iFogStor: An IoT Data Placement Strategy for Fog Infrastructure

Conference Paper · May 2017

DOI: 10.1109/ICFEC.2017.15

4 authors:

Mohammedislam Naas
Université de Bretagne Occidentale
5 PUBLICATIONS **27** CITATIONS
SEE PROFILE

Philippe Raipin Parvedy
Orange Labs
30 PUBLICATIONS **265** CITATIONS
SEE PROFILE

Jalil Boukhobza
Université de Bretagne Occidentale
110 PUBLICATIONS **237** CITATIONS
SEE PROFILE

Laurent Lemarchand
Université de Bretagne Occidentale
42 PUBLICATIONS **88** CITATIONS
SEE PROFILE

iFogStor: an IoT Data Placement Strategy for Fog Infrastructure

Mohammed Islam Naas, Philippe Raipin Parvedy
Orange
Rennes, France
Email: {mohammedislam.naas, philippe.raipin}@orange.com

Jalil Boukhobza, Laurent Lemarchand
Univ. Bretagne Occidentale UMR 6285, Lab-STICC
F-29200 Brest, France
Email: {boukhobza, laurent.lemarchand}@univ-brest.fr

***Abstract*—Internet of Things (IoT) will be one of the driving application for digital data generation in the next years as more than 50 billions of objects will be connected by 2020. IoT data can be processed and used by different devices spread all over the network. The traditional way of centralizing data processing in the Cloud can hardly scale because it cannot satisfy many of the latency critical IoT applications. In addition, it generates a too high network traffic when the number of objects and services increase. Fog infrastructure provides a beginning of an answer to such an issue. In this paper, we present a data placement strategy for Fog infrastructures called iFogStor. The objective of iFogStor is to take profit of the heterogeneity and location of Fog nodes to reduce the overall latency of storing and retrieving data in a Fog. We formulated the data placement problem as a Generalized Assignment Problem (GAP) and proposed two ways to solve it: 1) an exact solution using integer programming and 2) a heuristic one based on geographical zoning to reduce the solving time. Both solutions proved very good performance as they reduced the latency by more than 86% as compared to a Cloud based solution and by 60% as compared to a naive Fog solution. Using geographical zoning heuristic can allow solving problems with large number of Fog nodes efficiently and in a couple of seconds making iFogStor feasible in runtime and scalable.**

***Index Terms*—Internet of Things, Fog, Data Placement, Storage, Optimization, Generalized Assignment Problem.**

I. Introduction

Internet of Things (IoT) consists in making every smart object as part of the Internet, such as sensors, wearables, smart phones, cameras and vehicles. In the next few years, IoT devices will invade the world as many tens of billions of objects will be connected by 2020 [13]. This large number of objects will generate a massive amount of data; supposedly 40% of all data traffic will come just from sensors [12].

Nowadays, IoT data are generally processed and stored in the Cloud [9]. This Cloud-centric approach satisfies most of current IoT requirements like ubiquity and high availability with regards to compute and storage capabilities. However, with the increasing number of smart objects and the amount of data produced, the aforementioned approach will hardly scale because it generates high network traffic which increases the latency, thus degrading the Quality of Service (QoS) [13]. In effect, for several IoT applications such as Internet of vehicles (IoV), e-health, and industry 4.0, service latency is very critical and the least delay may cause critical issues [9].

To cope with the aforementioned challenges, the paradigm of Fog computing has been proposed. It aims to extend services provided by the Cloud down to the network edge [8]. It provides a hierarchical, dense and geographically distributed architecture to store and process data in network equipments located between end-users' smart objects and Cloud data-centers. Unlike the Cloud, the Fog has the ability to support latency-critical IoT applications requiring short response times. In fact, using Fog computing can allow for a drastic reduction of the overall network latency [13].

In Fog computing, processing and storage components, called Fog nodes, are heterogeneous in terms of performance and storage capabilities. They may be resource limited equipments such as set-top boxes and gateways, or resource enabled equipments such as base stations and Points of Presence (PoP) [14, 20]. The geographical distance between Fog nodes and smart objects can also be very variable, from few meters as for set-top boxes up to hundreds or even thousands of kilometers like PoPs. Furthermore, Fog nodes having modest resources are located near end-users' smart objects and they provide low latency, whereas the best performing nodes are located far from the end-users' smart objects and they expose a high latency. Because of this heterogeneity, investigating how to place data in the Fog infrastructure is crucial to minimize both operation cost and latency, and to maximize the throughput [20]. This is the challenge we are trying to tackle in this paper.

Some recent studies have been devoted to reduce response time and network latency in the context of Fog computing and IoT. For example, Aazam et al. [3] propose a Fog computing framework for emergency notification in health care and asset environments. They rely on a, so called, smart gateway for offloading and pre-processing purposes. In [17], Vural et al. use in-network caching (within routers) of IoT data in order to reduce the data retrieving latency.

In fact, existing studies used the Fog just as a one level infrastructure located prior to the Cloud essentially used for caching, offloading or pre-processing. This was done without investigating how data need to be placed within this infrastructure. Indeed, Fog infrastructure is geo-distributed, hierarchical and heterogeneous in terms of performance (processing and storage). Misplacing data in such architecture can increase the network latency, especially in case where data are shared and used between many nodes.

