

Final Year Project

By

Sourjya Mukherjee

(Roll:T91/CSE/164047)

Sumartya Sengupta

(Roll:T91/CSE/164034)

Rachita Mazumder

(Roll:T98/JFT/154003)

IFogStorR

A Data Placement
Strategy
For
Fog Infrastructure

Acknowledgements

We would like to express a special thanks of gratitude to our guide Proff. Rajib Kr Das who gave us the golden opportunity to explore this wonderful project.

Secondly we would also like to thank our parents and the CUCSE department who helped us a lot in finalizing this project within the limited time frame.

Statement Of Authenticity

This is to certify that the contents of this report entitled '**IFogStorR:A Data Placement Strategy for Fog Nodes**' by Sourjya Mukherjee(Roll No: T91/CSE/164047), Sumartya Sengupta(Roll No: T91/CSE/164034) and Rachita Mazumder(Roll No: T98/JFT/154003) is original work carried out and has not been submitted partly or fully to any other Institute or University for the award of any degree or diploma. The information submitted is true and original to the best of our knowledge. By signing our names below we are agreeing that we have read and understood the Statement of Authenticity.

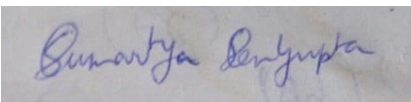
Signatures:-

7/30/2020

X Sourjya Mukherjee

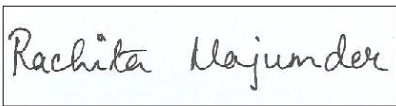
Sourjya Mukherjee
Final Year CS Undergrad, Calcutta University

7/30/2020

X 

Sumartya Sengupta
Final Year CS Undergrad, Calcutta University

7/30/2020

X 

Rachita Mazumder
Final Year CS Undergrad, Calcutta University

Contents

1) Abstract	-----	6
2) Introduction	-----	7
3) Background & Related Work	-----	8
4) IFogStorR Overview	-----	10
5) IFogStor Overview	-----	12
6) IFogStorR: Intuition	-----	13
7) IFogStorR: Problem Formulation	-----	14
8) Preliminaries	-----	15
9) Application & Evaluation	-----	20
10) Results	-----	22
11) Conclusion & Future Work	-----	26
12) References	-----	27

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 inspired by the iFogStor[1] strategy. We have produced an upgrade to iFogStor which we are calling iFogStorR. Unlike iFogStor that modelled the situation as an NP Hard GAP problem, we have modeled the situation as a common Assignment problem solvable in polynomial time thereby making it more feasible for larger networks. On an average our solution showed 11.58 % less execution time when simulated using iFogSim[2] simulator.

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 . This large number of objects will generate a massive amount of data; supposedly 40% of all data traffic will come just from sensors .Nowadays, IoT data are generally processed and stored in the Cloud . 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) . 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 . 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 .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 datacenters. 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.

Background And Related Work

Fog Computing

1) Definition: There are several definitions of Fog computing. While some researchers consider Fog as being located only at the network edge, others consider an architecture that includes both network edge and network core. We comply with the second description and we rely on the definition given by Bonomi et al. 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 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.

