A Graph Partitioning-based Heuristic for Runtime IoT Data Placement Strategies in a Fog infrastructure

Mohammed Islam NAAS
Orange
Rennes, France
mohammedislam.naas@orange.com

Laurent Lemarchand
Univ. Bretagne Occidentale UMR 6285, Lab-STICC F-29200
Brest, France
laurent.lemarchand@univ-brest.fr

Jalil Boukhobza
Univ. Bretagne Occidentale UMR 6285, Lab-STICC F-29200
Brest, France
jalil.boukhobza@univ-brest.fr

Philippe Raipin
Orange
Rennes, France
philippe.raipin@orange.com

ABSTRACT

Fog computing is a dense, heterogeneous and geographically distributed infrastructure. With the rise of IoT applications, objects may generate large amounts of data that may be processed at different locations of the Fog infrastructure. Data placement strategies have been designed to investigate the best storage location for data in order to reduce its access time for different IoT services spread over the infrastructure. Unfortunately, due to the large number of Fog nodes and the amount of data to be managed, placing data in such infrastructure is an NP-Hard problem. In this paper, we propose a divide and conquer heuristic for data placement strategies in Fog infrastructures. Our idea consists in dividing the original data placement problem into several balanced sub-problems using graph modeling and partitioning methods. Using our heuristic makes it possible to reduce the solving time by more than 450 times with less than 5% of optimality loss as compared to the exact solution (without subdivision). For a given number of partitions, our solution proved to be at least as close to the optimal as state-of-the-art solutions and 30% closer to the optimal for many workloads. In addition, our solution allowed for a better optimization in solving time as it is more flexible and scalable in terms of number of partitions.

CCS CONCEPTS

• **Information systems** → Distributed storage; • **Mathematics of computing** → Graph algorithms; • **Computer systems organization** → Peer-to-peer architectures;

KEYWORDS

IoT, Fog, Data Placement, Storage, Optimization, Generalized Assignment Problem, Graph Partitioning.

SAC 2018, April 9–13, 2018, Pau, France
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167217

ACM Reference Format:
Mohammed Islam NAAS, Laurent Lemarchand, Jalil Boukhobza, and Philippe Raipin. 2018. A Graph Partitioning-based Heuristic for Runtime IoT Data Placement Strategies in a Fog infrastructure . In *SAC 2018: SAC 2018: Symposium on Applied Computing , April 9–13, 2018, Pau, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3167132.3167217

1 INTRODUCTION

In the last decade, Internet of Things (IoT) has made a great leap and is continuously attracting more attention in both industry and academia. By 2020, the number of connected objects will exceed 50 billion as predicted by Cisco [9]. These smart objects will generate a huge amount of data; supposedly 27% of the worldwide data will be produced by mobile and connected devices as expected by the IDC [16].

IoT objects are geographically distributed. In order to reduce the processing latency, managing the generated data requires to be performed as close as possible to IoT objects. This makes it possible to reduce network traffic and service latency. In fact, this was the main motivation behind the Fog computing paradigm introduced by Cisco [15]. The idea consists to extend IoT services traditionally provided by the Cloud down to the network edge.

Fog components, also called Fog nodes, are heterogeneous in terms of processing capabilities, storage capacities and are geographically distributed [17, 18]. Indeed, Fog nodes with limited resources (e.g. set-top boxes) are located near to IoT devices, whereas resources-rich ones (e.g. Points of Presence) are located far from IoT devices. This heterogeneity imposes to investigate how data should be placed (stored) in order to minimize the service time in the whole infrastructure [18].

Fog infrastructures are also dynamic and their network topologies change over time. This is due, for instance, to smart objects mobility or IoT service migrations. Accordingly, data placement and migration strategies should be designed to be executed periodically and in runtime [13]. However, investigating the best placement strategy for such large infrastructures containing a high number of objects and Fog nodes, and supporting many IoT services may lead to a combinatorial explosion (in terms of processing time and memory space). This makes such a data placement strategy unfeasible for a runtime execution. One way to solve this issue is to use heuristics based-on divide and conquer approaches [13].

In this paper, we propose a heuristic method to reduce the complexity of data placement strategies in Fog infrastructures. The objective of such data placement strategies is to minimize the data transfer latency, that is the time to copy data from their source to their storage node, and then send them to the consumer node. Our heuristic is based on a divide and conquer approach in which the Fog infrastructure is subdivided into several parts and the data placement strategy can be ran separately and in parallel on each part (constituting a sub-problem). Exact solutions of sub-problems are aggregated to make a global near optimal solution. Additionally, our approach makes it possible to run data placement strategies only on a subpart of the infrastructure.

In this context, problem subdivision is a critical issue. One needs to ensure that the generated infrastructure subparts are as **balanced** and as **disconnected** as possible. Balance needs to be achieved in order to have equivalent subproblem sizes to minimize the overall solving time. To do so, we took into account data volume and fog nodes number in the division process. The infrastructure subparts need to be as disconnected as possible in order to avoid having nodes exchanging a lot of data located in separate subparts. We have considered for this sake the dependency between nodes in terms of data flows between data producers and data consumers in order to build disconnected parts.