In this paper, we propose iFogStor, a Fog-aware runtime strategy for placing IoT data in a Fog infrastructure, including data-centers, in order to minimize the overall storage and service latencies. iFogStor considers the overall hierarchical infrastructure for data placement sake, it also takes into account the heterogeneity of devices and latencies between nodes. The contributions of this paper are the following:

- We have provided a full modeling of the data placement problem in a Fog. The problem was formulated as a Generalized Assignment Problem (GAP) [4].
- We have proposed two solutions to solve the GAP formulated problem, one exact solution and a heuristic approach based on geo-partitioning the Fog infrastructure. The heuristic solution allows to limit the nodes that can store a given set of data for very big problem instances.
- We provided an overall framework to solve the data placement problem. We have upgraded *iFogSim* [9] to take into account our problem formulation and used *CPLEX MILP*[1] solver to find out the best placement in a reasonable time.

Our allocation strategy proved good performance. It reduced the overall latency by more than 86% compared to the traditional Cloud approach, and by 60% compared to the naive Fog computing approach. To the best of our knowledge, this is the first contribution towards data placement strategy taking into account geo-distribution, hierarchical and heterogeneous nature of a Fog architecture for storage latency optimization.

The rest of this paper is organized as follows: first, we give a brief introduction to Fog computing and introduce related work in Section 2. In Section 3, we formulate the problem and describe our solutions. We evaluate the proposed solutions in Section 4 and conclude in Section 5.

II. Background and Related Work

In this section, we first give a definition of Fog we comply with, then we present its generic architecture, and we finally describe some related work.

A. Fog Computing

1) Definition: There are several definitions of Fog computing. While some researchers consider Fog as being located only at the network edge [14, 19, 20], others consider an architecture that includes both network edge and network core [5, 8–10, 18]. We comply with the second description and we rely on the definition given by Bonomi et al. [5] where Fog computing is considered as *"a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud computer data-centers, typically, but not exclusively located at the edge of network."*.

2) Generic architecture: A generic Fog computing architecture is shown in figure 1. It presents a hierarchical structure. The bottommost layer encompasses wireless, smart, mobile or fixed end-users objects such as sensors, robots, smart phones, and cameras. Components from this layer use the above layer to connect with other elements (in the same layer) as well as with IoT services implemented in both network and Cloud layers. The network layer covers several sub-layers (network's edge, aggregation and core). It involves network components

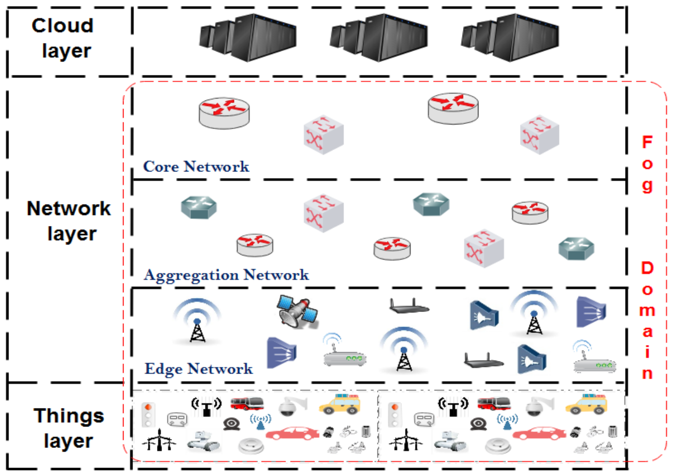


Fig. 1: Fog computing architecture.

such as gateways, switches, routers, PoPs and base stations. This layer is used also for hosting IoT applications that require low latency as well as performing data aggregation, filtering and pre-processing before sending to the Cloud. Finally, the last layer is the uppermost one, it involves powerful servers distributed within distant data-centers to host applications for big data analytics and permanent storage.

B. Related Work

Many research efforts were done to address the issue of reducing latency of IoT applications in the context of Fog computing. These efforts can be broadly divided into two classes: (i) those optimizing the placement of IoT applications such as in [16], and (ii) those dealing with IoT data placement. In this section, we focus on the second category.

In [17], Vural et al. investigated the cost benefits of in-network caching (within routers) of IoT data in order to reduce the data retrieving latency. Cache nodes are selected to host data items according to the trade-off between (i) the multi-hop delivery from the data source location to a requester, and (ii) the expected loss in the delivered data *freshness*. In [3], Aazam et al. propose E-HAMC, a service architecture for emergency notification. To overcome the network latency issue and provide quick notifications, E-HAMC is deployed in both Fog (within a smart phone) and Cloud. The former processes emergency events and allows users to make quick notifications, while the latter serves for historical analytics. In [13], Sakar et al. modeled the service latency on a Fog infrastructure. In order to prove the suitability of Fog computing, they provide a comparison between Fog and Cloud in term of the network latency. Their results show that using the Fog, the network latency is decreased by 50.09%.

In this work, we propose a strategy to place IoT data on a Fog infrastructures to reduce the overall latency. Different from state-of-the-art approaches, the strategy that we propose relies on the heterogeneity of nodes and their locations within the Fog to find out the better location for data storage. It also takes into account data sharing between IoT services.

