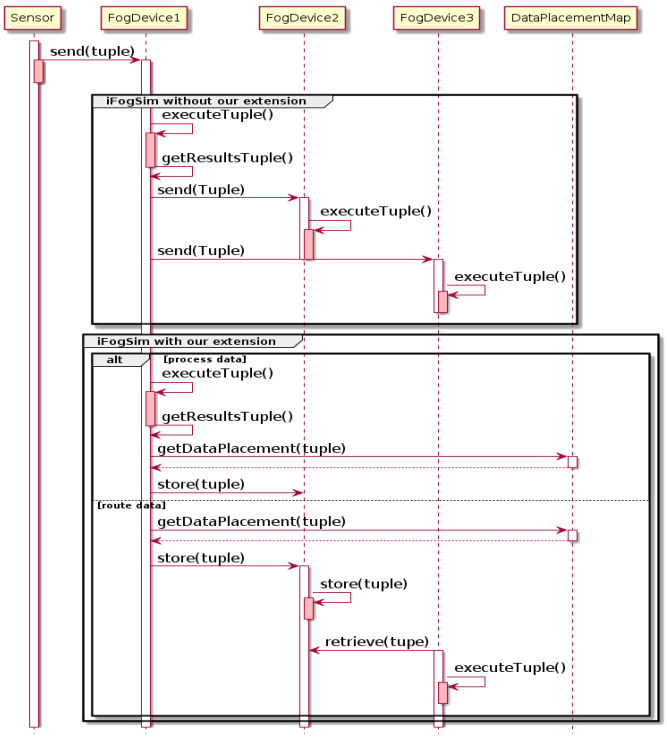


Fig. 4: Sequence diagram of data placement setup in iFogSim.

tuple. Then, the tuple is sent to the specified Fog node (FogDevice2) for storage purpose. Once the tuple reaches this node, consumer nodes (FogDevice3) retrieve the tuple then process it.

F. Floyd Warshall algorithm implementation

Floyd Warshall is an algorithm used to find all-pair shortest paths in a weighted graph [22]. *iFogSim* uses this algorithm to find the shortest paths between all nodes and the related latency before starting the simulation. This is done in order to simulate the required data transfer time between Fog nodes. When the number of nodes in the Fog infrastructure is too large (thousands or even millions), the computation time of this algorithm may cause a combinatorial explosion problem. This is due to its complexity which is $O(n^3)$; with n being the total number of nodes [22]. Hence, to accelerate the execution time of this algorithm, we implemented parallel version using the *OpenMP* library [8] written in the *C* language. Using our implementation, users can set the used number of threads to execute this algorithm in order to get a specific acceleration.

IV. Evaluation

This section presents the evaluation part of our contribution. We first describe the used methodology. Then, we give the experimental setup. Finally, results will be discussed.

A. Methodology

In this work, we are interested in enabling data placement management in *iFogSim* rather than evaluating the placement strategies themselves, which has already been studied in [16, 17]. In order to evaluate our contribution, we focused on the overhead added to *iFogSim* in terms of simulation time and memory utilization by integrating data placement management. Moreover, we investigated the speedup obtained by using our parallel implementation of the *Floyd-Warshall* algorithm. We evaluated these metrics by simulating our generic smart city scenario using different infrastructure configurations (see the next section).

- *Simulation time*: we measured the required time to accomplish a system simulation with and without using data placement management. For the first option, data were transmitted directly to the related IoT services after their generation without optimizing their placement. For the second option, data were stored in Fog nodes following the Fog naive data placement strategy.
- *Memory footprint*: we did the same as for the simulation time, we measured the size of the allocated heap in the main memory with and without using data placement management. To do this, we used *pidstat* the system activity information collector available in the *Sysstat* tool [12] to measure the heap allocation during the simulation.
- *Floyd Warshall speedup*: we measured the required time to compute all shortest paths existing between Fog nodes using the sequential (*iFogSim* default implementation) and our parallel versions of this algorithm. For the latter, we varied the number of threads from 1 to 32, and in each case, we computed the obtained speedup.

TABLE V: Network latencies.

Network link	Latency (ms)
Sensor - GW	10
GW - LPOP	50
LPOP - RPOP	5
RPOP - DC	100
RPOP - RPOP	5
DC - DC	100

TABLE VI: Storage capacities.

