IoT Data Replication and Consistency Management in Fog computing

Mohammed Islam Naas, Laurent Lemarchand, Philippe Raipin, Jalil Boukhobza

Noname manuscript No.
(will be inserted by the editor)

IoT Data Replication and Consistency Management in Fog computing

**Mohammed Islam Naas · Laurent Lemarchand ·
Philippe Raipin · Jalil Boukhobza**

Received: date / Accepted: date

Abstract Fog Computing has emerged as a virtual platform extending Cloud services down to the network edge especially (and not exclusively) to host IoT applications. Data replication strategies have been designed to investigate the best storage location of data copies in geo-distributed storage systems in order to reduce its access time for different consumer services spread over the infrastructure. Unfortunately, due to the geographical distance between Fog nodes, misplacing data in such an infrastructure may generate high latencies when accessing or synchronizing replicas, thus degrading the Quality of Service (QoS). In this paper, we present two strategies to manage IoT data replication and consistency in Fog infrastructures. Our strategies choose for each datum, the right replica number and their location in order to reduce data access latency and replicas synchronization cost. This is done while respecting the required consistency level. Also, we propose an evaluation platform based on the simulator *iFogSim* to enable users to implement and test their own strategies for IoT data replication and consistency management. Our experiments show that when using our strategies, the service latency can be reduced by 30% in case of small Fog infrastructures and by 13% in case of large scale Fog infrastructures compared to *iFogStor*, a state-of-the-art strategy that does not use replication.

Keywords Internet of Things · Fog computing · Data Placement · Replication · Consistency · P-median

1 Introduction

Nowadays, the use of Internet of Things (IoT) [1] devices increased significantly. It is predicted that, by 2025, there will be 75 billion connected objects producing 79.4 ZB of IoT data [4, 6]. With this large growth, relying exclusively on traditional centralized Clouds for processing and storage purposes, would cause high access latencies and network congestion, degrading the Quality of Service (QoS) related to service latency of IoT applications [2, 7]. On the other hand, IoT objects have become ubiquitous, which imposes the use of a geo-distributed architecture in order to overcome the high latency and the network congestion limitations of the centralized Cloud-based platform [9].

Fog computing is a highly virtualized platform that provides compute, storage, and networking services between end devices and Cloud datacenters, typically, but not exclusively located at the edge of network [3, 8]. As shown in Figure 1, the Fog presents a geo-distributed and heterogeneous infrastructure in which Fog nodes such as routers, switches, points of presence (PoP) and set-top-boxes can be used to process and store data.

One may consider two state-of-the-art work classes for latency minimization in the context of Fog and IoT. The first one includes studies that focus on service placement [10–13]. The placement of services

Mohammed-Islam NAAS · Laurent Lemarchand
Univ. Bretagne Occidentale, Lab-STICC UMR 6285 F-29200, Brest, France
E-mail: (mohammedislam.naas)(laurent.lemarchand)@univ-brest.fr

Philippe Raipin
Orange, Rennes, France
E-mail: philippe.raipin@univ-brest.fr

Corresponding author: Jalil Boukhobza
ENSTA-Bretagne, Lab-STICC UMR 6285 F-29200, Brest, France
E-mail: jalil.boukhobza@ensta-bretagne.fr

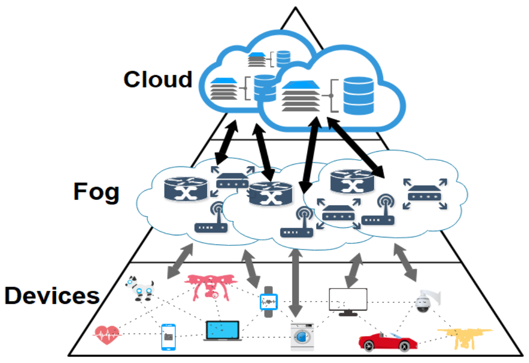


Fig. 1: Fog computing architecture.

can be constrained by properties other than latency, such as security/compatibility (e.g. services can be ran only in specific Fog nodes), availability, energy consumption or load balancing [14, 15]. The second class of studies investigated latency reduction by relying on data placement policies. For example, in [16], authors proposed *iFogStor*, an exact strategy based on integer programming to place IoT data in Fog infrastructures minimizing the overall service latency. Due to the high complexity of the data placement problem, the authors have proposed two heuristics to reduce the placement execution time [18]. There were no data replication considered in these approaches. In fact, maintaining a single copy per datum may raise several issues. First, concurrent reads may be delayed when the single server node cannot meet the required performance. Second, all read requests would be rejected if the storing node is down or unreachable, which may occur frequently in Fog infrastructures [20].

Adding data replication to the aforementioned state-of-the-art work may enhance the service latency [19]. Indeed, keeping multiple replicas on different Fog nodes and then satisfying each data request from the replica with the smallest access latency would decrease the global service latency. However, adding replication raises several issues to the system such as increasing the storage cost and the energy consumption or generating a high network traffic when distributing data copies [28]. Another major issue is to maintain data consistency between replicas (that is the ability of the system to adapt with its data updates and to synchronize replicas to the same last version [21]). This process of replica synchronization may add a latency overhead to the system which may vary according to the number of replicas and their location.

Managing data consistency in distributed storage systems is a known problem. Indeed, several solutions have been proposed [29–33]. With the advent of IoT and Fog computing, this problem has a new dimension due to the massive amount of data to be managed, the large number of objects to be taken into account, and the heterogeneity of the system (heterogeneity of data, platforms, protocols, etc.) [21]. In fact, in IoT context, a storage system can be used to store data for several applications with different data consistency requirements. For instance, in a smart home, security applications need to access to the latest information in order to trigger an alarm (e.g. intrusion, fire) as soon as the event happens, in order to make a fast decision hence reducing the damage. On the contrary, applications for energy management may tolerate some delay in making decisions (e.g. turn off the heating system). The former application requires a *strong consistency* model [29] (requiring always the latest data version) whereas the latter may use a *weak consistency* model [29] which may provide a non-updated information.

In this paper, we address the data placement problem with replicas by taking into account the heterogeneity of applications. As Fog infrastructure can be variable in size, we designed two heuristics, namely *iFogStorS* and *iFogStorP*. Both heuristics manage IoT data replication and consistency in Fog infrastructures while reducing both data access and replicas synchronization latencies. The first strategy, *iFogStorS*, is dedicated to small infrastructures (tens of nodes) and has a good latency reduction factor but its computation time increases exponentially with the number of Fog nodes. The second strategy, *iFogStorP*, is dedicated to large infrastructures (up to thousands of nodes), its performance are lower than the previous one but the computation time is better. Both strategies find for each datum the required number of replicas and their locations while respecting its consistency level.

We first rely on existing problem formulations such as Median [34], P-median [37] and shortest path [36] to place data replicas. Then, we simulate data production and consumption context to manage the data consistency. On the other hand, due to the lack of experimental tools in this context, we propose an extension to the simulator *iFogSim* [26], a Fog and IoT environments simulator dedicated for managing

IoT services placement and scheduling. Our extension enables users to implement and test their strategies for IoT data replication and consistency management.

The contributions of this paper are the following:

- *iFogStorS*, a data replication and consistency management strategy which finds the placement of P replicas minimizing both data access and replicas synchronization latencies considering all shortest path nodes between producers and consumers,
- *iFogStorP*, an evolution of *iFogStorS*, which selects P-median from the shortest paths nodes to place P replicas hence accelerating the computation time,
- *iFogSim* simulator extension to enable users to evaluate their own strategies toward IoT data replication and consistency management.