III. iFogStor: A Data Placement Strategy for Fog Infrastructure

A. iFogStor overview

In this section, we give an overview of our approach toward IoT data placement in a Fog infrastructure which aims to minimize the overall system latency.

As shown in figure 2, the system architecture that we consider comprises: a set of sensors, a set of Fog nodes, a set of data-centers and a set of IoT services (application instances) which can be executed in several Fog nodes and data-centers.

We assume that each application instance is deployed into a specific Fog node which is selected by an application instance orchestrator according to various criteria such as performance, or security, as proposed in [16]. Application instance placement is out of the scope of this paper.

Sensors (data producers) collect data from the real world. Collected data are sent through gateways to associated application instances (data consumers) for analysis and/or processing sake. Application instances may consume resulting (produced) data from other application instances (which in this case play also the role of data producers). IoT data can be stored in different locations in this Fog architecture. Practically, each Fog node can be used for data storage (data hosts). These nodes have limited capacities.

In our approach, we assume that the system has knowledge about: (i) the mapping between data producers and their data consumers (which consumers use data from which producers), (ii) the existing latencies between Fog nodes. Such knowledge can be inferred by deploying a software component which monitors the state of Fog nodes, latencies, storage capacities, and application instances, as proposed in [13].

In such a system, a naive solution towards reducing the overall system latency, is to place data close to their consumers. However, many issues need to be addressed. The first one is that data can be shared by many consumers in different locations. The second one is that data consumer location may change over time (or be created and removed dynamically) which highlights the need to dynamically update data placement according to new locations. Finally, each Fog node has limited capacity and is reached with a specific latency. Our approach tries to take into account the aforementioned constraints in order to achieve the best data placement strategy. We propose an algorithm which finds, for the generated data set, the Fog storage nodes which minimize the overall service latency while storing and retrieving this data set.

Each data item can be stored then reached with a given latency according to its location and characteristics. This latency is composed of two parts: (a) the time required to move data from the initial producer to the selected Fog storage node (ts in figure 2), and (b) the time required to retrieve it by the associated consumers (e.g. tr_1 and tr_2 in figure 2).

The data placement strategy we propose is to be deployed within a powerful node for runtime execution. This node should have access to information related to data flow, application instances' locations, existing network latencies and existing free storage capacities. Our data placement strategy can be triggered based on specific events, notably: emergency or deletion of nodes, and application instances reallocation.

As in IoT environment, data are periodically and permanently generated. Hence, to free storage space in a fog node for new incoming data, we assume that the system employs an archiving mechanism which migrates old data toward the Cloud, as proposed in [17].

Since in the problem we are trying to solve, we have data with different sizes to store over many nodes that can have different properties, we have modeled data placement as a GAP-like problem which is detailed in the next section.

B. Generalized Assignment Problem (GAP)

GAP is a well known NP-Hard problem in the combinatorial optimization literature. It consists in finding the best assignment of n tasks to m agents (e.g. n jobs to m processors) while minimizing the overall cost. Agents have different capacities described by $\{c_1, c_2, ..., c_m\}$ and each task has a different size and a different cost according to the agent it is assigned to. Sizes and costs are denoted $\{s_{1,1}, ..., s_{1,m}, s_{2,1}, ..., s_{2,m}, ..., s_{n,1}, ..., s_{n,m}\}$ and $\{v_{1,1}, ..., v_{1,m}, v_{2,1}, ..., v_{2,m}, ..., v_{n,1}, ..., v_{n,m}\}$ respectively. The solution should respect two constraints: (i) each task should be assigned to one agent, (ii) agents' capacities should not be exceeded [4]. GAP can be formulated as follows:

$$\begin{cases} Minimize & \sum_{i=1}^{n}\sum_{j=1}^{m} v_{i,j}.x_{i,j} \\ Subject\ to & \\ & \sum_{j=1}^{m} x_{i,j} = 1 \qquad \forall i \in I = [1..n] \\ & \sum_{i=1}^{n} s_{i,j}.x_{i,j} \leq c_j \quad \forall j \in \ J = [1..m] \\ & x_{i,j} \in \{0,1\} \qquad \forall i \in I, \forall j \in J \end{cases}$$

with $x_{i,j} = 1$ meaning that the task i is assigned to the agent j, otherwise $x_{i,j} = 0$.

C. Preliminaries

1) Notation: The used notation is summarized in table I.

2) Data actors: In this work, we have three main classes of actors from data point of view defined as follows:

- **DataHost** (dh)**:** it represents a storage node. It can be a Fog node or a data-center. This storage node may be located in any layer (except layer 0), see figure 2. Indeed, we do not use sensors for storage purpose.
- **DataProd** (dp)**:** a data producer is a physical or a logical entity that is able to produce output data. This includes sensors and service instances (that produce output data after processing input ones). *DataProd* can be located in various layers in figure 2.
- **DataCons** (dc)**:** it represents entities that may request data for reading/processing purposes including service instances implemented in various layers in figure 2.

One can note that a given Fog node can play the role of DataHost, DataProd, and DataCons at the same time.