Partitioning a Fog infrastructure to reduce the overall complexity of data placement strategy has been investigated in iFogStor [13]. The authors have formulated the data placement problem as a Generalized Assignment Problem (GAP) [5] using linear programming. They faced a combinatorial explosion when the Fog infrastructure was too large. As a consequence, they have proposed *iFogStorZ*, a technique that divides Fog infrastructure into several geographical zones according to the location of Regional PoPs[1]. In fact, even though this method is very simple to implement, it may lead to a significant loss of optimality in case of distributed data flows (consumers far from producers) within the infrastructure, as it can be the case for smart city application. In addition, the subdivision process is limited to the number of available Regional PoPs.

We have upgraded *iFogSim* [11] with our subdivision heuristic and compared it with *iFogStorZ* using three smart city scenarios having more than 10,000 Fog nodes. The achieved experiments have shown that our approach presents at least as good performance as *iFogStorZ* in the case where the generated data are consumed by neighbor nodes, and it outperformed it for the other scenarios on average by 30% (closer to the optimal solution). Contrary to *iFogStorZ*, our approach has shown a better scalability as the partitioning is not limited by the number of Regional PoPs.

The rest of this paper is organized as follows. Section II presents some background knowledge and discusses related work. Section III describes our solution. Section IV presents the evaluation of our heuristic, and Section V concludes this paper.

2 BACKGROUND AND RELATED WORK

In this section, we first summarize iFogStor [13] data placement solution. In fact, this solution was used for the data placement part of our solution. Then, we describe some related work.

[1]A Regional PoP covers a geographical zone in ISP infrastructures.

2.1 iFogStor

iFogStor was proposed to tackle the data placement issue in the context of Fog computing and IoT.

The data placement issue was formulated as a Generalized Assignment Problem (GAP) [5] and solved using a linear programming approach. The authors considered a system infrastructure that contains a set of storage nodes (Fog nodes and data centers) denoted by $SN = \{sn_1, ..., sn_n\}$, and an amount of IoT data denoted by $D = \{d_1, ..., d_m\}$ which are generated and consumed by a set of IoT services. An objective function has been defined with the aim to place D in SN while minimizing the transfer latency (transfer of D from producers to storage nodes, and then from storage nodes to consumers). The solution should respect two constraints: (i) the capacity of storage nodes should not be exceeded, (ii) all data items should be stored. *iFogStor* formulation was described by the following linear system.

$$\begin{cases} Min & \sum_{i\in[1..m]} \sum_{j\in[1..n]} \alpha_{i,j}.x_{i,j} & \\ Subject & & \\ & \sum_{i\in[1..m]} S(d_i).x_{i,j} \le F(sn_j) & \forall j \in J = [1..n] \\ & \sum_{j\in[1..n]} x_{i,j} = 1 & \forall i \in I = [1..m] \\ & x_{i,j} \in \{0,1\} & \forall i \in I, \forall j \in J \end{cases}$$

With:

- $\alpha_{i,j}$ being the generated latency by storing d_i in sn_j
- $x_{i,j} = 1$ if d_i is stored in sn_j, and 0 otherwise
- $S(d_i)$ returns the size (in bytes) of d_i
- $F(sn_j)$ returns the free storage capacity (in bytes) of sn_j.

Authors used *CPLEX MILP* [1] to solve this problem.

2.2 Related Work

To the best of our knowledge, there is a single study in the literature that deals with reducing the complexity of data placement strategies in a Fog context, it is called *iFogStorZ* [13]. The solution is based on a divide and conquer approach to reduce the compute complexity of *iFogStor*. It consists in subdividing the system infrastructure into several geographical zones. In each zone, a data placement sub-problem is modeled and solved separately hence reducing the overall computation time. However, this method does not take into consideration the existing inter-zones *data flows* and this can induce a loss in optimality for the global solution. Moreover, geographical zones can be heterogeneous in terms of number of Fog nodes and IoT services. Accordingly, this partitioning method may result in an unbalanced sub-problems' data placement complexity. In this work, we compare our solution to *iFogStorZ* in the evaluation part.

Many research efforts were done to solve placement issues notably using graph partitioning methods. For instance, in [10], authors have proposed a solution to place data around a cluster of servers in order to minimize inter-tasks communication cost. They modeled tasks and data as a bipartite graph which is partitioned into several parts keeping together tasks and the required data in the same partitions. Linquan et al.[19] have proposed a solution to place Virtual machines (VMs) in a modular data center which involves several computing Pods hence minimizing inter-VMs communication cost. Many others studies have been proposed using the graph partitioning theory to solve placement issues [4, 6, 7, 14].

Using graph partitioning for solving data placement issues is not novel, however, to the best of our knowledge, there is no work seeking IoT and Fog infrastructure related applications which objectives are different from the aforementioned studies and will be detailed in the next sections.

3 IFOGSTORG: A HEURISTIC TO REDUCE DATA PLACEMENT STRATEGIES COMPLEXITY

In this section we first give an overview of our contribution and then detail each step of the proposed solution.

3.1 Heuristic overview

We consider a system architecture that consists of a set of Fog nodes and a set of IoT services. We refer to IoT services that produce data as *data producers* and the ones that consume data as *data consumers* (one IoT service can be both). IoT data can be stored in different locations of the Fog architecture. We refer to the Fog nodes that may store data as *data hosts*.

In our approach, we assume that IoT services are already placed in the infrastructure and the system has knowledge about: (i) IoT services location, (ii) the existing *data flows* between *data producers* and *data consumers*, (iii) network latencies between Fog nodes, and (iv) *data hosts* location.