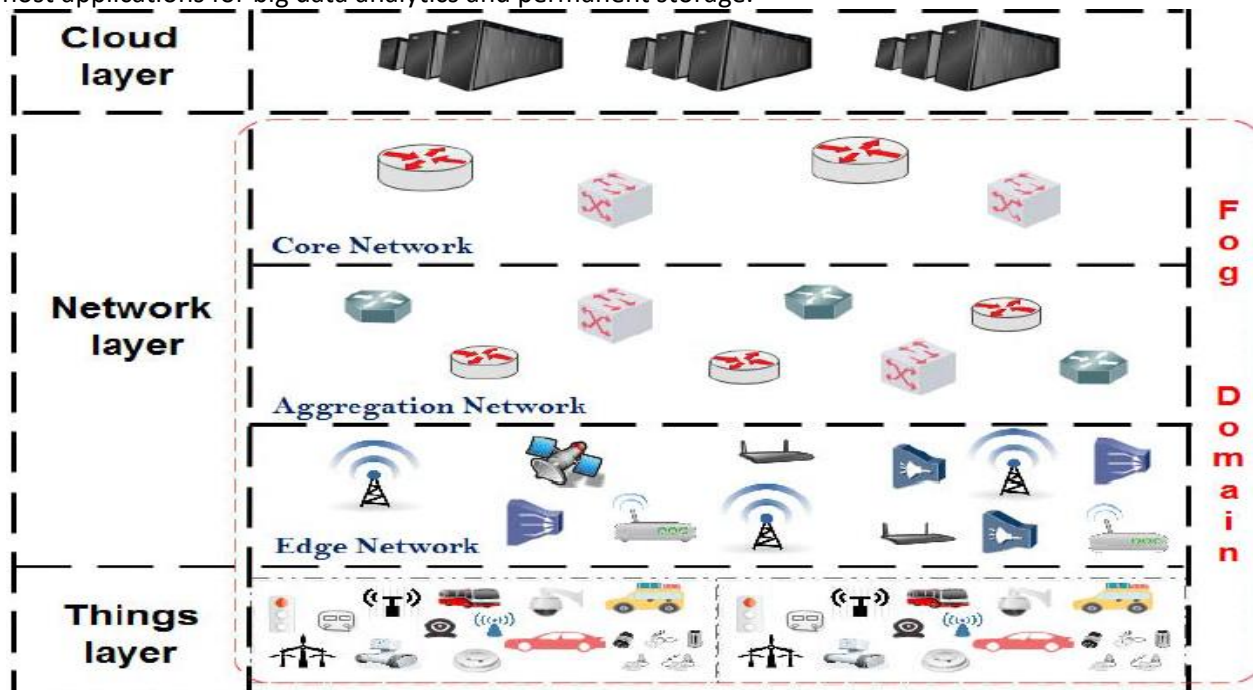


Fig. 1: Fog computing architecture.

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 , and (ii) those dealing with IoT data placement.

In this section, we focus on the second category. In [3], 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 multihop delivery from the data source location to a requester, and (ii) the expected loss in the delivered data *freshness*. In [4], 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 [5], 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.

IFogStorR: A Data Placement Strategy for Fog Infrastructure

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.