The experiments show that the service latency can be reduced by 30% when using *iFogStorS* in case of small Fog infrastructures, and by 13% when using *iFogStorP* in case of large scale Fog infrastructures compared to *iFogStor*, a state-of-the-art strategy that does not use replication.

The rest of this paper is structured as follows. Section II gives some background knowledge about the Median, P-median, *iFogStor* and *iFogSim* which are used in this work. Section III details our strategies for IoT data replication and consistency management in Fog infrastructures, and Section IV describes the *iFogSim* simulator extension. Section V discusses the evaluation part and Section VI presents some related work. Section VII concludes this paper and gives perspectives for future work.

2 Background

In this section, we first give the formulation of the Median and the P-median problems and then we describe both the *iFogStor* strategy and the *iFogSim* simulator. These formulations and tools are used in this work. More precisely, we used the Median and the P-median to formulate and to solve the replicas placement problem, *iFogStor* as a reference method to compare our strategies, and *iFogSim* to perform the experimentation.

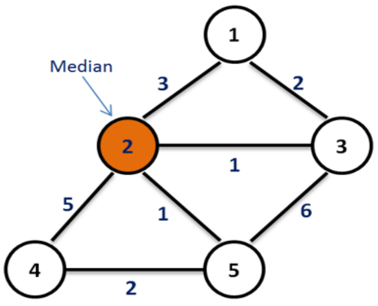
2.1 Median problem

In graph theory, the median vertex of a given undirected weighted-edge graph is the vertex for which the sum of the shortest paths costs to all other vertices is the smallest one [34]. Mathematically, this can be formulated as follows.

Let an undirected graph G be given as $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is the vertex set and $E = \{e_1, e_2, ..., e_m\}$ is the edge set. Let $c_{i,j}$ be the cost of a shortest path existing between the vertex v_i and the vertex v_j. The median vertex $Y \in V$ can be found by solving the following linear formula.

$$Y = argmin_{j\in[1..n]} \sum_{i\in[1..n]} c_{i,j}$$

An example of this problem is illustrated in Figure 2. In this problem, the median is represented by the vertex 2 with which the cost to join all others vertices equals 8.



Source	Destination	Path	Cost
1	2	1-2	3
2	2	2-2	0
3	2	3-2	1
4	2	4-5-2	3
5	2	5-2	1

Fig. 2: An example of the Median problem.

2.2 P-median problem

The P-median is a generalization of the median problem. The difference is in the number of chosen medians. Here, the problem consists to find a subset of P medians that minimizes the sum of the shortest path costs existing between medians and all others vertices. Each vertex should be attached to only its nearest median. This problem can be formulated by the following linear system [37].

$$\begin{cases} Min & \sum\limits_{i\in[1..n]} \sum\limits_{j\in[1..n]/\{i\}} c_{i,j}.x_{i,j} \\ Subject & \\ & \sum\limits_{i\in[1..n]} x_{i,j} = 1 \qquad \forall j \in [1..n] \quad (1) \\ & x_{i,j} \leq x_{i,i} \qquad \forall i \in [1..n], \forall j \in [1..n] \quad (2) \\ & \sum\limits_{i\in[1..n]} x_{i,i} = P \qquad (3) \\ & x_{i,j} \in \{0,1\} \qquad \forall i \in [1..n], \forall j \in [1..n] \quad (4) \end{cases}$$

With:

- $x_{i,i} = 1$ means that the vertex v_i is chosen as a median, otherwise, 0
- $x_{i,j} = 1$ means that the vertex v_j is affected to the median v_i, otherwise, 0
- Constraint 1 ensures that each vertex v_j is attached to one and only one median v_i
- Constraint 2 verifies that a vertex is attached to a median
- Constraint 3 sets the number of medians to P
- Constraint 4 forces variables to be set by binary values.

From the complexity point view, Hakimi and Kariv have shown that the P-median problem is NP-hard [35].

2.3 iFogStor strategy

iFogStor was proposed in [16] to address the data placement problem in the context of Fog computing and IoT. It was formulated as a Generalized Assignment Problem (GAP) [25] and solved by linear programming. The authors considered a system infrastructure that contains a set of storage nodes (Fog nodes and data centers) denoted $SN = \{sn_1, ..., sn_n\}$, and an amount of IoT data denoted $D = \{d_1, ..., d_m\}$ which are generated and consumed by a set of IoT services. An objective function was 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 respected two constraints: (i) the capacity of storage nodes was not exceeded, (ii) all data items were stored. *iFogStor* formulation was described by the following linear system.

$$\begin{cases} Min & \sum\limits_{i\in[1..m]} \sum\limits_{j\in[1..n]} \alpha_{i,j}.x_{i,j} \\ Subject & \\ & \sum\limits_{i\in[1..m]} S(d_i).x_{i,j} \leq F(sn_j) \quad \forall j \in [1..n] \\ & \sum\limits_{j\in[1..n]} x_{i,j} = 1 \qquad \forall i \in [1..m] \\ & x_{i,j} \in \{0,1\} \qquad \forall i \in [1..m], \forall j \in [1..n] \end{cases}$$

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
- $S(d_i)$ is the size (in bytes) of d_i
- $F(sn_j)$ is the free storage capacity (in bytes) of sn_j.

Note that *iFogStor* leads to an NP-hard optimization problem [16]. The authors in [16] used *CPLEX MILP* [39] to solve it.

2.4 iFogSim simulator

iFogSim [26] is a Fog and IoT environments simulator developed in Java. As shown in Figure 3, *iFogSim* is composed of a set of physical entities that may be of different types: *FogDevice* (i.e. Fog node), *Sensor*, and *Actuator*. *iFogSim* is also composed of a set of logical entities to model the application scenario. For instance, as shown in Figure 3, *AppModule* models IoT services, *AppEdge* defines the data dependency between a pair of IoT services, and *Tuple* models the fundamental unit of communication between entities that are the data.

In *iFogSim*, IoT services can be placed and then scheduled for execution among Fog nodes driven by some objectives related to end-to-end latency minimization, network utilization, energy consumption or operational cost reduction. The *AppModulePlacement* and the *AppModuleScheduler* in Figure 3 are abstractions of IoT service placement and scheduling policies respectively, and the *AppModuleMapping* holds, for each IoT service, the Fog node to which it is affected.

In [17], the authors have proposed an extension to *iFogSim* to enable the design of data placement strategies. As shown in Figure 3, three components were added by this extension:

1. *Data Placement*: it offers to users a set of functionalities to compute a data placement using integer programming.
2. *Infrastructure Partitioning*: this component offers the possibility to subdivide the simulated infrastructure into several parts and runs a data placement strategy on each one of them hence accelerating the computation time.
3. *Workload Repartition*: it includes a generic scenario of smart city that encompasses different configurations of data flows offering to users the possibility to test their strategies with different data workloads.