Our objective is to reduce the complexity of data placement process in this infrastructure and make it feasible at runtime so that the system can adapt to the dynamic nature of the Fog infrastructure. For this purpose, we followed a divide and conquer approach. Our idea consists in subdividing the infrastructure into several parts, thus several sub-problems. Then, a data placement strategy can be ran separately on each part. Finally, those solutions are aggregated to make a global solution.

Balanced subparts: as the complexity of data placement depends on the number of data items (to be stored) and the existing *data hosts*, we ensured that their number was balanced over the subparts in the division process.

Disconnected subparts: moreover, the efficiency of a data placement strategy depends on considering the relationship between *data producers* and the respective *data consumers* locations. For this sake, in our division process, *data producers* and associated *data consumers* are maintained as much as possible in the same subparts (when this is not possible, data remain near to the source).

We employed a graph partitioning approach in which vertex weights are used to balance data items and *data hosts* among subparts, whereas edge weights are used to minimize inter-parts *data flows* (for placement efficiency). Our heuristic method followed four steps as shown in figure 1:

1. *Infrastructure modeling:* we model the physical infrastructure as an undirected graph. As shown in figure 1 (a) and (b), vertices represent Fog nodes and edges represent existing physical links. The obtained graph is to be weighted for both vertices and edges.
2. *Graph weighting:* vertex weights are allocated by counting the number of data items to be stored. Edges' weights are allocated by counting the number of *data flows* passing through these links.
3. *Graph partitioning:* once the graph has been weighted, we apply a *k-way partitioning* method to divide it into k sub-graphs using *Metis* [2]. The partition process respects two criteria: (i) the sum of vertex weights should be balanced among sub-graphs, and (ii) the sum of cut edges' weights should be minimized.
4. *Data placement solving:* once the system infrastructure is partitioned, for each partition, a data placement problem is modeled and solved separately.

In this work, we used the notations summarized in table 1.

Table 1: Notation dictionary

Notation	Description
S	The overall system
fn	Fog node
dh	DataHost, Fog node that can host data
dp	DataProd, IoT service that can produce data
dc	DataCons, IoT service that can use data
FN	Fog nodes set
DH	DataHost set
DP	DataProd set
DC	DataCons set
f	Data flow, dependency between dp and dc
DFM	Matrix containing existing data flows
ADM	Adjacency Matrix containing latencies of existing physical links between nodes
DPM	Matrix mapping Fog nodes to executed dp
DCM	Matrix mapping Fog nodes to executed dc
G	Graph
v	Vertex
e	Edge
V	Vertices set
E	Edges set
P	Graph partition set
C	Cut edges set

3.2 Infrastructure modeling

Our modeling approach follows two steps. The first one consists in a system modeling of the Fog infrastructure with the executed services. The second step consists in building the graph that will be used in the rest of the solution.

3.2.1 System modeling. Considering a system S composed of a set of Fog nodes $FN = \{fn_1, fn_2, ..., fn_n\}$, a set of IoT services which are considered as *data producers* $DP = \{dp_1, dp_2, ..., dp_l\}$, and a set of IoT services which are considered as *data consumers*

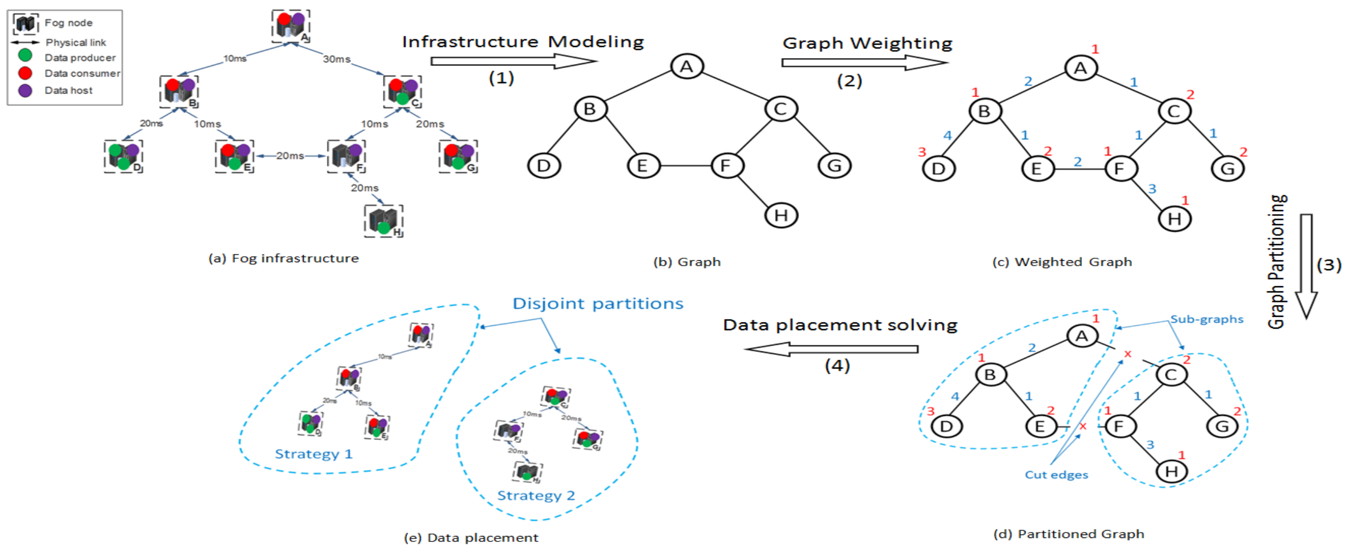


Figure 1: Heuristic steps.