Fog node	Storage capacity
GW	100 GB
LPOP	10 TB
RPOP	100 TB
DC	10 PB

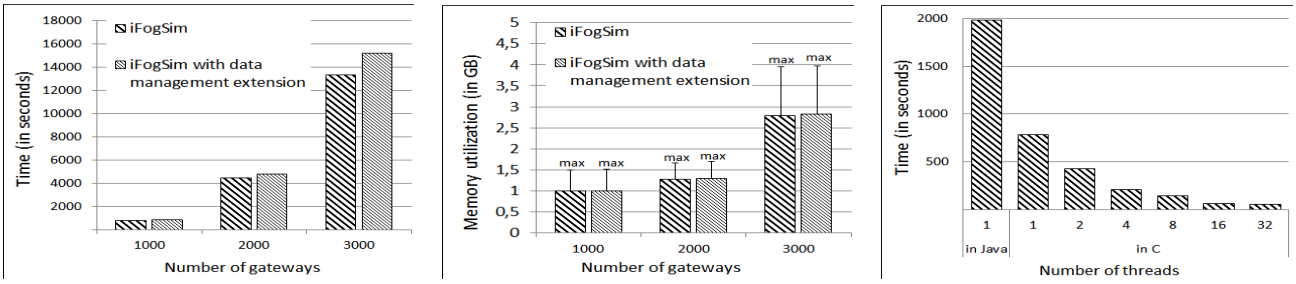
B. Experimental Setup

The simulations have been achieved using a computer having 48 CPU cores and 348 GB of RAM.

The simulated system infrastructure involves 5 datacenters, 10 RPOPs, 100 RPOPs. In order to evaluate different case-study complexities, we varied the number of gateways from 1000 to 3000 by steps of 1000, as the network edge is more dynamic than the network core. We have configured 15 sensors and 5 actuators per gateway. We supposed that data were exchanged between the system entities with packets [17]. In our experiments, we set the size of a data packet generated by a sensor to 96 bytes and the size of a data packet generated by an IoT service to 960 bytes [17]. Table V shows the considered network latencies between nodes, and Table VI illustrates the available storage capacity for each type of *Fog node* [16].

C. Results and Discussion

In this section we discuss the simulation results about: (i) the overhead added to *iFogSim* by our extension in terms of



(a) Simulation time. (b) Mean and max of memory utilization. (c) Floyd-Warshall execution time.

Fig. 5: Data placement management overhead in *iFogSim*.

simulation time and memory footprint; and (ii) the obtained speedup using our parallel implementation of the *Floyd-Warshall* algorithm.

TABLE VII: Speedup of the parallel version of Floyd-Warshall algorithm as compared to one thread C version.

Number of threads	Speedup
2	1.84
4	3.65
8	5.42
16	11.83
32	12.99

1) Simulation time: Figure 5 (a) shows the required time (in seconds) for *iFogSim* to simulate three Fog infrastructure configurations with 1000, 2000 and 3000 gateways respectively.

We observe that the simulation time increases exponentially with the number of gateways. For instance, in case of an infrastructure involving 1000 gateways, *iFogSim* took about 800 seconds, whereas, it took about 13400 seconds to simulate an infrastructure with 3000 gateways, which means an increase by a factor 16 for 3 times more gateways.

Concerning the time overhead added to *iFogSim* by integrating data placement management, it is about 5%, 7% and 13% for an infrastructure with 1000, 2000, 3000 gateways, respectively. This low time overhead is due to the additional processing of data placement management events such as retrieving the storage node location of a given Tuple or storing the data in a specified node.

2) Memory footprint: Figure 5 (b) shows the average and the peak (max) memory utilization of *iFogSim* with and without data placement management. We observe that data placement management has a very small impact on this metric for various infrastructure configurations. This small overhead is due to the memory allocation of the added events for data placement management. The peak memory utilization happens during the simulation initialization phase (sending events for infrastructure creation), and it is not so much impacted by our extension. On the other hand, we see that *iFogSim* memory utilization is acceptable as it takes about 4 GB to simulate a Fog infrastructure encompassing 3000 gateways with 15 sensors and 5 actuators attached to each one.

3) Floyd Warshall execution time: Figure 5 (c) shows the required execution time (in seconds) to compute all shortest paths in a Fog infrastructure encompassing about 5000 Fog nodes. This execution time is measured for two different implementations of the *Floyd-Warshall* algorithm; the first one is the sequential implementation existing in *iFogSim*; and the second one is our parallel implementation with various configurations for the number of threads. Table VII illustrates the obtained speedup using various number of threads for our implementation.

One may observe that using the existing implementation, the algorithm takes about 2000 seconds to find all shortest paths. As Fog infrastructures are dynamic and change their topologies frequently, this execution time can be considered to be too high for a runtime execution. Our implementation with a single thread decreases the execution time to 780 seconds, see Figure 5 (c), giving a speedup of about 2.5 compared to the initial Java implementation in *iFogSim*. Furthermore, as shown in Table VII, when increasing the number of threads, the execution time accelerates significantly. In fact, using 32 threads, the algorithm takes about 60 seconds to find all shortest paths, decreasing the computation time by 12 times as compared to the C implementation with one thread, and by 32 times as compared to the initial Java implementation.