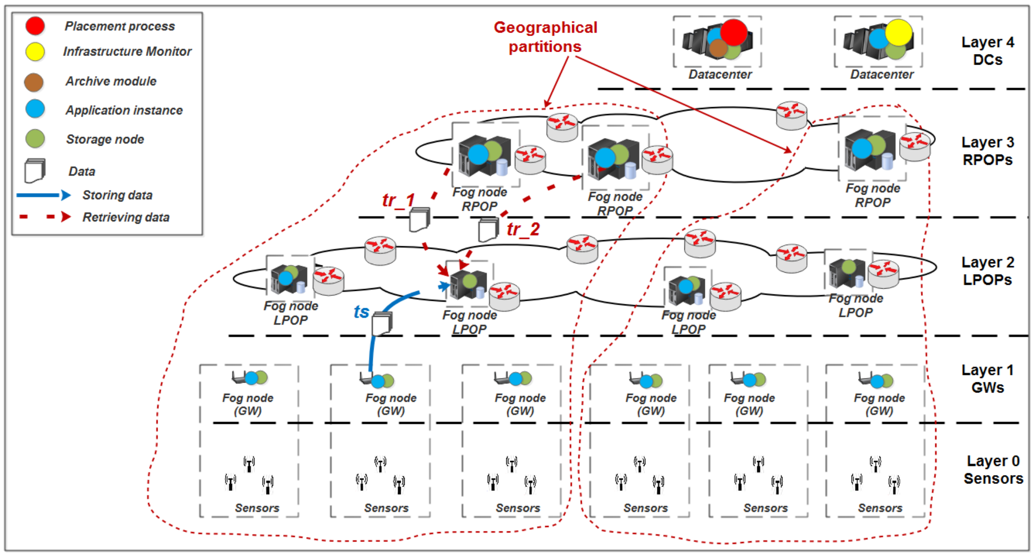


Fig. 2: System architecture.

TABLE I: Notation dictionary

Notation	Description
S	The overall system
dp_i	DataProd, an entity that can produce data
dh_j	DataHost, an entity that can host data
dc_k	DataCons, an entity that can use data
DH	DataHost set
DP	DataProd set
DC	DataCons set
d_i	Data produced by dp_i
s_{d_i}	size of d_i (in bytes)
D	Data set produced by DP
A_D	Assignment matrix of D to DH
$x_{i,j}$	= 1 if d_i is assigned to dh_j
$Overall\_Latency$	The overall system latency of storing D at DH and retrieving it by DC
b	Minimum data exchange granularity
TS_D	Matrix containing the time latencies to transfer D from DP to DH (for storing)
$ts_{i,j}$	Time required to transfer b from dp_i to dh_j
$ts_{d_{i,j}}$	Time required to transfer d_i from dp_i to dh_j
TR_D	Matrix containing the time latencies to transfer D from DH to DC (for requests)
$D_{k,j}$	The requested data subset of dc_k from dh_j
$s_{D_{k,j}}$	Data size of $D_{k,j}$ (in bytes)
$tr_{k,j}$	Time required to transfer b from dh_j to dc_k
$tr_{D_{k,j}}$	Time required to transfer $D_{k,j}$ from dh_j to dc_k
PC_D	Matrix mapping data producers to theirs data consumers
f_{dh_j}	Free storage capacity of dh_j

3) Assumptions: This subsection gives the set of assumptions made to formulate the data placement problem:

- Every dp has a subset of related dc that request its data.
- The basic (minimum) transfer unit for data exchange between data actors is denoted b (in bytes).
- Between each (dp_i, dh_j) and (dh_j, dc_k) pairs, there exists a basic transfer time denoted $ts_{i,j}$ and $tr_{k,j}$, respectively. It represents the required time to move then store b from dp_i to dh_j and to retrieve it from dh_j to dc_k, respectively.
- Data d_i produced by dp_i will be stored in only one *DataHost* (no data replication considered).

D. Data Placement Model

1) Problem formulation: Let S be a system composed of a set of *DataHost* $DH = \{dh_1, dh_2, ..., dh_n\}$, a set of *DataProd* $DP = \{dp_1, dp_2, ..., dp_l\}$ and a set of *DataCons* $DC = \{dc_1, dc_2, ..., dc_m\}$. DP produces the data set D with $D = \{d_1, d_2, ..., d_l\}$ where d_i is produced by dp_i. Each d_i has a size (in bytes) denoted s_{d_i}. The produced data set D will be stored in DH and is supposed to be used by DC.

Let A_D be the matrix assigning D to DH. This matrix maps each data d_i to the dh_j which stores it. Coefficient $x_{i,j} = 1$ means that d_i is stored in dh_j, otherwise $x_{i,j} = 0$.

$$A_D = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{l,1} & \cdots & x_{l,n} \end{bmatrix}, x_{i,j} \in \{0,1\} \qquad (1)$$

Let TS_D be the matrix containing the time (in milliseconds) required to transfer D from DP to DH. Each coefficient describes the transfer time of each $d_i \in D$ from its producer dp_i to its destination storage node dh_j.

$$TS_D = \begin{bmatrix} ts_{d_{1,1}} & \cdots & ts_{d_{1,n}} \\ \vdots & \ddots & \vdots \\ ts_{d_{l,1}} & \cdots & ts_{d_{l,n}} \end{bmatrix} \qquad (2)$$