$DC = \{dc_1, dc_2, ..., dc_m\}$. Let $DH = \{dh_1, dh_2, ..., dh_k\}$, $DH \subset FN$, be the subset of Fog nodes which are considered as *data hosts*.

Let *DFM* be the matrix of *data flows* that links each *data producer* with the related *data consumers* set. A given element $f_{i,j} = 1$ means that dc_j requests data from dp_i.

$$DFM = \begin{bmatrix} f_{1,1} & \cdots & f_{1,m} \\ \vdots & \ddots & \vdots \\ f_{l,1} & \cdots & f_{l,m} \end{bmatrix}, f_{i,j} \in \{0, 1\} \qquad (1)$$

DPM is the matrix that maps Fog nodes with their executed *data producers* (IoT services). A given element $p_{g,i} = 1$ means that fn_g implements dp_i.

$$DPM = \begin{bmatrix} p_{1,1} & \cdots & p_{1,l} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,l} \end{bmatrix}, p_{g,i} \in \{0, 1\} \qquad (2)$$

DCM is the matrix that maps Fog nodes with the implemented *data consumers*. A given element $c_{g,j} = 1$ means that fn_g implements dc_j.

$$DCM = \begin{bmatrix} c_{1,1} & \cdots & c_{1,m} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,m} \end{bmatrix}, c_{g,j} \in \{0, 1\} \qquad (3)$$

ADM is the adjacency matrix of Fog nodes. $t_{g,u}$ represents the existing latency (in milliseconds) of the physical link located between fn_g and fn_u. If $t_{g,u} = \infty$ means that there is no physical link between fn_g and fn_u.

$$ADM = \begin{bmatrix} t_{1,1} & \cdots & t_{1,n} \\ \vdots & \ddots & \vdots \\ t_{n,1} & \cdots & t_{n,n} \end{bmatrix}, t_{g,u} \in \{\Re^+\} \qquad (4)$$

3.2.2 *Graph building.* The infrastructure of the system S can be modeled as an undirected graph $G = (V, E)$, as shown in figure 1 (a) and (b). $V = \{v_1, ..., v_n\}$ is the set of vertices which represent Fog nodes, and $E = \{e_1, ..., e_r\}$ is the set of edges which represent the existing physical links. As aforementioned, the graph G is weighted for both vertices and edges.

3.3 Graph weighting

The weighting process is made following the next two steps.

3.3.1 *Vertex weighting.* Vertex weights are used to balance the complexity of the data placement problem which depends on both the number of data items and storage nodes in the studied system. Thus, our idea is to weight each vertex with the value corresponding the number of data items plus one if the node is considered as a *data hosts*. We assume that each *data producer* generates a data item per period of time, thus the number of data items equals the number of *data producers*. For example, in figure 1 (a), the Fog node D executes two *data producers* and is considered as a *data host*. Thus, the weight of its related vertex is 3 (see figure 1 (c)).

3.3.2 *Edge weighting.* As previously mentioned, inter-parts *data flows* may lead to a loss in placement optimality. Thus, our idea is to weight each edge e existing in E by the number of *data flows* passing through the related physical link. In order to count this number of *data flows*, we assume that data are always transmitted on shortest paths between *data producers* and *data consumers*. In order to calculate the shortest paths, we used *Floyd*'s algorithm [8], which takes the adjacency matrix *ADM* as input.

As a consequence, by minimizing the sum of the weights of the cut edges, the inter-parts *data flows* are minimized.

For example, we assume that the *data producer* executed in the Fog node H, in figure 1 (b), has three *data flows*: ($f_1 = (H \rightarrow B)$, $f_2 = (H \rightarrow C)$, $f_3 = (H \rightarrow E)$). Thus, the physical link (H, F) is used as shortest path for three *data flows*. Then, the allocated weight for the concerned edge is 3. Edge weighting pseudo code is described in Algorithm 1.

3.4 Graph Partitioning

In this step, we seek to partition the resulting weighted graph into several sub-graphs respecting two criteria: (i) vertex weights should be balanced over the set of sub-graphs, and (ii) the sum of cut edge weights should be minimized. We formulate this problem as follows.

Let $P = \{P_1, ..., P_k\}$ be the set of sub-graphs and $C = \{e_1, ..., e_w\}$, $C \in E$, be the set of cut edges resulting from partitioning step of the graph G.

Consider $Weight(P_z)$ and $Weight(C)$ be the functions that return the sum of vertex weights of P_z and the sum of edge weights of C respectively. The partition problem formulation is described as follows [12]:

$$\begin{cases} Minimize & Weight(C) \\ & \\ Subject\ to & \\ & Weight(P_1) \cong \ldots \cong Weight(P_k) \\ & \\ & \bigcup\limits_{1 \le i \le k} P_i = V \\ & \\ & P_i \cap P_j = \emptyset, i \neq j, i\ and\ j \in [1, k] \end{cases}$$

We used *Metis* to solve the partition problem by applying a *k-way partitioning* method. This method subdivides a weighted graph into k non-null sub-graphs while balancing vertex weights and minimizing the sum of cut edge weights.