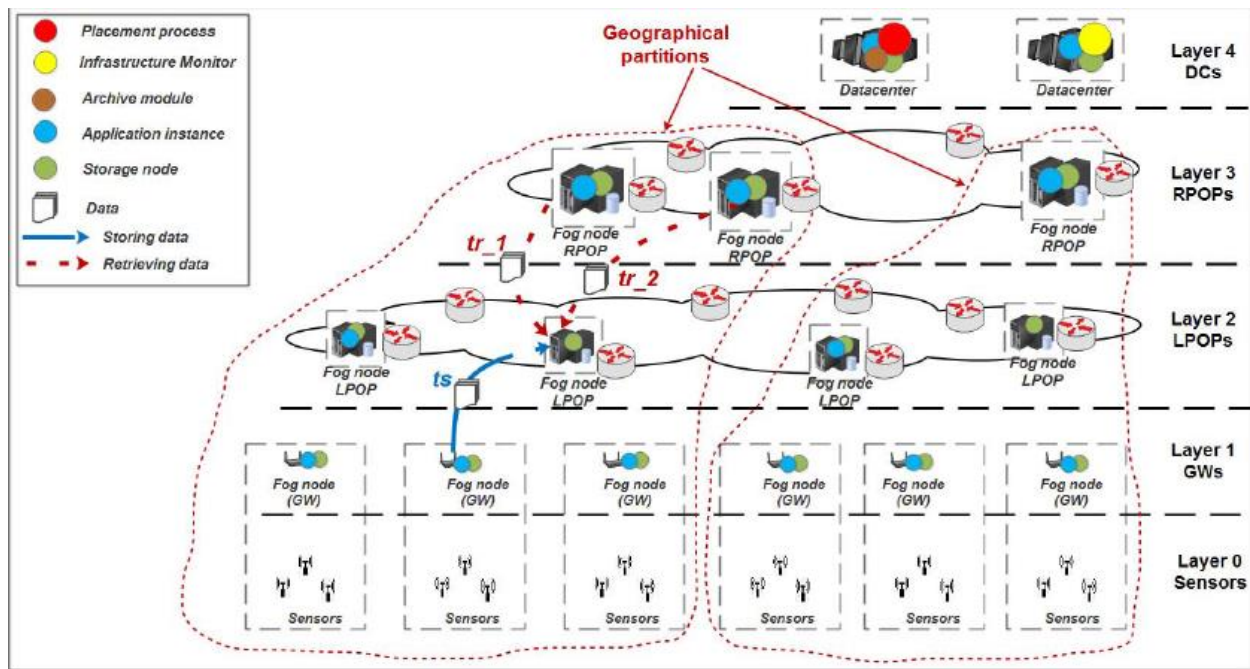


Fig. 2: System architecture

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 [5].

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. Of course our algorithm is obtained by introducing slight modifications in iFogStor. 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 , and
- (b) the time required to retrieve it by the associated consumers .

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.

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 an Assignment problem which is detailed in the next section. But before we proceed further we must understand iFogStor and how they modeled the data placement strategy.

A Brief Overview Of iFogStor

IFogStor[1] modeled the data placement scenario into a GAP-like(Generalized Assignment) problem.

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 $\{c_1, c_2, \dots, c_m\}$ and each task has a different size and a different cost according to the agent it is assigned to. Sizes and costs are denoted $\{s_{1,1}, \dots, s_{1,m}, s_{2,1}, \dots, s_{2,m}, \dots, s_{n,1}, \dots, s_{n,m}\}$ and $\{v_{1,1}, \dots, v_{1,m}, v_{2,1}, \dots, v_{2,m}, \dots, v_{n,1}, \dots, v_{n,m}\}$ respectively.

The solution should respect two constraints: -

- (i) each task should be assigned to one agent,
- (ii) agents capacities should not be exceeded .

GAP can be formulated as follows:-

$$\text{Minimize} \quad \sum_{i=1}^n \sum_{j=1}^m v_{ij} x_{ij}$$

$$\text{Subject To} \quad \sum_{j=1}^m x_{ij} = 1 \quad \forall i \in I = [1..n]$$

$$\sum_{i=1}^n s_{ij} x_{ij} \leq c_j \quad \forall j \in J = [1..m] \text{ where}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, \forall j \in J$$

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

The above model performs well for small networks, but having a NP Hard problem at its core gives exponential running costs for larger networks with thousands of nodes. So to deal with that the author proposes heuristics such as to partition the network into geographical zones and apply and solve the GAP for each region and combine the results to produce a single solution. As good as it sounds this method incurs a lot of additional overheads for geo partitioning the whole network , solving the GAP multiple times and finally to combine the smaller solutions to come up with a single solution. These overheads make this solution a little less suitable for dynamic networks where the data placement strategy might have to be re evaluated frequently. Now this where our solution comes to the picture.

IFogStorR: Intuition

We observe that the only difference between the GAP formulated in iFogStor and a common Assignment Problem is the constraint that every datahost should not be assigned data items more than its capacity, that is

$$\sum_{i=1}^n s_{ij} x_{ij} \leq c_j \quad \forall j \in J = [1..m]$$

If this constraint is removed, the problem is reduced to a simple Assignment problem solvable in polynomial time.

So, we propose a solution to this which is as follows:-

1. We calculate a constant for a given network size called Replication Factor which is the No.of Producers divided by the No.of Datahosts
2. This involves a reasonable assumption, that the No.of producers in a network are greater than or equal to the No.of Datahosts
3. We then replicate each datahost by a factor equal to the Replication Factor. As a result of this the No.of data hosts becomes equal to the No.of data producers.
4. The point to keep in mind over here is that ,the replication of datahosts is only logical or conceptual and not physical.
5. It can be thought of as if each datahost is comprised of a number of storage nodes equal to the Replication Factor.

With the following points in mind we are now ready to explore our solution in depth.

IFogStorR: Quantitative Problem Formulation

As described earlier we have modeled the data placement as an Assignment Problem.

Assignment Problem

The Assignment Problem is one of the fundamental problems 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 $\{c_1, c_2, \dots, c_m\}$ and each task has a different size and a different cost according to the agent it is assigned to. Sizes and costs are denoted $\{s_{1,1}, \dots, s_{1,m}, s_{2,1}, \dots, s_{2,m}, \dots, s_{n,1}, \dots, s_{n,m}\}$ and $\{v_{1,1}, \dots, v_{1,m}, v_{2,1}, \dots, v_{2,m}, \dots, v_{n,1}, \dots, v_{n,m}\}$ respectively.