Each d_i will be requested by one ore more dc_k, either for simply reading or for processing purposes. Let PC_D be the matrix that links each d_i with its consumers. A coefficient $pc_{i,k} = 1$ means that d_i is used/requested by dc_k. Note that a given dc_k may require several data items from the same dh_j . Let $D_{k,j} \subset D$ be the data subset that will be requested by dc_k from dh_j, and $tr_{D_{k,j}} \in TR_D$ is its retrieving time.

$$PC_D = \begin{bmatrix} pc_{1,1} & \cdots & pc_{1,m} \\ \vdots & \ddots & \vdots \\ pc_{l,1} & \cdots & pc_{l,m} \end{bmatrix}, pc_{i,k} \in \{0,1\} \tag{3}$$

$$TR_D = \begin{bmatrix} tr_{D_{1,1}} & \cdots & tr_{D_{1,n}} \\ \vdots & \ddots & \vdots \\ tr_{D_{m,1}} & \cdots & tr_{D_{m,n}} \end{bmatrix} \tag{4}$$

Storing D in DH and requesting it by DC generates the following overall latency:

$$Overall\_Latency = \sum_{ts_{d_{i,j}} \in TS_D} ts_{d_{i,j}} + \sum_{tr_{D_{k,j}} \in TR_D} tr_{D_{k,j}} \tag{5}$$

For simplicity, we considered the overall latency to be the sum of all generated latencies.

Let FC be the vector that represents the free storage capacity of every $dh_j \in DH$:

$$FC = \{f_{dh_1}, f_{dh_2}, ..., f_{dh_n}\} \tag{6}$$

2) Constraints: In this work, we have two main constraints:

- The amount of data that is assigned to a given dh_j must be lower than its free storage capacity f_{dh_j}:

$$\forall f_{dh_j} \in FC : \sum_{i \in [1..l]} s_{d_i}.x_{i,j} \leq f_{dh_j} \tag{7}$$

- Each d_i must be assigned to only one dh_j. Thus, the sum of values of each row in A_D must be equal to 1:

$$\forall i \in [1..l] : \sum_{j \in [1..n]} x_{i,j} = 1 \tag{8}$$

3) Latency model: In this subsection, we provide a model for the overall generated latency to evaluate the data placement defined by A_D. As mentioned earlier, the overall latency is the total of the required time for storing D at DH and transferring it from DH to DC.

Firstly, we model the transfer time $ts_{d_{i,j}} \in TS_D$ generated while storing d_i in dh_j (see ts in figure 2). This latency is calculated as follows:

$$ts_{d_{i,j}} = \left\lceil \frac{1}{b}.s_{d_i} \right\rceil .ts_{i,j} \tag{9}$$

Secondly, we model the transfer time $tr_{D_{k,j}} \in TR_D$ that is generated by dc_k while retrieving its requested data $D_{k,j}$ from dh_j (see tr_1 and tr_2 in figure 2). $tr_{D_{k,j}}$ is calculated as follows:

$$tr_{D_{k,j}} = \left\lceil \frac{1}{b}.s_{D_{k,j}} \right\rceil .tr_{k,j} \tag{10}$$

The requested amount of data $s_{D_{k,j}}$ is calculated by:

$$s_{D_{k,j}} = (s_{d_1}.pc_{1,k}, ..., s_{d_l}.pc_{l,k}). \begin{pmatrix} x_{1,j} \\ \vdots \\ x_{l,j} \end{pmatrix} \tag{11}$$

By substituting (11) in (10) we have:

$$tr_{D_{k,j}} = tr_{k,j}.(\left\lceil \frac{1}{b}.s_{d_1} \right\rceil .pc_{1,k}.x_{1,j} + ... + \left\lceil \frac{1}{b}.s_{d_l} \right\rceil .pc_{l,k}.x_{l,j}) \tag{12}$$

Then, after replacing (12) and (9) in (5), we have:

$$Overall\_Latency = \sum_{i \in [1..l]} \sum_{j \in [1..n]} \alpha_{i,j}.x_{i,j} \tag{13}$$

With $\alpha_{i,j} = \lceil \frac{1}{b}.s_{d_i} \rceil . \left(ts_{i,j} + \sum_{k \in [1..m]} tr_{k,j}.pc_{i,k} \right)$

The goal of our placement strategy is to find the best assignment of D to DH in order to minimize the overall latency. This means finding (1) that minimizes (13). This problem can be modeled as follows:

$$\begin{cases} Minimize & \sum_{i \in [1..l]} \sum_{j \in [1..n]} \alpha_{i,j}.(x_{i,j}) & \\ Subject\ to & & \\ & \sum_{i \in [1..l]} s_{d_i}.x_{i,j} \leq f_{dh_j} & \forall j \in J = [1..n] \\ & \sum_{j \in [1..n]} x_{i,j} = 1 & \forall i \in I = [1..l] \\ & x_{i,j} \in \{0,1\} & \forall i \in I, \forall j \in J \end{cases}$$

E. Data Placement Problem Solving

In this work, we define the data placement as an Integer Programming problem and we propose two solutions to solve it. The first one is an exact solution whereas the second one is a divide-and-conquer heuristic approach to reduce the problem solving time in case of very large scale applications. In both, we used the *CPLEX MILP* solver.