3.5 Data placement solving

Once the previous step is done, our heuristic subdivides the infrastructure into k partitions according to the graph partitioning result, see figure 1 (e). In each partition, a data placement sub-problem is formulated and solved separately using iFogStor [13]. Different data placement algorithms may also be used. Then, the global solution is formed by aggregating sub-problems' solutions.

Increasing the number of partitions k helps reducing sub-problems size and therefore decreases the complexity of data placement strategies. However, it may lead to a loss of optimality induced by generating more inter-part *data flows* (as more edges are cut). The trade-off between the complexity reduction and optimality loss of data placement strategies is investigated in the evaluation section.

Algorithm 1: Edge Weighting

```
Input: $G$, $DFM$,$DPM$,$DCM$,$ADM$
Output: $EdgeWeight$
1 function EdgeWeightCompute()
2     forall $e \in E(G)$ do
3         $EdgeWeight(e) \leftarrow 0$                // initialization
4         forall $f \in DFM$ do
5             $Path \leftarrow shortestPath(f, DPM, DCM, ADM)$
6             if $e \in Path$ then
7                 $EdgeWeight(e) \leftarrow EdgeWeight(e) + 1$
8     return $EdgeWeight$
```

4 EVALUATION

In this section, we first describe the case study and the experimental setup we used. Then, we detail our methodology to evaluate our data placement heuristic. We finish by discussing simulation results.

4.1 The case study

To apply our heuristic method, we used the generic use-case of smart city proposed in [13], where several types of sensors (humidity, temperature and luminosity) collect environmental data and send them to a variety of IoT services for processing and analytical sake. These IoT services are implemented within an infrastructure encompassing Fog nodes and Cloud data-centers. As shown in figure 2, Fog nodes consist of gateways (GW), Local PoPs (LPOP) and Regional PoPs (RPOP), which are arranged hierarchically. In this use-case, IoT services may (i) require data from other IoT services, and (ii) share data between them. We used 3 different types of workloads:

(1) **Zoning workload**: here, data are used by consumers located in the same geographical zone as the producers. A zone is defined as a geographical space covered by at least one RPOP.
(2) **Distributed workload**: in this case, data are consumed by randomly selecting IoT services from the whole infrastructure.
(3) **Mixed workload**: for this last workload, half of the data are used by consumers from the same zone, and the other half by randomly selected consumers.

4.2 Experimental Tools and Setup

In this work, we used (i) *iFogSim* to simulate the system behavior, apply our data placement heuristic and measure the generated latency, (ii) *Metis* as a graph partition tool, and (iii) *iFogStor* as a data placement strategy.

As shown in figure 3. First, *iFogSim* starts and generates the system infrastructure (creation of sensors, Fog nodes, datacenters, IoT services and *data flows*). Then, it models the generated infrastructure as a graph which is to be weighted in the next step. Once the graph is weighted, *iFogSim* stores it in a *Metis* compliant format. Once *Metis* receives the graph, it partitions it into k sub graphs using *k-way partitioning* method, and returns the result to *iFogSim*. This

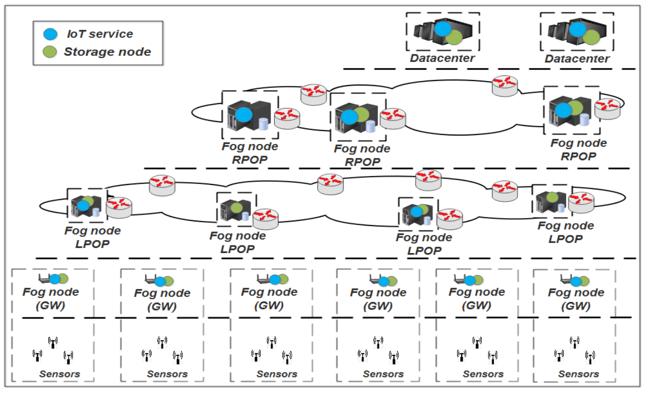


Figure 2: Use-case infrastructure.

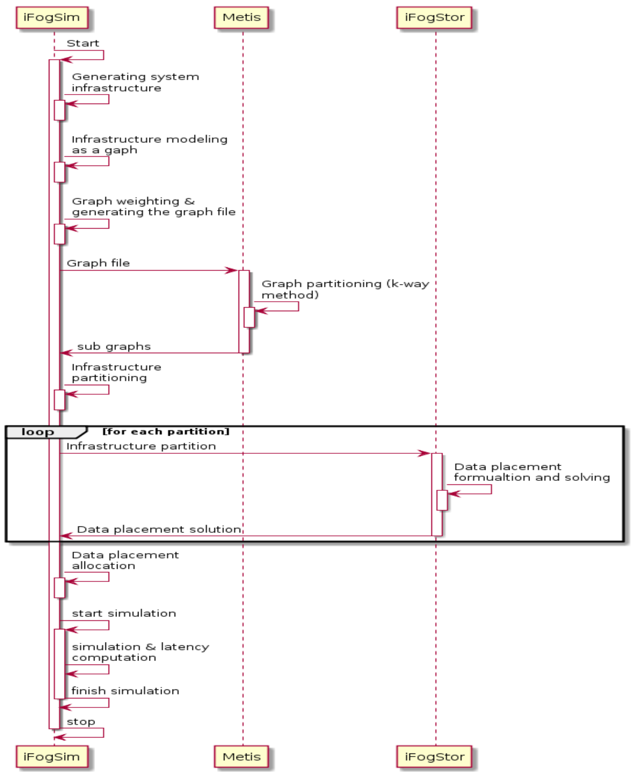

iFogSim
Metis
iFogStor
Start
Generating system infrastructure
Infrastructure modeling as a gaph
Graph weighting & generating the graph file
Graph file
Graph partitioning (k-way method)
sub graphs
Infrastructure partitioning
loop
[for each partition]
Infrastructure partition
Data placement formualtion and solving
Data placement solution
Data placement allocation
start simulation
simulation & latency computation
finish simulation
stop
iFogSim
Metis
iFogStor