The solution should respect two constraints: -

- (i) each task should be assigned to only one agent,
- (ii) each agent should be assigned atmost one task.

Then the Assignment problem can be formulated as follows:-

$$\text{Minimize} \quad \sum_{i=1}^n \sum_{j=1}^m v_{ij} x_{ij}$$

$$\text{Subject To} \quad \sum_{j=1}^m x_{ij} = 1 \quad \forall i \in I = [1..n]$$

$$\sum_{i=1}^n x_{ij} \leq 1 \quad \forall j \in J = [1..m] \text{ where}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, \forall j \in J$$

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

Preliminaries

1) Notation: The used notation is summarized in the following table:-

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

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.

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).

Data Placement Model

1) Problem formulation: Let S be a system composed of a set of *DataHost* $DH = \{dh_1, dh_2, \dots, dh_n\}$, a set of *DataProd* $DP = \{dp_1, dp_2, \dots, dp_l\}$ and a set of *DataCons* $DC = \{dc_1, dc_2, \dots, dc_m\}$. DP produces the data set D with $D = \{d_1, d_2, \dots, d_l\}$ 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 AD 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 $x_{i,j} = 0$.

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

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 .

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

Each d_i will be requested by one or 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.

$$PC_D = \begin{bmatrix} pc_{1,1} & \cdots & pc_{1,m} \\ \vdots & \ddots & \vdots \\ pc_{l,1} & \cdots & pc_{l,m} \end{bmatrix} \quad \text{where } pc_{i,k} \in \{0, 1\} \quad (3)$$

$$TR_D = \begin{bmatrix} tr_{D_{1,1}} & \cdots & tr_{D_{1,n}} \\ \vdots & \ddots & \vdots \\ tr_{D_{l,1}} & \cdots & tr_{D_{l,n}} \end{bmatrix} \quad (4)$$

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

$$\text{Overall Latency} = \sum_{ts_{d_{i,j}} \in TS_D} ts_{d_{i,j}} + \sum_{tr_{d_{k,j}} \in TR_D} tr_{d_{k,j}} \quad (5)$$

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

2) Latency model: In this subsection, we provide a model for the overall generated latency to evaluate the data placement defined by AD . 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:-

$$ts_{d_{i,j}} = \left[\frac{1}{b} s_{d_i} \right] ts_{i,j} \quad (9)$$

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: -

$$tr_{D_{k,j}} = \left[\frac{1}{b} S_{D_{k,j}} \right] tr_{k,j} \quad (10)$$

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

$$S_{D_{k,j}} = (s_{d_1} \cdot pc_{1,k}, \dots, s_{d_l} \cdot pc_{l,k}) \begin{pmatrix} x_{1,j} \\ \vdots \\ x_{l,j} \end{pmatrix} \quad (11)$$

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

$$tr_{D_{k,j}} = tr_{k,j} \left(\left[\frac{1}{b} s_{d_i} \right] \cdot pc_{1,k} \cdot x_{1,j} + \dots + \left[\frac{1}{b} s_{d_l} \right] \cdot pc_{l,k} \cdot x_{l,j} \right) \quad (12)$$

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

$$\text{Overall Latency} = \sum_{i \in [1..l]} \sum_{j \in [1..n]} \alpha_{i,j} \cdot x_{i,j} \quad (13)$$

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

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:

$$\left\{ \begin{array}{l} \text{Minimize} \quad \sum_{i \in [1..l]} \sum_{j \in [1..n]} \alpha_{i,j} \cdot x_{i,j} \\ \text{Subject to} \quad \sum_{j=1}^m x_{ij} = 1 \quad \forall i \in I = [1..n] \\ \sum_{i=1}^n x_{ij} \leq 1 \quad j \in J = [1..m] \\ x_{ij} \in \{0, 1\} \quad \forall i \in I, j \in J \end{array} \right.$$

Data Placement Problem Solving: We define the data placement as an Assignment Problem which is a special case of the Integer Programming Problem for which a solution can be found in polynomial time.