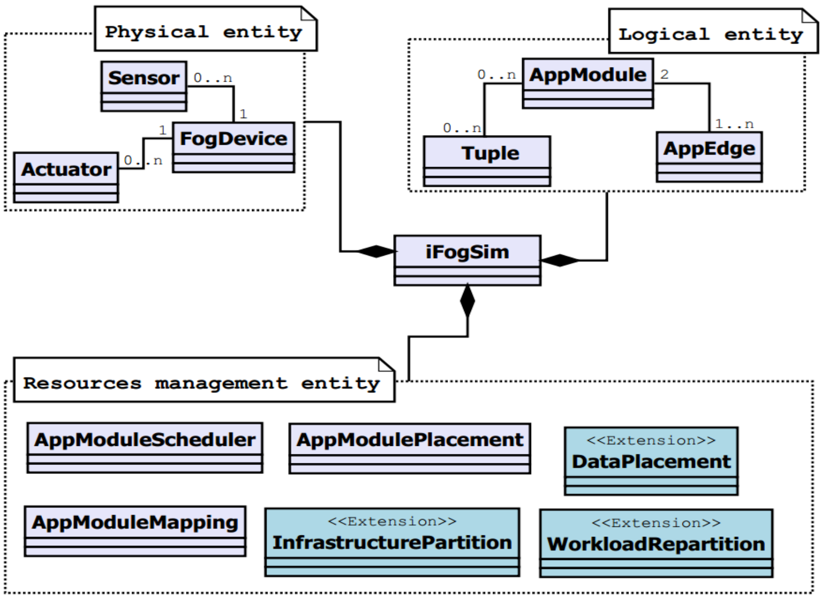


Fig. 3: Class diagram of the main components of *iFogSim*.

3 Proposed data replication and consistency management methods

In this section, we start by giving an overview of the contributions of this paper. Then, we give an Integer Linear Programming (ILP) formulation of the data replication problem in Fog infrastructures. After that, we give an exact method and two heuristic-based methods to solve it. Each proposed solution is detailed in a dedicated section. Finally, we will describe how consistency has been managed.

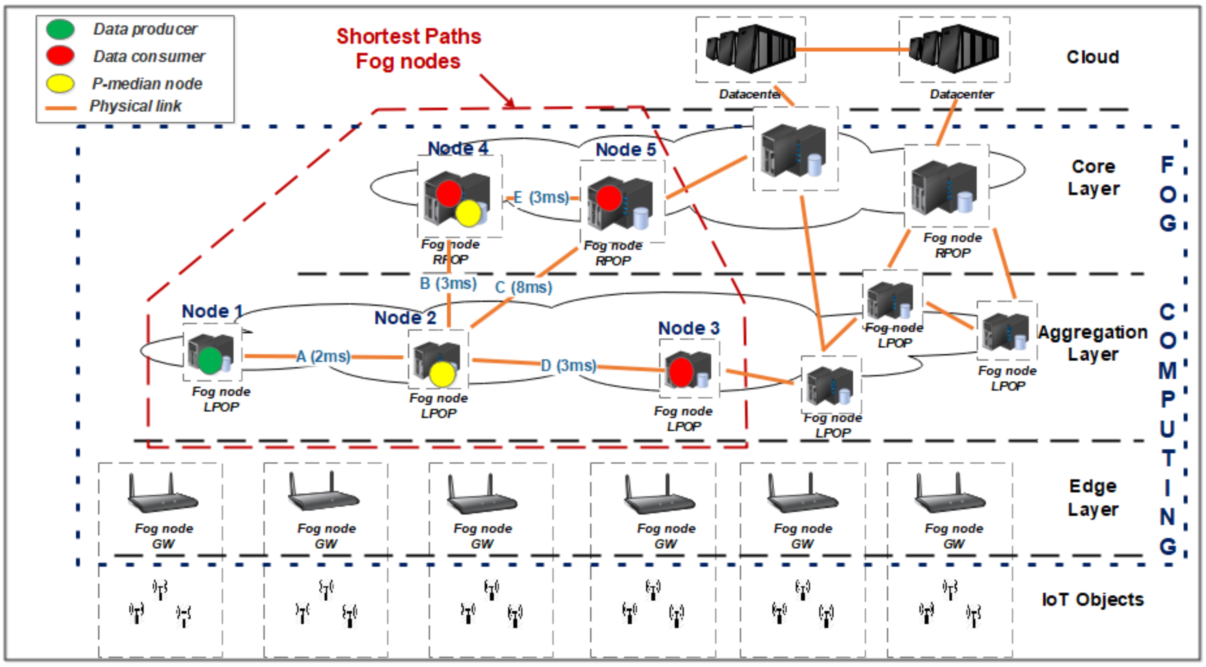


Fig. 4: System architecture.

3.1 Overview

As shown in Figure 4, we consider a very simplified system architecture (to ease the comprehension) that consists of a set of sensors, a set of Fog nodes, a set of datacenters and a set of IoT services. In this architecture, a Fog node can be located in the edge layer (e.g. a GW), in the aggregation layer (e.g. a LPOP) or in the core layer of the network (e.g. a RPOP) [16]. We refer to IoT services and to sensors that produce data as data producers, and IoT services that consume data as data consumers. An IoT service can be a producer and a consumer at the same time. IoT data can be stored in different locations of the Fog architecture and datacenters.

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 locations and their storage capacities [16]. Indeed, this assumption may hold as our data placement and replication strategies are to be implemented by a service provider who has the knowledge of the IoT services location (by deploying a service instance placement strategy such as proposed in [11]) and their data dependencies. For the remaining information, it can be inferred by deploying a software component which monitors the state of Fog nodes, latencies and storage capacities, as proposed in [7].

First, sensors collect data from the physical environment. Once collected, data are replicated and sent through gateways to the storage nodes. Data replication is used in order to reduce the access latency. Once stored, associated IoT service instances (data consumers) can retrieve data from storage nodes for analysis / processing sake. To ensure the reliability of IoT services, each pair of IoT data - IoT service should obey to a given consistency requirement which is specified in the Service Level Agreement (SLA) adopted between the service provider and the service owner, and it is assumed to be known and to be guaranteed by the storage system. Indeed, the system should select a synchronized data replica from the existing ones to respond to a given data access request emitted by an IoT service. As aforementioned, replicas synchronization process may add a latency overhead to the system. This latency depends on the number of replicas, their locations and on the associated consistency level.

Our objective is to reduce the overall service latency which encompasses the data transfer and the replicas synchronization latencies. To do so, we propose two heuristics : *iFogStorS* and *iFogStorP*. Both heuristics find for each produced IoT datum in the system, the number of replicas to be stored and their locations minimizing the overall service latency and conforming to the required consistency level.

In the following sections, an exact method and the two proposed heuristics for managing data replication are described.

Table 1: Notation table.

Notation	Description
FN	The set of Fog nodes, with $\|FN\| = n$
Fn_i	The Fog node fn_i
$F(Fn_i)$	The free storage capacity (in bytes) of the Fog node fn_i
D	The data amount to be replicated and stored in FN, with $\|D\| = m$
d_j	The datum d_j, with $d_j \in D$
$S(d_j)$	The size (in bytes) of the datum d_j
Q_j	The used data consistency protocol for the datum d_j
W_j	The number of available replicas needed to process a write request on the datum d_j
R_j	The number of available replicas needed to process a read request on the datum d_j
p	The number of replicas of a datum, with $p \in [P_{min}, P_{max}]$
$C_n^{j,p}$	The set of possible assignments of p replicas of the datum d_j in FN
c_p	The number of possible assignments of p replicas of the datum d_j in FN
$A_c^{j,p}$	The c^{th} assignment of p replicas of the datum d_j in FN
$L_c^{j,p}$	The system latency overhead generated by the c^{th} assignment $A_c^{j,p}$ of p replicas of the datum d_j in FN
$y_{i,c}^{j,p}$	Selection variables of Fog nodes that form the $c^t h$ assignment of p replicas of the datum d_j in FN
$x_c^{j,p}$	Selection variable of assignments of the datum d_j

3.2 Data replication problem formulation

In this section, we give a formulation of the data replication problem using an Integer Linear Programming method. The used notations are summarized in Table 1.