Figure 3: Simulation sequence

Table 2: 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 3: Free storage capacities

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

latter builds the infrastructure partitions according to *Metis* results, and then for each part of the infrastructure, it applies an *iFogStor* instance to formulate and solve the data placement subproblem in parallel.

Once subproblems are solved, *iFogSim* retrieves solutions, sets the data placement allocation and performs the simulation to measure the latency.

The simulated system infrastructure consists of 5 datacenters, 10 RPOPs, 100 LPOPs and 10,000 GWs [3]. We have fixed 100 sensors per GW. Table 2 shows the considered network latencies between nodes, and table 3 illustrates available storage capacity on each type of *data host* [13].

In our experiments, data exchange is achieved per packet. We set the size of data packets generated by sensors to 96 bytes and data packets generated by IoT services to 960 bytes [13]. We generated 3,41 GB of IoT data during the simulation. For services data sharing, we varied the number of consumers that share data (denoted by *cp*) from 1 to 5, as proposed in [13]. The evaluation has been achieved using a machine having 48 CPU cores and 348 GB of RAM.

4.3 Evaluation Methodology

To evaluate our heuristic, we considered two metrics: (1) the quality of data placement which is measured by the sum of all the generated latencies related to data movement (storage and access) over a given period of time, and (2) the required time to compute the overall data placement which involves the time for data placement problem formulation and solving, and infrastructure partitioning. We compared these metrics using the next data placement strategies.

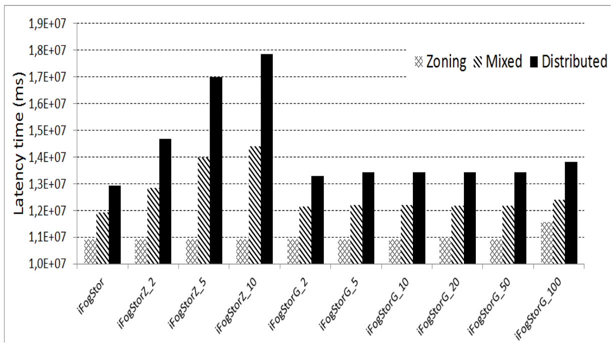
(1) **iFogStor** [13]: this strategy is an exact algorithm that finds the optimal solution to place data. Using this strategy, all data items are stored in *data hosts* which minimize the overall latency.
(2) **iFogStorZ** [13]: this is a heuristic strategy, it divides the original problem geographically into several sub-problems, and for each one, it calls *iFogStor* to find the exact solution to place data.
(3) **iFogStorG** (our approach): we divided the system infrastructure into *k* parts following our method of graph modeling and partitioning, and then for each partition, we used *iFogStor* to place data. For this latter, we gave *iFogStorG* as a name of our heuristic.

In order to compare *iFogStorG* (our heuristic) with *iFogStorZ*, in our experiments, we used the same number of partitions $k =$ 2, 5 and 10, as proposed in [13]. To test the scalability (defined as the efficiency with a high number of partitions) of *iFogStorG*, we also evaluated scenarios with 20, 50 and 100 partitions. These configurations were not tested with *iFogStorZ* as the maximum number of possible zones equals the number of RPoPs (limited to 10 in our configuration as in [13]).

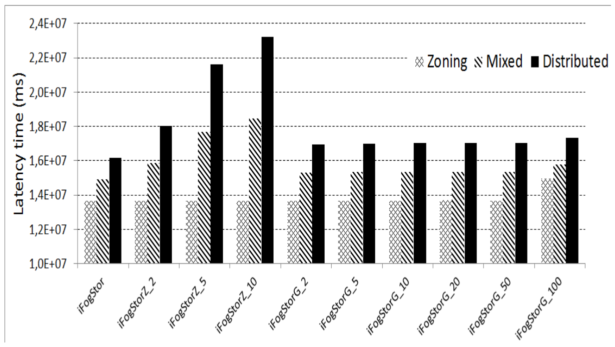
4.4 Results and Discussion

4.4.1 The quality of data placement: Overall latency. Graphics in figure 4 (a),(b) and (c) show the generated latency for each data placement strategy with different workloads for various *cp* (number of IoT services that share the same data). One can observe that overall, the trend is the same with an additional overhead for larger values of *cp*. This is related to the additional data transfer time.