*1) **iFogStor**, the Exact Solution:* This solution consists in solving the placement problem as a single Integer Program using *CPLEX MILP* solver. It finds the optimal placement of the generated data on Fog storage nodes to optimize the overall latency. As mentioned previously, the data placement problem is NP-Hard. Thus, in case of large scale applications, the problem solving time may be unacceptable. Indeed, data placement is to be executed in runtime.

*2) **iFogStorZ**, the Heuristic Solution:* To cope with the issue mentioned above, we used problem partitioning as an approximation heuristic. More specifically, we relied on geography as a partitioning criteria to reduce the problem solving time. In fact many state-of-the-art studies have shown that geographical location is a relevant criteria for IoT applications [15]. For instance, in smart environment domains such as home-automation, smart city, e-health, industry 4.0, IoV, etc, the generated data is processed locally for making fast notifications or triggered events. So geographical location seems to be a relevant criteria for our heuristic.

In iFogStorZ, a geographical partition, also called a *zone*, contains Fog nodes that are located within the same geographical area. Each zone defines a sub-problem that can be solved separately. Exact solutions of sub-problems are aggregated to make a global solution. This solution may not find the best data placement with the optimal latency, but it decreases drastically the problem solving time. The trade-off between solving time and solution optimality is investigated in the evaluation section by varying the number of zones.

There may be many ways to define how to perform zoning. In this work, we chose to define Regional PoPs [1] (RPOP) as points of partitioning. As shown in figure 2, these equipments are located in layer 3. This way, the maximum number of partitions one can have is equal to the number of RPOPs. However RPOPs can be grouped to compose a given zone.

IV. Application and Experimental Evaluation

Smart city is a major application domain of IoT [8]. It aims at enhancing service performance and the wellness of urban citizens by providing an intelligent way to manage transportation, homes, health and energy, etc. Smart city encompasses several use-cases related to previous examples towards a smart management (energy, traffic, home, etc). This smart management may involve information sharing between services. For instance, smart energy can use information provided by smart home, smart building or smart traffic.

To evaluate our data placement solution, we considered a generic use case of smart city, where multiple types of sensors send data to a variety of IoT applications widespread across Fog and Cloud nodes.

In our use case, we consider an infrastructure that encompasses a set of sensors, a set of Fog nodes and a set of data-centers (as shown in figure 2). Fog nodes consist of gateways (GW), local PoPs (LPOP) and RPoPs arranged in a hierarchical topology. For the purpose of our experiments, we fixed the number of sensors managed by a GW to 100. Note that sensors transmit their data only to the associated GW. Thus, they do not need a placement optimization.

Data transmitted by sensors are consumed by IoT applications implemented in both Fog nodes and Cloud data-centers.

The data flow considered in our use case is described in this paragraph. First, sensors collect data from the real world and transmit them to application instances located in GWs. Then, each GW's application instance processes incoming data and sends the result to one or several application (up to 5) instances according to the number of $DataCons$ per $DataProd$. These application instances are selected randomly form the possible ones which are located within the same geographical zone (i.e. the associated LPOP, the associated RPOP, or the associated RPOP's LPOPs). In the same way, application instances located in LPOPs, after processing incoming data, send the output data to a subset of possible application instances located in RPOPs or DCs. Application instances located in RPOPs send their output data to a subset of possible DCs' application instances. Finally, application instances located in DCs process the incoming data and store the result locally for further historical and business processing sake.

A. Experimental Tools and Setup

We used *iFogSim* [9] for simulation and to test our data placement strategy. *iFogSim* is a toolkit for modeling and simulating IoT and Fog computing environments. It considers parameters such as network latency, bandwidth utilization, energy consumption and cost measurement. We upgraded *iFogSim* (i) to formulate the data placement problem (ii) to call *CPLEX MILP* and solve the problem, and (iii) to recover the problem solution and set the data placement allocation.

The evaluation has been achieved using *RuggedPoD* [2], a mini data-center (representing a specific kind of Fog node). This machine has 32 CPU cores and 128 GB of RAM.

As mentioned previously, *DataHosts* have limited storage capacities, and between different nodes there exists a specific latency. Table II summarizes existing free storage capacities on each type of *DataHost*, and table III illustrates existing latencies between the infrastructure components [11].

We suppose that *data actors* transmit their data in packets [13]. In our experiments, we set the size of data packets generated by sensors to 96 bytes [6] and data packets generated by application instances to 960 bytes (i.e. a selectivity of 0.1).

In our experimentations, we used 5 data-centers, 10 RPOPs and 50 LPOPs, and we varied the number of GWs within the range [1,000..7,000] by steps of 1,000. 7,000 GWs represent more than one city [7]. For service composition and data sharing simulation, we varied the number of $DataCons$ per $DataProd$ (denoted cp) from 1 to 5.

B. Methodology

To evaluate our IoT data placement strategy, we used two metrics: 1) the overall latency time generated, and 2) the solving time as our strategy is designed for runtime execution.