Let $FN = \{fn_1, fn_2, \ldots, fn_n\}$ be a Fog infrastructure and $D = \{d_1, d_2, \ldots, d_m\}$ the data amount to be replicated and stored in FN.

Let P_{min} and P_{max} be the minimum and maximum number of replicas used in the system, respectively.

Let $Q_j = [W_j, R_j]$ be the data consistency protocol based on `Quorum` (see Section 3.6.2) used in the system for the datum d_j, with W_j is the number of invoked replicas to answer a write (replicas synchronization) request, and R_j is the number of invoked replicas to answer a read (data access) request.

Let $C_n^{j,p} = \{A_1^{j,p}, A_2^{j,p}, \ldots, A_{c_p}^{j,p}\}$ be the set of possible placement assignments of p replicas of the datum d_j in FN, with $c_p = |C_n^{j,p}| = \frac{n!}{p!(n-p)!}$ (the number of combination nCp of p replicas from n nodes), and $A_c^{j,p}$ is the c^{th} assignment of p replicas of the datum d_j in the infrastructure FN. Note that, for each assignment $A_c^{j,p}$ of p replicas, p Fog nodes are chosen to place these p replicas.

In this system, each assignment $A_c^{j,p}$ generates $L_c^{j,p}$ latency overhead. This latency represent the synchronization (write) time of W_j replicas of the datum d_j plus the retrieving (read) time of R_j replicas of the datum d_j. Since the latency $L_c^{j,p}$ depends on the locations of replicas, the data consistency protocol used for the datum d_j, and the arrival time of read and write requests for the datum d_j, this latency cannot be known in advance, but simulation-based or machine learning-based methods can help to estimate its value (in this work, we used a simulation-based method to estimate the replica synchronization latency, see Section 3.3).

The goal of our data replication and placement strategy is to find for each datum $d_j \in D$, the number of replicas $p \in [P_{min}, P_{max}]$ and their locations in FN, which minimize the overall system latency. That is to select for each $d_j \in D$, the assignment $A_c^{j,p} \in C_n^{j,p}$ which minimizes the global system latency. This returns to solving the following linear system.

$$\begin{cases} Min \quad \sum_{j\in[1,m]} \sum_{p\in[P_{min},P_{max}]} \sum_{c\in[1,c_p]} L_c^{j,p}.x_c^{j,p} \\ Subject \\ \quad \sum_{p\in[P_{min},P_{max}]} \sum_{c\in[1,c_p]} x_c^{j,p} = 1 \; \forall j \in [1,m] & (1) \\ \quad \sum_{i\in[1,n]} y_{i,c}^{j,p} = p \qquad \forall x_c^{j,p} = 1 & (2) \\ \quad y_{i,c}^{j,p} \leq x_c^{j,p} \qquad \forall i \in [1,n], \forall j \in [1,m], \forall p \in [P_{min},P_{max}], \forall c \in [1,c_p] & (3) \\ \quad \sum_{j\in[1,m]} S(d_j).y_{i,c}^{j,p} \leq F(fn_i) \quad \forall x_c^{j,p} = 1, \forall i \in [1,n] & (4) \\ \quad y_{i,c}^{j,p} \in \{0,1\} \qquad \forall i \in [1,n], \forall j \in [1,m], \forall p \in [P_{min},P_{max}], \forall c \in [1,c_p] & (5) \\ \quad x_c^{j,p} \in \{0,1\} \qquad \forall j \in [1,m], \forall p \in [P_{min},P_{max}], \forall c \in [1,c_p] & (6) \end{cases}$$

With:

- $x_c^{j,p} = 1$ means that the c^{th} assignment of p replicas is chosen to replicate and place the datum d_j in FN, and 0 otherwise.
- $y_{i,c}^{j,p} = 1$ means that the Fog node fn_i is chosen to place a replica of the datum d_j for the c^{th} assignment of p replicas, and 0 otherwise
- $S(d_j)$ is the size (in bytes) of d_j
- $F(fn_i)$ is the free storage capacity (in bytes) of fn_i
- Constraint 1 ensures that one and only one assignment (combination) of $p \in [p_{min}, p_{max}]$ replicas is chosen (among all possible combinations of replicas assignments) for each datum d_j.
- Constraint 2 ensures that p Fog nodes are chosen to place p replicas
- Constraint 3 ensures that the chosen Fog node fn_i to place a replica of datum d_j for the c^{th} assignment, belongs to the c^{th} assignment
- Constraint 4 ensures that the data amount assigned to a given Fog node fn_i (specified by $y_{i,c}^{j,p} = 1$) must be lower than its free storage capacity $F(fn_i)$
- Constraint 5 ensures that the selection variables of Fog nodes of the c^{th} assignment, are binary
- Constraint 6 ensures that the selection variables for assignments of the datum d_j, are binary.

3.3 Motivation: the exact method

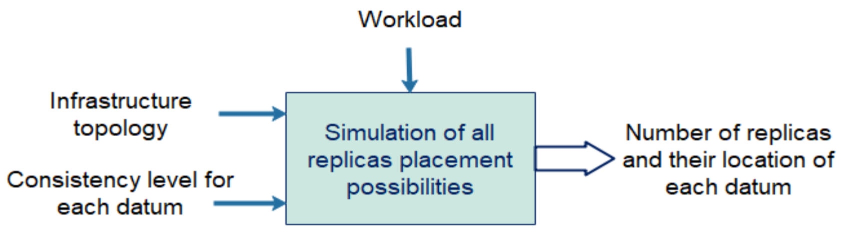

Workload
Infrastructure topology
Consistency level for each datum
Simulation of all replicas placement possibilities
Number of replicas and their location of each datum

Fig. 5: Exact method model.

A possible way to solve the ILP formulation presented above that is to find the optimal number of replicas and their placement for minimizing both data access and replica synchronization latencies consists to enumerate all replica placement options. As shown in Figure 5, based on the infrastructure topology (latencies, IoT services placement and Fog nodes free storage capacities), the predicted workload and the required consistency level per datum, a micro-simulation (considering only the underlying datum) is ran in order to estimate the generated latency for each placement case. Once all placement options simulated, for the sake of simplification, the best solution with the minimum latency is selected for each datum (instead of solving the linear system using an ILP solver). This is done by satisfying the following two constraints: (i) the amount of data affected to a Fog node must be lower than the free storage capacity in that Fog node, and (ii) the number of data replicas must be included between a minimum and a maximum fixed in advance for each data. The overall service latency is represented by the sum of all data access and replicas synchronization latencies. Hereinafter the description of the search space size (the number of simulated cases) of the exact method followed by an example, which could give an intuition of its complexity by multiplying the complexity of a placement case computation by the search space size.

3.3.1 Exact method search space size

The total number of simulated cases Nbc is computed by:

$$Nbc_d = C_n^{p_{min}} + C_n^{p_{min+1}} + ... + C_n^{p_{max}}.$$
$$Nbc = d \times Nbc_d.$$

With:

- Nbc_d : the number of simulations for each datum.
- Nbc : the total number of simulations.
- d : the number of data elements to be stored.
- n : the number of Fog nodes (data hosts).
- p_{min} : the minimum number of replicas.
- p_{max} : the maximum number of replicas.
- C_n^p : a combination of p from n, $C_n^p = \frac{n!}{p!(n-p)!}$.