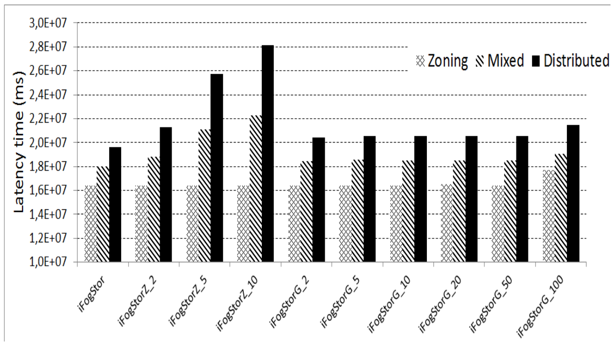
First, when using the **Zoning** workload, which is supposed to be suitable for *iFogStorZ*, we observe that both *iFogStorZ* and *iFogStorG* have approximately the same performance as *iFogStor*. This is the case regardless of the number of zones. As shown in table 4, the placement optimality is close to 100% for both strategies. *iFogStorG* proves to be scalable as it keeps a good level of optimality when



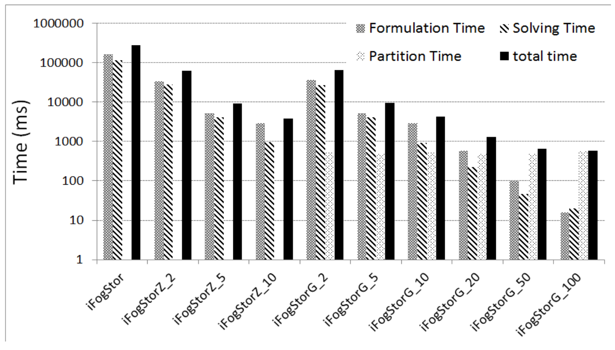
(a) Overall latency for cp =3.



(b) Overall latency for cp =4.



(c) Overall latency for cp =5.



(d) Time stats.

Figure 4: Simulation results.

Table 4: Data placement optimality

Strategy	Zoning	Mixed	Distributed
iFogStorZ_2	99.99%	95.72%	92.38%
iFogStorZ_5	99.99%	85.72%	76.31%
iFogStorZ_10	99.99%	80.82%	69.72%
iFogStorG_2	99.97%	97.64%	96.12%
iFogStorG_5	99.99%	97.17%	95.49%
iFogStorG_10	99.99%	97.30%	95.49%
iFogStorG_20	99.22%	97.29%	95.50%
iFogStorG_50	99.92%	97.43%	95.50%
iFogStorG_100	92.83%	97.54%	91.38%

using less than 50 partitions, and in the worst case (100 partitions), it still performs well (92.83%).

Second, using the **Mixed** workload, we observe that latencies are increased for all strategies. This is due to the additional data transfer delay when data are requested by consumers located at different geographical zones as producers. We observe also that when increasing the number of zones, *iFogStorZ* generates higher latencies and loses significantly in optimality. For instance, the placement optimality drops from 95% when using 2 zones to 80% when using 10 zones. Conversely, *iFogStorG* keeps the optimality close to 97% and presents a good placement quality, regardless of the number of zones (2, 5 and 10). Moreover, it shows a good scalability as the optimality remains above 97% for 100 zones.

Finally, when using the **Distributed** workload, we observe that with the increase of the number of zones, *iFogStorZ* generates higher latencies and loses significantly in optimality. As shown in table 4, using *iFogStorZ*, the data placement optimality is decreased from 92% with 2 zones, down to 69% when using 10 zones. Conversely, *iFogStorG* presents a good placement quality regardless of the number of zones (2, 5 and 10 zones) and gives solutions close to the exact solution (95% of optimality). In the worst case, the optimality decreases to 91% using 100 zones.

To summarize, *iFogStorG* presents good placement quality and keeps same performances whatever the type of used workloads, whereas *iFogStorZ* behaves poorly in both **Mixed** and **Distributed** workloads and shows a quality degradation when increasing the number of partitions. Moreover, *iFogStorG* can scale up to a 100 zones while providing a good data placement optimality, whereas *iFogStorZ* is limited to the number of geographical zones.

4.4.2 Data placement computation time. In this subsection, we compare the required time to compute the data placement solution for each strategy. As previously mentioned, this involves the time for formulating and solving the data placement problem, in addition to the partitioning time in the case of our strategy *iFogStorG*.

Figure 4 (b) shows the time duration of the different phases for each strategy. We can observe that both formulation and solving

Table 5: Heuristics speedup

Strategy	iFogStorZ_2	iFogStorZ_5	iFogStorZ_10
Speedup	4.83	35.46	81.22
Strategy	iFogStorG_2	iFogStorG_5	iFogStorG_10
Speedup	5.09	32.96	90.26
Strategy	iFogStorG_20	iFogStorG_50	iFogStorG_100
Speedup	223.17	442.70	473.55

time are very high when using the exact method *iFogStor* for all the used workloads. Increasing the number of zones drastically reduces these times. This is due to the complexity reduction of the data placement problem which is NP-Hard. We can also observe that the partitioning time is close to zero (less to one second) compared to the formulation and solving time. Note that *iFogStorZ* partitions the infrastructure geographically, and *iFogStor* does not implement any partitioning method. Thus, we consider that the partition time for both *iFogStor* and *iFogStorZ* is 0.