We compared two storage modes: **Cloud Storage** and **Fog Storage**. With the first one, all generated data are stored in Cloud data-centers. Then, every $DataCons$ in the system can request part of them for its own use. This case illustrates basically the traditional method to store IoT data. Conversely, in the second mode, the storage system stores the generated data in Fog nodes following three different strategies:

1) **Closest** $DataHost$ (**Closest** dh): data are stored in the closest $DataHost$ to its $DataProd$ in term of latency, a naive approach that does not consider data sharing.
2) **iFogStor**: data are stored in the $DataHost$ which minimizes the overall latency according to our strategy.
3) **iFogStorZ**: data are stored in the $DataHost$ which minimizes the overall latency according to our strategy within a given geographical zone. We used 2, 5 and 10 geographical zones, denoted iFogStorZ(2), iFogStorZ(5), and iFogStorZ(10) respectively. For each case, we investigated the impact of the number of zones on the overall latency and the gain in problem solving time as compared to iFogStor.

C. Results and Discussion

In this section we discuss the simulation results both in terms of overall latency time and problem solving time.

1) Overall latency time: Figure 3 (a) shows the overall latency for each storage mode normalized to the performance of the Cloud storage case. This figure represents the average for the different cases tested (from 1000 to 7000 GWs). We noticed the same performance behavior whatever the number

TABLE II: Free storage capacities.

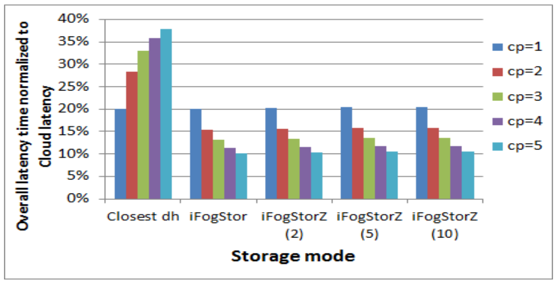
DataHost	Free storage capacity
GW	100 GB
LPOP	10 TB
RPOP	100 TB
DC	10 PB

TABLE III: Latencies.

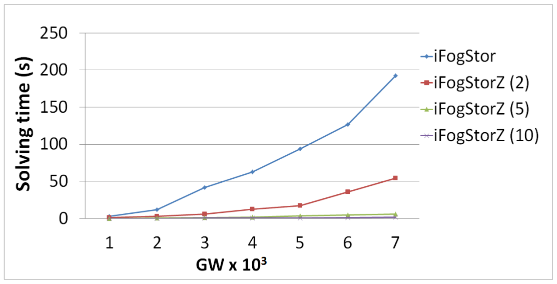
Network link	Latency (ms)
IoT - GW	10
GW - LPOP	50
LPOP - RPOP	5
RPOP - DC	100
RPOP - RPOP	5
DC - DC	100

TABLE IV: Latency comparison.

Storage mode	Avg saving (%)
Closest dh vs Cloud	67.06
iFogStor vs Cloud	87.02
iFogStor vs Closest dh	60.60
iFogStorZ (2) vs iFogStor	-2.39
iFogStorZ (5) vs iFogStor	-3.41
iFogStorZ (10) vs iFogStor	-3.77



(a) Overall latency time.



(b) Problem solving time.

Fig. 3: Simulation results.

of nodes. As mentioned previously, cp defines the number of $DataCons$ that request (consume) the same data. It reflects information sharing between services.

As expected, one can observe that the overall latency of using a Cloud storage is very high as compared to other modes. This is due to the distance between data consumers and data hosts. In addition, this generates a high data traffic while sending/retrieving data to/from the cloud.

As shown in figure 3 (a), using the Closest dh strategy decreases the overall latency by around 80% as compared to the Cloud storage when no data are shared between Fog nodes (cp=1). This illustrates the basic Fog utilization advantage.

However, when cp increases, one can observe that the Closest dh strategy performs less efficiently. The enhancement over the Cloud storage drops to 62% when cp is equal to 5. In fact, this strategy does not take into consideration data sharing and so the issued data placement can generate very high latencies when consumers are far from each other.

Using iFogStor, the overall latency is drastically decreased as compared to both Cloud and the Closest dh strategies. As shown in table IV, iFogStor enhances the overall latency by 87.02% on average over the Cloud strategy and 60.60% on average over the Closet dh strategy. One can notice that for cp equal to 1, iFogStor performs as good as the Closest dh as the best solution is to store data next to the unique consumer. However, when cp is equal to 5, iFogStor still performs well and decreases the overall latency by 73.07% compared to Closest dh and by 89,81% compared to the Cloud storage.

The partitioning approach used in iFogStorZ proved very good results. For instance, as shown in table IV, when using 2 zones the latency is increased by 2.39% compared to iFogStor, whereas when using 10 zones the latency is increased only by 3.77%.

As mentioned previously, we used RPOPs as points of partitioning in iFogStorZ. In fact, interactions between services in different RPOPs are not frequent (contrary to LPOPs). This means that both $DataProd$ and their associated $DataCons$ are generally located under the same RPOP (within the same zone). This is the reason why iFogStorZ performed very well in terms of latency.

TABLE V: Gain in solving time.