The total number of cases is equal to the number of data multiplied by the sum of the possible placement options for each datum for all possible number of replicas.

We can bound the Nbc_d using the following formula:

$$\frac{n!}{p_{min}!(n-p_{min})!} \leq Nbc_d \leq 2^n$$

Explanation:

First, we demonstrate the left part of the inequality.
As mentioned above, the number of simulations for each datum is

$$Nbc_d = C_n^{p_{min}} + C_n^{p_{min+1}} + ... + C_n^{p_{max}} \text{ ...(1)}$$

From the equation (1) we get

$$C_n^{p_{min}} = Nbc_d - (C_n^{p_{min+1}} + ... + C_n^{p_{max}}) \text{ ...(2)}$$

As the combination of p from n is always greater than 0, whenever p is greater or equal to 0, we have

$$C_n^p > 0, p \geq 0 \text{ ...(3)}$$

From the equation (2) and the equation (3) we get

$$C_n^{p_{min}} \leq Nbc_d \text{ ...(4)}$$

As mentioned above, the factorial representation of a combination of p from n is

$$C_n^{p_{min}} = \frac{n!}{p_{min}!(n-p_{min})!} \text{ ...(5)}$$

By substituting the factorial representation of a combination given by the equation (5), in the equation (4), we get the left part of the inequality

$$\frac{n!}{p_{min}!(n-p_{min})!} \leq Nbc_d$$

Now, we demonstrate the right par of the inequality. Considering (1) and (3) we get

$$Nbc_d = C_n^{p_{min}} + ... + C_n^{p_{max}} \leq C_n^0 + ... + C_n^{p_{min}} + ... + C_n^{p_{max}} + ... + C_n^n \text{ ...(6)}$$

Furthermore, it is well known that the sum of the binomial coefficients

$$C_n^0 + C_n^1 + ... + C_n^n = 2^n \text{ ...(7)}$$

By substituting the equation (7) in the equation (6), we get the right side of the inequality

$$Nbc_d \leq 2^n$$

3.3.2 Example

To replicate 200 data elements with a number of replicas varying from 3 to 5, in three infrastructures: FN_1, FN_2 and FN_3 involving respectively 20, 25 and 30 data hosts, the total number of simulated cases is:

- For FN_1, $Nbc = 200 \times (C_{20}^3 + C_{20}^4 + C_{20}^5) = 4,297,800$
- For FN_2, $Nbc = 200 \times (C_{25}^3 + C_{25}^4 + C_{25}^5) = 13,616,000$
- For FN_3, $Nbc = 200 \times (C_{30}^3 + C_{30}^4 + C_{30}^5) = 34,794,200.$

As shown in Figure 6, from the large increase in number of possible placement cases Nbc by varying the number of Fog nodes or by varying the number of data replicas, we can conclude that we encounter a combinatorial explosion problem.

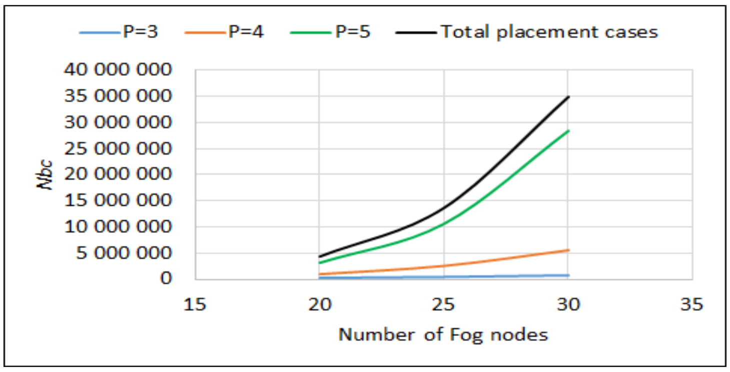


Fig. 6: Number of placement cases according to the Fog infrastructure size and the number of replicas.

3.4 Heuristic 1: iFogStorS

As we have seen above, the number of possible placement options enumerated by the exact method can be very large. Thus, the idea behind the heuristic *iFogStorS* is to reduce the number of possible replica placement possibilities by considering only nodes on the shortest paths between producers and consumers rather than the whole infrastructure nodes.

For example in Figure 4, considering the network topology information presented in Table 2, the set of shortest paths nodes selected by *iFogStorS* as candidates to store the data generated by the Node 1 and consumed by Node 3, Node 4 and Node 5, are $\{Node\ 1, Node\ 2, Node\ 3, Node\ 4, Node\ 5\}$.

Table 2: Network topology.

Source	Destination	Shortest Path	cost (ms)
Node 1	Node 3	A-D	5
Node 1	Node 4	A-B	4
Node 1	Node 5	A-B-E	8

Our idea is relevant in case the number of shortest path nodes is small. In this case, the number of placement cases generated can be calculated in runtime. This happens for a small infrastructure comprising tens of Fog nodes (it depends of course on the performance of the server running the calculations), or in case of large infrastructure in which data consumers are concentrated in a given region (or part) of the infrastructure, for instance, in Figure 4, all consumers of the data generated by the $Node$ 1 are located in the left-part of the infrastructure. This latter case implies that the optimal replica placement solution could, very likely, be located in this same part of the infrastructure near the consumers. This eliminates the calculation of the placement of replicas in the rest of the infrastructure. However, in case of large infrastructures with highly distributed consumers, the second heuristic *iFogStorP*, presented in the following section, is more relevant.

Algorithm 1 describes the operating process of the heuristic *iFogStorS*. First, based on the infrastructure topology, *iFogStorS* computes the set of nodes of the shortest path existing between producers and consumers. Then, for each possible number of replicas P, *iFogStorS* simulates each placement case of P replicas and estimates its latency overhead. Finally, it chooses the number of replicas and their locations for which the sum of both data access and replica synchronization latencies is minimized. Like the exact method, this is done by satisfying two constraints: (i) the amount of data affected to a Fog node must be lower than the free storage capacity in that Fog node, and (ii) the number of data replicas must be included between a minimum and a maximum fixed in advance for each data. This process is repeated for each datum produced in the system.

Algorithm 1: iFogStorS algorithm

```
  Input: Workload, Producers and consumers locations, Network latency between Fog nodes, Required consistency
         level
  Output: Number of replicas and their locations
1 function iFogStorS ()
2     foreach data do
3         Get all shortest paths nodes between producers and consumers
4         foreach $P \in [p_{min}, p_{max}]$ do
5             foreach $C \in$ all possible placement cases of $P$ replicas do
6                 Estimate the latency overhead of C
7         Choose C with the minimum latency overhead
8     return All data replicas locations
```

3.4.1 iFogStorS search space size

The number of simulated replica placement cases Nbc done by *iFogStorS*, is computed by the following formula.

$$Nbc_i = (C_{n_i}^{p_{min}} + C_{n_i}^{p_{min+1}} + ... + C_{n_i}^{p_{max}}).$$
$$Nbc = \sum_{i \in D} Nbc_i.$$

- i : a datum.
- Nbc_i : the number of simulations for the datum i.
- D : the set of data to be stored.
- n_i : the number of shortest path Fog nodes existing between the producer of datum i and its consumers.