We have solved the GAP problem for an identical network using the **CBC Mixed Integer Programming Solver** provided by **Google OR Tools** and the Assignment Problem using a JAVA implementation of the **Hungarian Algorithm** .

IFogStorR: This solution comprises of solving the Assignment Problem in polynomial time using a Java implementation of the **Hungarian Algorithm**. It finds the optimal placement of the generated data on Fog storage nodes to optimize the overall latency. Since the assignment problem is always solvable in polynomial time, our solution scales very efficiently for larger networks. The polynomial runtime of the solution also makes it suitable for more dynamic networks where data placement is to be executed in runtime.

The optimality of the solution and its performance in a real network is investigated in the evaluation section by varying the number of fog nodes at each level of the network hierarchy.

APPLICATION AND EXPERIMENTAL EVALUATION

Smart city is a major application domain of IoT [6]. 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. . 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 a scaled down 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 5. 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 from 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.

Experimental Tools and Setup

We used *iFogSim* [2] 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 the *CBC MIP Solver* and the *Hungarian Solver* and solve the problems, and
- (iii) to recover the problem solution and set the data placement allocation.

The evaluation has been achieved using a laptop that has 2 CPU cores and 8 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 . We suppose that *data actors* transmit their data in packets [1]. In our experiments, we set the size of data packets generated by

sensors to 96 bytes and data packets generated by application instances to 960 bytes (i.e. a selectivity of 0.1). In our experimentations, we used 2 data-centers, 4 RPOPs and 8 LPOPs, and we varied the number of GWs within the range [56.....168] by steps of 56. For service composition and data sharing simulation, we varied the number of *DataCons* per *DataProd* (denoted *cp*) from 1 to 5.

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

Evaluation 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:-

- **Cloud Only:** all data items are stored on a central node located in the cloud layer of the hierarchy without considering about data sharing.

- **Nearest DataHost (Nearest dh):** data are stored in the closest *DataHost* to its *DataProd* in term of latency, a naive approach that does not consider data sharing.
- **iFogStor:** data are stored in the *DataHost* which minimizes the overall latency according to the solution of a GAP problem .
- **iFogStorR:** data are stored in the *DataHost* which minimizes the overall latency according to our strategy by solving the Assignment Problem. We used 4, 8 and 12 as the Replication Factor, denoted iFogStorR(4), iFogStorR(8), and iFogStorR(12) respectively. For each case, we investigated the impact of the Replication Factor on the overall latency and the gain in problem solving time as compared to iFogStor.