Storage mode	Avg gain (%)
iFogStorZ (2) vs iFogStor	77.13
iFogStorZ (5) vs iFogStor	96.34
iFogStorZ (10) vs iFogStor	99.05

2) Solving time: Figure 3 (b) shows the problem solving time in case of iFogStor and iFogStorZ. We observe that in case of iFogStor, the problem solving time is increased dramatically with the increasing number of $DataHost$. For instance, *CPLEX MILP* solver takes around to 200 seconds to solve a data placement problem with 7,000 GWs (7,065 $DataHost$: 7,000 GWs, 50 LPOPs, 10 LPOPs, 5 data-centers).

As previously mentioned, the zoning storage is an approximation heuristic that aims at reducing the problem solving time. We observe that for iFogStorZ, the problem solving time is drastically decreased. As shown in table V, the solving time is reduced by around 77.13% when using two zones. Of course, the solving time increases with respect to the GWs number. For instance, it is around 50 seconds when solving a problem with 7,000 GWs.

We observe also that by increasing further the number of zones, the solving time is reduced. This is due to the problem complexity reduction obtained by splitting the problem. For instance, the solving time has been reduced by around 99.05% when using 10 zones. Indeed, *CPLEX MILP* takes around 1.4 second for a data placement problem with 7,000 GWs, all with a small reduction in terms of quality. This is a very good result for runtime execution.

V. Conclusion and Future Work

This paper presents iFogStor, a runtime IoT data placement strategy for latency reduction in a Fog architecture. iFogStor takes profit of the complexity of Fog infrastructures to adapt data placement according to node characteristics. Those characteristics are related to proper performance of the node, its location, and the nature of data to store (i.e. data sharing). Our contribution was threefold: 1) we have formulated the data placement problem in a Fog as a GAP, 2) we proposed an exact and a heuristic solution based on geographical zoning, and 3) we provided a framework for testing our strategies based on *iFogSim*. iFogStor have shown promising results as it largely enhanced Cloud and naive Fog data placement strategies from a latency point of view. In addition, the heuristic developed gave very good results while drastically reducing solving time even for very large problems.

For future work, more refined models can be considered, e.g. taking into account IoT services variable criticality, or additionally to latency, other optimization criteria as data processing overheads. The archiving service can also be designed jointly with data placement algorithms to come up with a fully optimized solution. Finally, to handle those complex models, more sophisticated partitioning strategies based on clustering and graph partitioning are to be studied.

References

[1] Ibm ilog cplex optimization toolkit. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud. Accessed: 2017-02-22.

[2] Ruggedpod. https://www.indiegogo.com/projects/ruggedpod-outdoor-cloud-computing. Accessed: 2016-12-21.

[3] M. Aazam and E.-N. Huh. E-hamc: Leveraging fog computing for emergency alert service. In *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*, pages 518–523. IEEE, 2015.

[4] S. R. Balachandar and K. Kannan. A new heuristic approach for the large-scale generalized assignment problem. *International Journal of Mathematical and Statistical Sciences*, 1(4), 2009.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[6] C. Cecchinel, M. Jimenez, S. Mosser, and M. Riveill. An architecture to support the collection of big data in the internet of things. In *2014 IEEE World Congress on Services*, pages 442–449, June 2014.

[7] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs. Building a big data platform for smart cities: Experience and lessons from santander. In *2015 IEEE International Congress on Big Data*, pages 592–599, June 2015.

[8] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya. Fog computing: Principals, architectures, and applications. 2016.

[9] 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.

[10] K.P.saharan and A. Kumar. Fog in comparison to cloud: A survey. *International Journal of Computer Applications*, 122(3):10–12, July 2015.

[11] P. Meye, P. Raipin, F. Tronel, and E. Anceaume. Mistore: A distributed storage system leveraging the dsl infrastructure of an isp. In *2014 International Conference on High Performance Computing Simulation (HPCS)*, pages 260–267, July 2014.

[12] A. Noronha, R. Moriarty, K. OConnell, and N. Villa. Attaining iot value: How to move from connecting things to capturing insights, 2014.

[13] S. Sarkar, S. Chatterjee, and S. Misra. Assessment of the suitability of fog computing in the context of internet of things. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2015.

[14] Y. Shi, G. Ding, H. Wang, H. E. Roman, and S. Lu. The fog computing service for healthcare. In *Future Information and Communication Technologies for Ubiquitous HealthCare (Ubi-HealthTech), 2015 2nd International Symposium on*, pages 1–5. IEEE, 2015.

[15] E. van der Zee and H. Scholten. Application of geographical concepts and spatial technology to the internet of things. *Research Memorandum*, 33, 2013.

[16] K. Velasquez, D. P. Abreu, M. Curado, and E. Monteiro. Service placement for latency reduction in the internet of things. *Annals of Telecommunications*, pages 1–11, 2016.

[17] S. Vural, P. Navaratnam, N. Wang, C. Wang, L. Dong, and R. Tafazolli. In-network caching of internet-of-things data. In *2014 IEEE International Conference on Communications (ICC)*, pages 3185–3190, June 2014.

[18] M. Yannuzzi, R. Milito, R. Serral-Graci, D. Montero, and M. Nemirovsky. Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing. In *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 325–329, Dec 2014.

[19] S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78, Nov 2015.

[20] S. Yi, C. Li, and Q. Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, pages 37–42. ACM, 2015.