The total number of cases is equal to the sum of the possible placement cases for each datum. The complexity of *iFogStorS* can be computed by multiplying the number of simulated cases by the computational complexity of a simulation, plus the complexity of a shortest path computation algorithm (e.g. in the case of Floyd-Warshall algorithm [36] which is used in this work, the complexity is $O(n^3)$, with n being the number of Fog nodes).

3.4.2 Example

Considering the same infrastructure configuration example presented in Section 3.3.2 with FN_1 (i.e. 20 Fog nodes), and assuming that for each datum, the number of the shortest path nodes is 7 (one third of the number of Fog nodes in the infrastructure). The total number of simulated cases is computed as follows:

$$Nbc = \sum_{i=1}^{200} (C_7^3 + C_7^4 + C_7^5) = 18,200.$$

For the aforementioned infrastructure configuration, *iFogStorS* reduces the number of simulated cases by 236 times as compared to the exact method.

However, in case of large scale infrastructures involving thousands of Fog nodes with highly distributed consumers, selecting all shortest path nodes may result in a large number of nodes. For instance, beyond 20 shortest path nodes, the number of possible cases for replica placement exceeds million (see the example in Section 3.3.2). This gives a huge number of possible replica placement options giving rise to a combinatorial explosion problem. To solve this issue, we propose *iFogStorP*, a second heuristic to manage data replication and consistency in case of large scale infrastructures.

3.5 Heuristic 2: iFogStorP

iFogStorP is an improvement of *iFogStorS* in terms of complexity. It reduces drastically the number of possible replica placement cases by working with median nodes. In fact, rather than enumerating all possible cases of replica placement across the shortest paths nodes set, it selects P-median nodes to place P replicas. As shown in Figure 4, *iFogStorP* selects two median nodes (Node 2 and Node 4) from the set of shortest path nodes (Node 1, Node 2, Node 3, Node 4 and Node 5) to place two replicas for the data generated by Node 1 and consumed by Node 3, Node 4 and Node 5. Thus, *iFogStorP* does just one simulation for a given number of replicas. The operating process of *iFogStorP* is described in algorithm 2. *iFogStorP* starts by computing all shortest path nodes existing between the producer of each datum and its consumers. Then, for each number of replicas, it selects P placement nodes from the shortest path nodes solving a P-median problem and estimates its latency overhead. Finally, it chooses for each datum the number of replicas (i.e. medians) which minimizes both data access and replicas synchronization latencies. As the exact method and *iFogStorS*, *iFogStorP* should satisfy the two constraints: (i) the amount of data affected to a Fog node must be lower than the free storage capacity in that Fog node, and (ii) the number of data replicas must be included between a minimum and a maximum fixed in advance for each data.

Algorithm 2: iFogStorP algorithm

```
Input: Workload,Producers and consumers locations, Network latency between Fog nodes, Required consistency
       level
Output: Number of replicas and their locations
1 function iFogStorP ()
2     foreach data do
3         Get all shortest paths nodes between producers and consumers
4         foreach $P \in [p_{min}, p_{max}]$ do
5             Select P placements to place P replicas
6             Estimate the latency overhead
7         Choose P with the minimum latency overhead
8     return All data replicas locations
```

3.5.1 iFogStorP search space size

The number of simulated replica placement cases Nbc by *iFogStorP*, is calculated as follows:

$$Nbc = d \times (C_{p_{min}}^{p_{min}} + C_{p_{min}+1}^{p_{min}+1} + ... + C_{p_{max}}^{p_{max}}) =$$
$$d \times (p_{max} - p_{min} + 1).$$

The total number of cases is equal to the number of data multiplied by the sum of each combination of P replicas from P medians with P ranging from the minimum number of replicas to the maximum number of replicas. As $C_p^p = 1$, the possible placement cases for each datum is equal to: $p_{max} - p_{min} + 1$.

The complexity of *iFogStorP* can be computed by multiplying the number of simulated cases by the computational complexity of a simulation. As *iFogStorP* uses shortest path nodes and P-median computation algorithms, the the complexity of *iFogStorP* should include these algorithms' complexities. As aforementioned, the complexity of the shortest path nodes computation is $O(n^3)$, while the complexity of P-median computation is $O(n^2.P^2)$, with n being the number of Fog nodes and P the number of medians [38].

3.5.2 Example

Considering the same infrastructure configuration presented in Section 3.3.2 with FN_1 (i.e. 20 Fog nodes), the number of possible replica placement cases Nbc is computed as follows.

$$Nbc = 200 \times (5 - 3 + 1) = 600.$$

For this infrastructure configuration, *iFogStorP* reduces Nbc by a factor of 7,163 as compared to the exact method and by 30.33 times as compared to *iFogStorS*.

Note that the computation of P-median nodes may add an overhead to the execution time of *iFogStorP*. This overhead will be discussed in the evaluation Section.

3.6 Data consistency protocols

In the previous section, we proposed two heuristics to find the best number of replicas and their locations for each datum to reduce both data access and replicas synchronization latencies while respecting their consistency protocols. One must notice that we can affect a consistency level on a per datum basis in our work. In this section, we describe how these consistency protocols are considered in this work. We first give the set of assumptions and then we explain how data consistency protocols were deployed in this work.

3.6.1 Assumptions

In this work, we have considered a set of assumptions in order to manage data consistency for both aforementioned strategies, which are given below [5, 21, 22, 33]. To facilitate the understanding of certain assumptions, we will refer to Figure 7 in which an example of read and write requests on data replicas is illustrated.

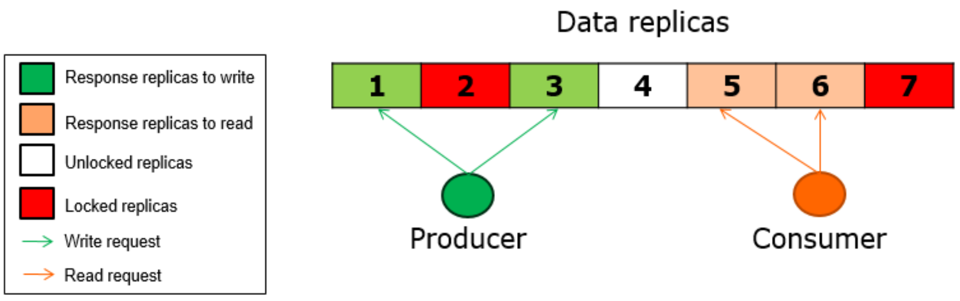


Fig. 7: Write/reading protocol.

1. A consistency protocol is defined by the number of replicas that should respond to read/write requests [21]. That means, to consider a given read/write request as a completed request, it should be made simultaneously on a required number of available replicas which we call *response replicas* (replicas 1 and 3 for write request and replicas 5 and 6 for read request in Figure 7).
2. Response replicas are chosen from the set of the nearest (to reduce the latency) and unlocked replicas (i.e. replicas for which there are no concurrent requests) to the requester.
3. Response replicas for a given write request are locked for all other concurrent read/write requests (to avoid conflict) [22], for example, replicas 1 and 3 in Figure 7.
4. Response replicas for a given read request are locked for all other concurrent write requests (replicas 5 and 6 in Figure 7). So, several concurrent read requests could be made on the same replica [22].
5. Write requests are delayed and processed in background for non-response replicas (the replica number 4 in Figure 7) [5, 33].
6. When a write request arrives, and one or several replicas are locked by other read/write requests (for example, replicas 2 and 7 in Figure 7), this write request is saved in the system for the locked replicas in order to synchronize them once unlocked.
7. Write requests for which the required number of response replicas is not satisfied are called blocked writes. They are timestamped and saved in the system. They are triggered when the number of required response replicas for writes is satisfied following a chronological order (from older to recent) in order to maintain the *Linearizability* of writes (i.e. all write requests are ordered chronologically by their arrival time in the system and all requests always see the effects of the preceding ones) [33].
8. Read requests for which the required number of response replicas is not satisfied will not be served and an ”unavailable data” event is sent to the requester.