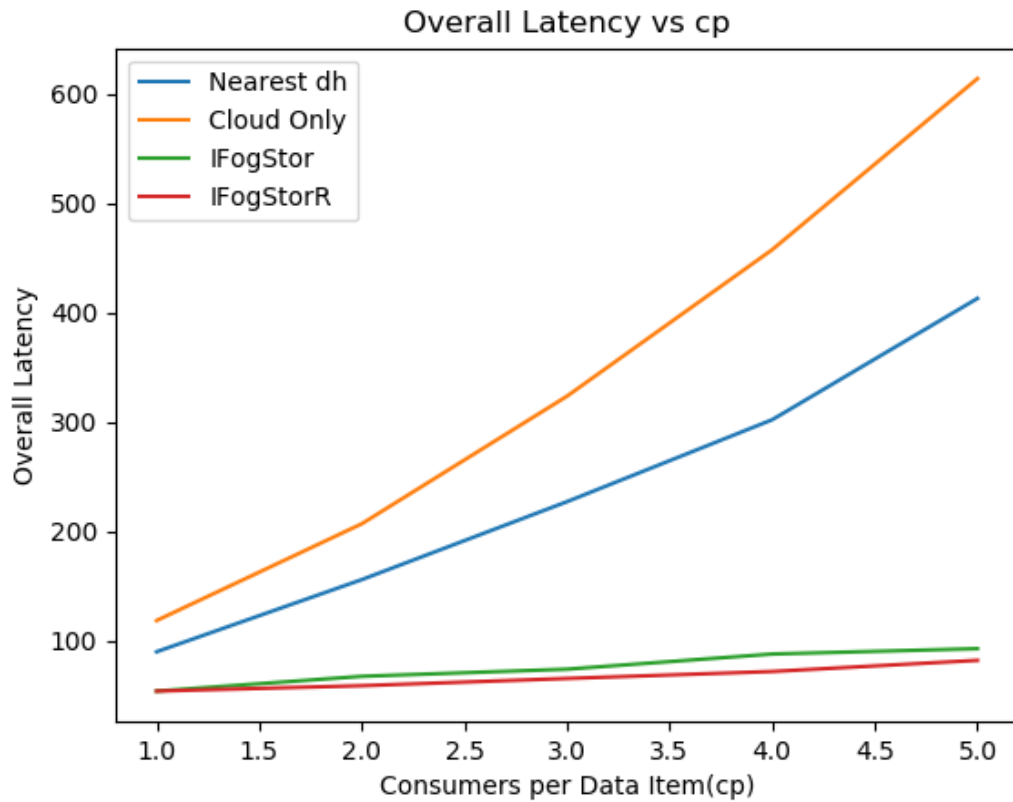
Results and Discussion

We have tabulated the results for both overall latency and the solution time. All graphs have been plotted using the Matplotlib library in python.