Table 5 shows *iFogStor* speedup (the exact solution) obtained while varying the number of zones for both *iFogStorZ* and *iFogStorG*. The speedup is defined as the ratio between the computation time of *iFogStor* and the one of *iFogStorZ* (respectively *iFogStorG*). We observe that heuristic methods present nearly the same speedup when using the same number of zones. For instance, both heuristics accelerate the computation time by 5 times when using 2 zones, and by 30 times when using 5 zones. Furthermore, *iFogStorG* can reduce the computation time by up to 473 times with 100 zones proving its scalability. Note that, *iFogStorG* has no partition limits;

To summarize, the computation time of the data placement solution in a Fog is very high (close to 327 seconds). However, it can be highly decreased by increasing the number of zones using divide and conquer heuristics (600 milliseconds with 100 zones) making it feasible in runtime.

5 CONCLUSION

In this paper, we presented an approximation heuristic to reduce the compute time of data placement strategies aiming minimizing the service latency in Fog infrastructures. We followed a divide and conquer approach in which the data placement problem is subdivided into several problems hence balancing the placement complexity. In order to minimize loss information during the dividing process which induce to a placement optimality losing, we used a method based on graph modeling and partitioning to subdivide the original data placement problem to several sub-problems. We have upgraded the Fog environment simulator *iFogSim* to apply and evaluate our heuristic using three different workloads. Our heuristic shown promising results as the data placement computation time is accelerated by more than 450 times when using 50 sub-problems with less to 5% of the global solution optimality losing.

An interesting direction for future work is investigating how can our heuristic approach can be adapted to minimize the compute time of services placement in the Fog infrastructure in order to minimize the overall service latency time, load balancing or availability enhancement.

REFERENCES

[1] [n. d.]. IBM ILOG CPLEX Optimization toolkit. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud. ([n. d.]). Accessed: 2017-09-20.
[2] [n. d.]. Metis, a graph partition tool. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview. ([n. d.]). Accessed: 2017-07-19.
[3] [n. d.]. Orange network infrastrucutre france. http://wholesalefrance.orange.fr/en/Content/Our-Folders/Our-Networks-Folder/Network-infrastructure-fundamentals. ([n. d.]). Accessed: 2017-09-20.
[4] K. Ashwin Kumar, A. Deshpande, and S. Khuller. 2013. Data Placement and Replica Selection for Improving Co-location in Distributed Environments. *ArXiv e-prints* (Feb. 2013). arXiv:cs.DB/1302.4168
[5] S Raja Balachandar and K Kannan. 2009. A new heuristic approach for the large-scale Generalized assignment problem. *Int J Math Comput Phys Elect Comput Eng* 3 (2009), 969–974.
[6] Ümit V. Çatalyürek, Kamer Kaya, and Bora Uçar. 2011. Integrated Data Placement and Task Assignment for Scientific Workflows in Clouds. In *Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing (DIDC '11)*. ACM, New York, NY, USA, 45–54. https://doi.org/10.1145/1996014.1996022
[7] A Di Nardo, M Di Natale, GF Santonastaso, V Tzatchkov, and VH Alcocer Yamanaka. 2014. Divide and conquer partitioning techniques for smart water networks. *Procedia Engineering* 89 (2014), 1176–1183.
[8] Robert W Floyd. 1962. Algorithm 97: shortest path. *Commun. ACM* 5, 6 (1962), 345.
[9] Vangelis Gazis, Alessandro Leonardi, Kostas Mathioudakis, Konstantinos Sasloglou, Panayotis Kikiras, and Raghuram Sudhaakar. 2015. Components of fog computing in an industrial internet of things context. In *Sensing, Communication, and Networking-Workshops (SECON Workshops), 2015 12th Annual IEEE International Conference on*. IEEE, 1–6.
[10] Lukasz Golab, Marios Hadjieleftheriou, Howard Karloff, and Barna Saha. 2014. Distributed Data Placement to Minimize Communication Costs via Graph Partitioning. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management (SSDBM '14)*. ACM, New York, NY, USA, Article 20, 12 pages. https://doi.org/10.1145/2618243.2618258
[11] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. 2016. iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments. *Technical Report CLOUDS-TR-2016-2, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia* (2016).
[12] Bruno Menegola. 2012. A Study of the k-way Graph Partitioning Problem. (2012).
[13] Mohammed Islam Naas, Philippe Raipin, Jalil Boukhobza, and Laurent Lemarchand. 2017. iFogStor: an IoT Data Placement Strategy for Fog Infrastructure. In *IEEE 1st International Conference on Fog and Edge Computing*. Madrid, Spain. https://doi.org/10.1109/ICFEC.2017.15
[14] Abdul Quamar, K Ashwin Kumar, and Amol Deshpande. 2013. SWORD: scalable workload-aware data placement for transactional workloads. In *Proceedings of the 16th International Conference on Extending Database Technology*. ACM, 430–441.
[15] S. Sarkar, S. Chatterjee, and S. Misra. 2015. Assessment of the Suitability of Fog Computing in the Context of Internet of Things. *IEEE Transactions on Cloud Computing* PP, 99 (2015), 1–1. https://doi.org/10.1109/TCC.2015.2485206
[16] Vernon Turner, John F Gantz, David Reinsel, and Stephen Minton. 2014. The digital universe of opportunities: Rich data and the increasing value of the internet of things. *IDC Analyze the Future* (2014).
[17] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. 2015. Fog computing: Platform and applications. In *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*. IEEE, 73–78.
[18] Shanhe Yi, Cheng Li, and Qun Li. 2015. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 37–42.
[19] Linquan Zhang, Xunrui Yin, Zongpeng Li, and Chuan Wu. 2015. Hierarchical virtual machine placement in modular data centers. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 171–178.