3.6.2 Data consistency protocols implementation

As aforementioned, in our work, the consistency level is expressed by the number of replicas that should respond to read/write requests. In case of a strong consistency requirement, a quorum based replica selection algorithm is applied [21]. This algorithm consists in selecting from N unlocked replicas, QW replicas to respond to a given write request and QR replicas to respond to a given read request. Fixing QW and QR must satisfy two conditions: $QW > N/2$ and $QW + QR > N$. This is done in order to force the system to select a replica with the last version. An example with $N = 5$, $QW = 3$ and $QR = 3$ is illustrated in Figure 8. This example consists in writing/reading simultaneously in/from three available replicas. First, a write request arrives to the system which chooses 3 unlocked replicas from the 5 available ones to respond to this request. By completing this request, 3 replicas (the responding ones) will have the recent version and the 2 others replicas will have an old version. After that, a read request arrives to the system requiring $QR = 3$, the system chooses 3 replicas from the 5 available ones to respond to this request. Logically, at least one replica (2 replicas in this example) from the chosen ones must show the recent version of data assuring thereby a strong consistency context.

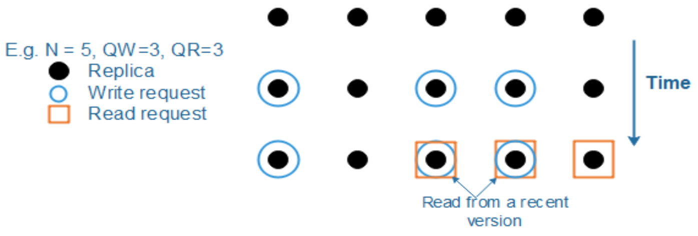


Fig. 8: An example of a strong consistency protocol.

On the other hand, in case of a weak consistency requirement, we use the quorum based algorithm but without satisfying both conditions on the number of response replicas. That means, QW and $QW + QR$ can be less than $N/2$ and N, respectively. An example with $N = 5$, $QW = 3$ and $QR = 1$ is illustrated in Figure 9. This example consists in writing in three available replicas and reading from one available replica. Contrary to the previous example, here the read request requires only one response replica ($QR = 1$) thereby the system cannot be forced to chose a response replica with the recent version.

For the weak consistency, we assume that replicas will be eventually updated [33]. That is, they will converge to the same latest version in case there is no new data update requests.

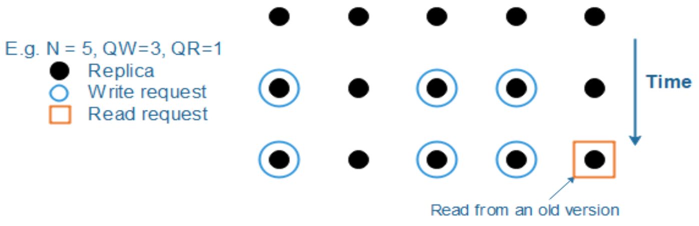


Fig. 9: An example of a weak consistency protocol.

4 iFogSim extension

In this section, we will describe our extension to *iFogSim* which can be used to test and evaluate strategies.

As shown in Figure 10, we added three main components to resource management entities in *iFogSim*: *ConsistencyManager*, *ConsistencyProtocol* and *DataReplicationAndConsistencyStrategy*. Also, we implemented three interfaces to model and solve all shortest path nodes, Median and P-median problems which are used in this extension. The other classes are sub-components which inherit the *DataReplicationAndConsistencyStrategy* and the *ConsistencyProtocol* components.

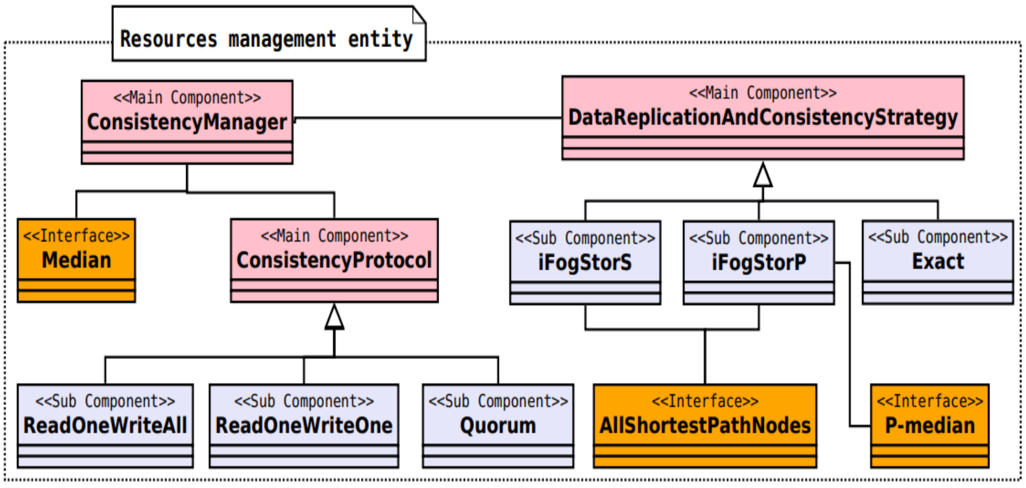


Fig. 10: Class diagram of the main added components to *iFogSim*.

In brief, the *ConsistencyManager* is responsible for handling replicas locking, unlocking and synchronization as well as managing read and write requests. The *ConsistencyProtocol* is an implementation of three data consistency management protocols. Each generated datum in the system should obey to one protocol (obviously, users can define new protocols). The *DataReplicationAndConsistencyStrategy* implements strategies for managing data replication and consistency in Fog infrastructures. These strategies could be employed by users to compare with their methods. They rely on the consistency protocol of the previous component. The set of added components are detailed below.

4.1 Consistency Manager

This component is the most important one in this extension. It manages the data consistency in *iFogSim* by ensuring mutual exclusion between read/write requests and maintaining replica synchronization. To do so, it uses a set of functions, for instance, for identifying the set of replicas of a given datum and locking/unlocking the set of response replicas for a given read/write request. Also, this component ensures that all read/write requests should be done according to the required data consistency protocol associated to each datum. Furthermore, this component manages locked and blocked write requests by implementing chronological queues (see, respectively, the 6^{th} and the 7^{th} elements of the set of assumptions given in Section 3.6.1).

It is worth to mention that, to respond to a read request, the *ConsistencyManager* uses the *Median* API to choose a replica among the set of response replicas to send data to the requester. In fact, a median of all response replicas is used to solve a consensus problem of the most recent version of the stored data in the set of response replicas. In case several replicas have the updated version, the nearest one (in terms of latency) to the requester is chosen. The *Median* API is used to formulate and solve a median problem; see Section 2.1.