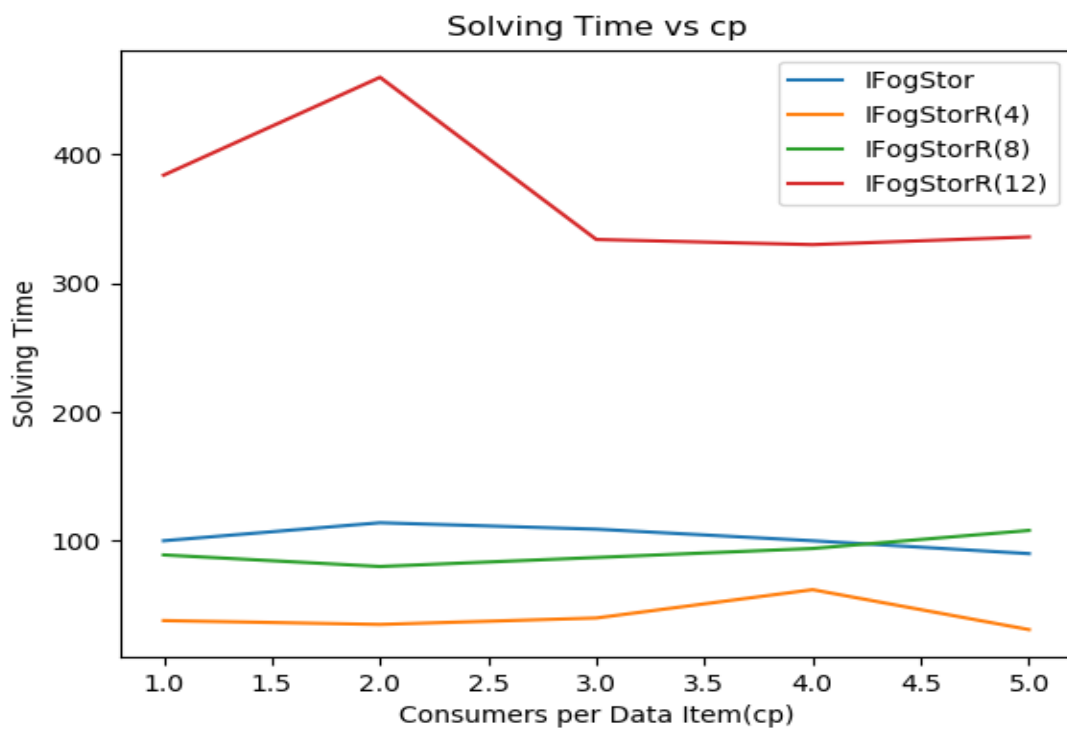
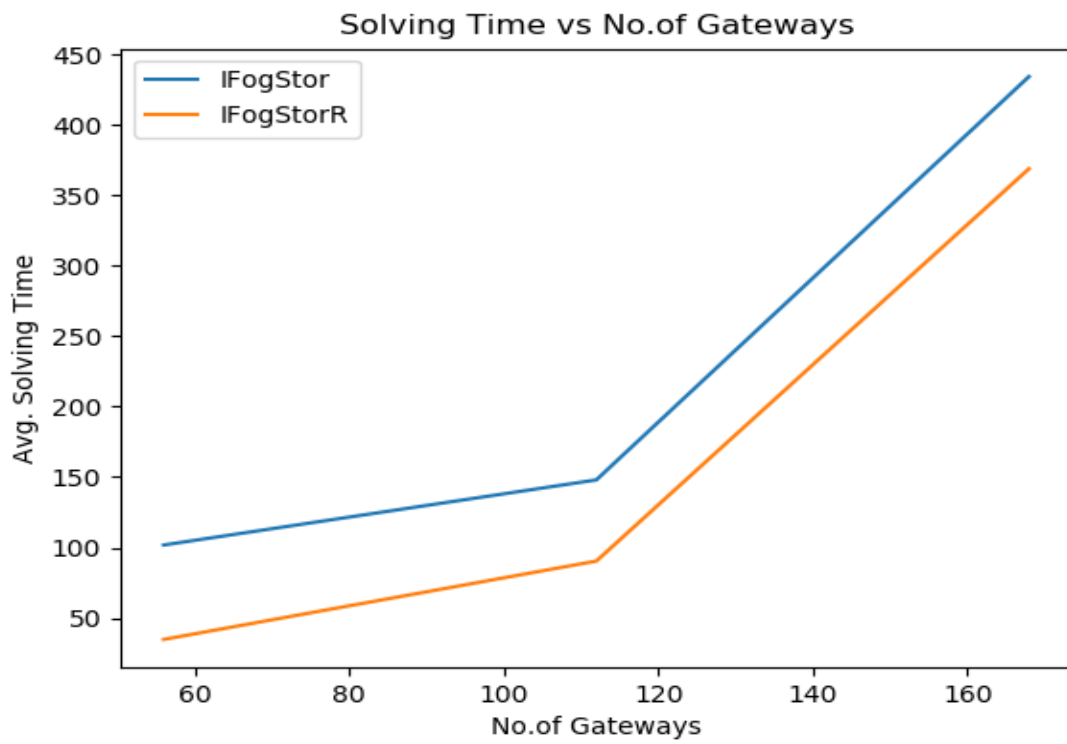
Overall Latency: As tabulated below, iFogStorR provides a remarkable amount of time saving as compared to other data placement strategies.