V. Conclusion

In this paper, we presented an extension to *iFogSim* to be able to model and simulate scenarios with data placement policies. Using our extension, users can deploy their solutions to manage data placement in the simulated Fog infrastructure driven by objectives such as minimizing service latency, network congestion or energy consumption. The extension consists in adding three main functionalities to *iFogSim*: (i) data placement computation using linear programming methods, (ii) Fog infrastructure decomposition into several parts offering the possibility to place data separately within a sub-part of the system hence reducing the placement computation and (iii) a smart city scenario with various workload distribution for evaluation purpose. The extension also includes a parallel implementation of the *Floyd-Warshall* algorithm to reduce shortest paths computation time. As shown in the evaluation section, our extension has a small overhead in terms of simulation time and memory utilization.

An interesting direction for future work is to make our extension able to solve several data placement subproblems

in a parallel way in case of infrastructure partitioning hence reducing the data placement computation time. More generally, we would like to make parallel simulations on iFogSim possible whether on a single machine or in a cluster, hence reducing the simulation time which might be very high in case of large Fog infrastructures (e.g. iFogSim takes 13400 seconds to simulate an infrastructure with 3000 Fog nodes) as shown in the evaluation section.

The source code of our extension is made available in: https://github.com/medislam/iFogSimWithDataPlacement.git[1]

REFERENCES

[1] Ibm ilog cplex optimization toolkit. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud. Accessed: 2017-09-20.

[2] Metis, a graph partition tool. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview. Accessed: 2017-07-19.

[3] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne. Fit iot-lab: A large scale open experimental iot testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464, 2015.

[4] S. R. Balachandar and K. Kannan. A new heuristic approach for the large-scale generalized assignment problem. *Int J Math Comput Phys Elect Comput Eng*, 3:969–974, 2009.

[5] A. Botta, W. de Donato, V. Persico, and A. Pescapé. Integration of cloud computing and internet of things. *Future Gener. Comput. Syst.*, 56(C):684–700, 2016.

[6] C. C. Byers and P. Wetterwald. Fog computing distributing data and intelligence for resiliency and scale necessary for iot: The internet of things (ubiquity symposium). *Ubiquity*, 2015:4:1–4:12, Nov. 2015.

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, 2011.

[8] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998.

[9] C. Dsouza, G. J. Ahn, and M. Taguinod. Policy-driven security management for fog computing: Preliminary framework and a case study. pages 16–23, 2014.

[10] M. Etemad, M. Aazam, and M. St-Hilaire. Using devs for modeling and simulating a fog computing environment. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 849–854, 2017.

[11] V. Gazis, A. Leonardi, K. Mathioudakis, K. Sasloglou, P. Kikiras, and R. Sudhaakar. Components of fog computing in an industrial internet of things context. pages 1–6, 2015.

[12] S. Godard. Sysstat utilities home page. *sebastien.godard.pagesperso-orange.fr*, 2010. Accessed: 2017-11-10.

[13] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *Technical Report CLOUDS-TR-2016-2, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia*, 2016.

[14] E. K. Markakis, K. Karras, N. Zotos, A. Sideris, T. Moysiadis, A. Corsaro, G. Alexiou, C. Skianis, G. Mastorakis, C. X. Mavromoustakis, and E. Pallis. Exegesis: Extreme edge resource harvesting for a virtualized fog environment. *IEEE Communications Magazine*, 55(7):173–179, 2017.

[15] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran. Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. In *in Proceedings of 2017 IEEE Fog World Congress (FWC '17)*, 2017.

[16] M. I. Naas, L. Lemarchand, J. Boukhobza, and P. Raipin. A graph partitioning-based heuristic for runtime iot data placement strategies in a fog infrastructure. In *SAC 2018: Symposium on Applied Computing*. ACM, 2018.

[17] M. I. Naas, P. R. Parvedy, J. Boukhobza, and L. Lemarchand. ifogstor: An iot data placement strategy for fog infrastructure. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, volume 00, pages 97–104, 2017.

[18] P. G. V. Naranjo, Z. Pooranian, M. Shojafar, M. Conti, and R. Buyya. Focan: A fog-supported smart city network architecture for management of applications in the internet of everything environments. *arXiv preprint arXiv:1710.01801*, 2017.

[19] C. V. Networking. Ciscoglobal cloud index: forecast and methodology, 2015-2020. white paper, 2017.

[20] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar. Towards qos-aware fog service placement. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 89–96, 2017.

[21] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya. A toolkit for modelling and simulating data grids: An extension to gridsim. *Concurr. Comput. : Pract. Exper.*, 20(13):1591–1609, 2008.

[22] R. L. R. Thomas H. Cormen, Charles E. Leiserson and C. Stein. Introduction to algorithms. pages 693–700, 2009.

[23] S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog computing: Platform and applications. In *Proceedings of the 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, HOTWEB '15, pages 73–78, Washington, DC, USA, 2015. IEEE Computer Society.

[1] Some work is being done on the documentation of the extension