Storage Mode	Average Saving%
Nearest dh vs Cloud Only	30.92
iFogStor vs Nearest dh	68.29
iFogStorR vs Cloud Only	80.63
iFogStorR vs Nearest dh	71.96
iFogStorR vs iFogStor	11.58

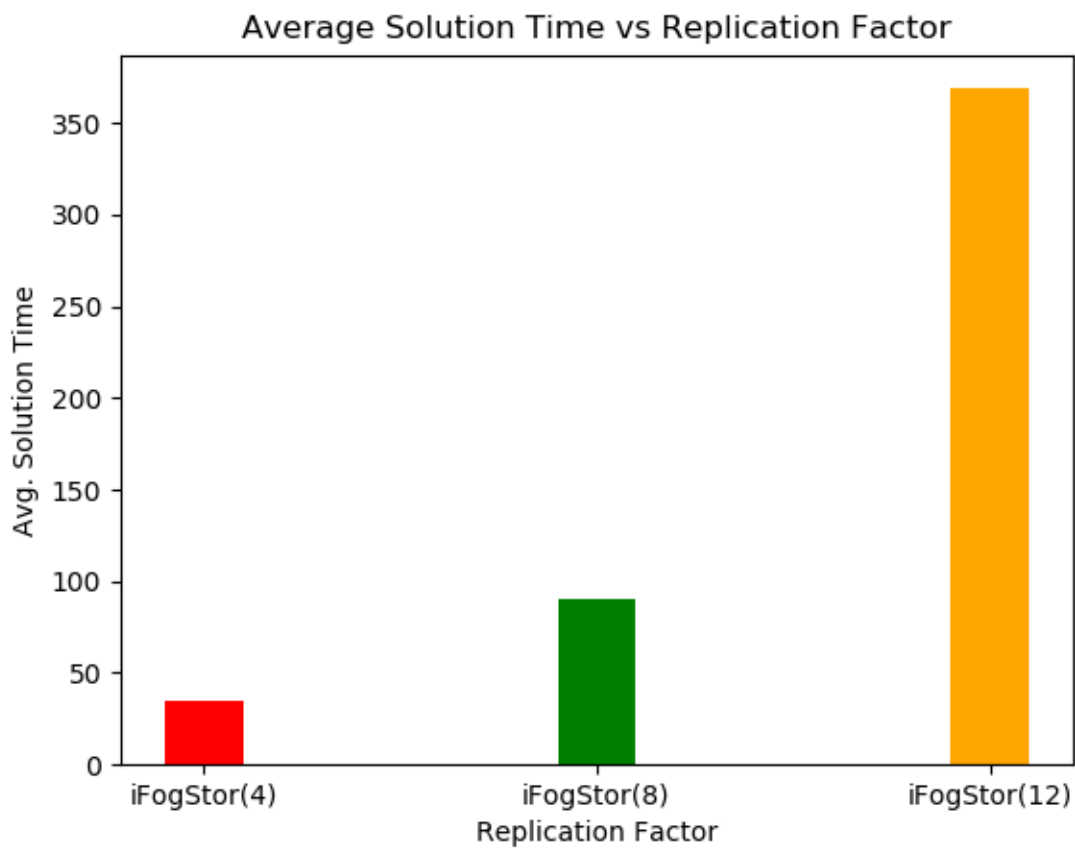
As shown in the graph between Overall Latency and the Consumers per Data Item(cp) , one can observe that the overall latency of using a Cloud storage is very high as compared to other modes. This is due to the distance between data consumers and data hosts. In addition, this generates a high data traffic while sending/retrieving data to/from the cloud. Also as the value of cp increases , all the strategies show a steady increase in the overall latency except iFogStor and iFogStorR. It is also clearly visible from the graph that the least latency is obtained through iFogStorR(saving over 70% time from the Nearest dh and 80% time from the Cloud only strategy) . Infact our strategy beats iFogStor by 11.58% in terms of time saved.



Solving Time: A graph between Average Solution Time and the no. of gateway devices also roughly confirms our idea about the differences between the GAP and the Assignment problem as the former always takes more time to be solved as the no. of gateway devices is varied from 56 to 168. However, the solution time seems to be independent of the cp value for all versions of iFogStorR. This is inferred from the graph between solution time and cp as more or less all of the curves do not show any trending behavior as the cp value is varied from 1 to 5.



A comparison was made to study the effect of the Replication Factor on the problem solving time. The bar graph shows that the average solving time increases with increase in the Replication Factor. This is only normal because an increase in the replication factor causes the size of the corresponding assignment problem to increase , thereby requiring more solving time. However the increase was found to be somewhat drastic for change in the replication factor from 8 to 12.



CONCLUSION AND FUTURE WORK

This paper presents iFogStorR, a runtime IoT data placement strategy for latency reduction in a Fog architecture. iFogStorR 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 an Assignment Problem,
- 2) we proposed an exact polynomial time solution to the problem based on the Hungarian Algorithm,
- 3) we provided a framework for testing our strategies based on *iFogSim*.

iFogStorR has shown promising results as it largely enhanced Cloud and naive Fog data placement strategies from a latency point of view. In addition, the Assignment Problem developed gave very good results while drastically reducing solving time even for very large problems.

For future works, this strategy has to be implemented on much larger networks comprising of a few thousand nodes to get a clearer idea about its performance. Also more refined data placement models have to be considered that take into account other factors such as energy consumption in the nodes, the total network utilization, etc.

REFERENCES

- Mohammed Islam Naas, Philippe Raipin Parvedy, Jalil Boukhobza, Laurent Lemarchand iFogStor: an IoT Data Placement Strategy for Fog Infrastructure. 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)
- 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.
- 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.
- [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.
- 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.
- A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya. Fog computing: Principals, architectures, and applications. 2016.

I also referenced the Google OR Tools page on Optimization problems extensively to be found at <https://developers.google.com/optimization/introduction/overview>. and used a Java implementation of the Hungarian Algorithm licensed by MIT to be found at the following link <https://github.com/aalmi/HungarianAlgorithm>

All of my source code can be accessed from <https://github.com/sourjyar7/IFogStorR>.