4.2 Consistency protocol

In this extension, we implemented three consistency protocols. Each datum in the system uses a given protocol. Users have the possibility to add new protocols. The implemented protocols are the following:

- *Read One-Write All*: this is a strong consistency protocol. It consists in writing simultaneously on all replicas and reading from one replica [40].
- *Read One-Write One*: this is a weak consistency protocol in which a given read/write request requires to be applied to one (unlocked) replica to be validated [5].
- *Quorum*: with this protocol, the user defines the number of replicas that should respond to a read/write request. This protocol could be implemented for strong as well as for weak consistency as mentioned in Section 3.6.2.

4.3 Data replication and consistency strategy

This component is an abstraction of three data replication and consistency management strategies defined in Section 3:

- *Exact*: an implementation of the exact strategy which consists in enumerating all replica placement options to choose the optimal solution.
- *iFogStorS*: an implementation of the first heuristic proposed in this work. It enumerates all possibilities of replica placement within shortest path nodes rather than the whole infrastructure. This strategy uses the *AllShortestPathNodes* API to compute the shortest path nodes between data producers and consumers.
- *iFogStorP*: an implementation of the second heuristic proposed in this work which reduces the number of replica placement options by choosing P median nodes from the shortest paths nodes to place P replicas. This strategy uses the *AllShortestPathNodes* API to compute the shortest path nodes and then the *P-median* API to compute the P median nodes.

5 Evaluation

This section presents the evaluation part of our strategies for managing data replication and consistency in Fog infrastructures. We first describe the used evaluation methodology and the use-case scenario. Then, we give the experimental setup. Finally, results will be discussed.

5.1 Methodology

We evaluated a set of metrics on a given set of strategies on which we applied different workloads with data having different consistency levels.

5.1.1 Evaluated strategies

The considered replicas placement and consistency management strategies are the following:

a) *iFogStor* [16]: this is the data placement strategy from state-of-the-art that we consider as a reference in our evaluation for comparison sake. It is an exact method that finds the location to store one copy of each datum in order to minimize the overall service latency. This strategy does not use any replica, it was presented in Section 2.3.
b) *3-Replicas*: this strategy stores three replicas for each datum. It computes replicas location by solving the P-median problem presented in Section 2.2, using the Cplex MILP solver [39]. Note that, this strategy does not take into account the latency induced by the data consistency management, when computing the replicas location. We introduce this strategy also for comparison sake to emphasize the usefulness of considering synchronization cost in the problem solving.
c) *Exact*: this strategy enumerates all data replication options to choose the best storage location solution minimizing the overall service latency while respecting the required consistency level for each datum.
d) *iFogStorS*: this strategy reduces the number of replicas location options by considering only the set of shortest path nodes existing between producers and consumers. In this work, we used the Floyd-Warshall algorithm to compute the set of shortest path nodes.
e) *iFogStorP*: this strategy chooses P medians from the set of shortest path nodes to place P replicas. In this work, we used the Cplex MILP [39] to solve *iFogStorP*'s P-median sub-problems.

5.1.2 Metrics of comparison

Several metrics were evaluated:

a) *Overall service latency*: the main objective in this work is to reduce the service latency in Fog infrastructures. Thus, for each strategy, we evaluated the overall generated latency time, that is the sum of generated latencies for accessing all the data in the system.

Table 3: Placement strategies computation time components.

	Problem formulation	**Problem solving**	**Simulation**	**Shortest path nodes**	**P-median resolving**
iFogStor	X	X			
3-Replicas					X
Exact			X		
iFogStorS			X	X	
iFogStorP			X	X	X

b) *Computation time*: as Fog infrastructures are dynamic and their topology often changes, management strategies should be feasible at runtime (online) [18]. Thus, in our experiments, we measured the computation time taken by each strategy. The computation time is the time spent to compute the data locations by formulation and solving data placement (sub-)problems (e.g. the shortest path nodes and P-median sub-problems for *iFogStorP*) as well as the simulation time taking by *iFogSim* to simulate the Fog infrastructure with a given workload and a given data placement solution in order to compute the system latency related to this data placement solution. As shown in Table 3, the computation time encompasses (i) the one-copy placement problem formulation and solving times for *iFogStor*, (ii) the P-median solving time for the *3-Replicas* strategy, (iii) the simulation time for the *Exact* strategy, (iv) the shortest path nodes computation time and the simulation time for *iFogStorS*, and (v) the shortest path nodes computation time, the P-median resolving time and the simulation time in case of *iFogStorP*.

c) *Number of replicas*: we measured the average number of replicas generated by applying each strategy. This metric can give an idea about the network traffic, the storage cost and the energy consumption of each strategy. These are all correlated to the the number of replicas generated.

d) *Unsatisfied read requests*: another important metric to be evaluated is the number of unsatisfied read requests. This metric translates the availability of the service according to a given number of replicas. In fact, the unsatisfied read requests happen when the required number of available replicas (for read requests) is not satisfied, see Section 3.6.1. That means, replicas are currently occupied by write requests.

5.1.3 Data workloads

We used two different types of synthetic data workload distribution [18]: (i) *Zoned* in which data are consumed by IoT services located in the neighborhood of producers Fog nodes (such as traffic management applications), and (ii) *Distributed* in which data are consumed by IoT services located anywhere in the infrastructure (such as in online gaming applications). Note that we applied these two data workload distribution mainly in case of large scale infrastructures in which the distribution nature of the used data workload would make a significant difference (in replicas synchronization time and in number of shortest path nodes) in contrast to small infrastructures in which all nodes may be considered as neighbors. In this latter case, we only show results of the *Zoned* data workload (as they are similar to distributed ones).

5.1.4 Data criticality

As in a given IoT application data may not be all subject to the same consistency protocol [21], we varied the amount of data that have strong consistency requirements (critical data). We considered this proportion to be 0%, 10%, 20% and 30% of the total data volume. In each case, the rest of data are assigned a weak consistency requirement. These levels of data criticality are applied for both *Zoned* and *Distributed* workloads, and for all proposed strategies except the *3-Replicas* strategy. For the latter, data are managed following a weak data consistency protocol. In effect, with this consistency management protocol *3-Replicas* shows the best performance. Indeed, by increasing criticality, the latency would increase due to replicas synchronization. Note that *iFogStor* uses only one copy of data and it does not need a consistency protocol (of course, read and write requests are mutually exclusive, see Section 3.6.1). This variation of data workload distribution and data consistency requirement allows to properly study the behavior of each strategy.

We used the generic use-case of smart city proposed in [16], where several types of sensors (humidity, temperature and luminosity) collect environmental data and send them to a variety of IoT services for

Table 4: Infrastructure configuration.

Strategy	Workload	Criticality	Infrastructure	Objective
iFogStor *3-Replicas* *Exact* *iFogStorS* *iFogStorP*	Zoned	all	1 DC 1 RPOP 1 LPOP 10 GW	To compare our proposed method to manage data replication and consistency with the state of the art
iFogStor *3-Replicas* *iFogStorS* *iFogStorP*	Zoned	all	1 DC 1 RPOP 1 LPOP 50 GW	To compare our proposed heuristics
iFogStor *3-Replicas* *iFogStorP*	Zoned Distributed	all	1 DC 10 RPOP 100 RPOP 1,000 GW	To test the scalability of *iFogStorP*

Table 5: 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 6: Free storage capacities.

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

processing and analytical sake. These IoT services are implemented within an infrastructure encompassing Fog nodes and Cloud data-centers. As shown in Figure 4, Fog nodes consist of gateways (GW), Local PoPs (LPOP) and Regional PoPs (RPOP), arranged hierarchically.

5.2 Experimental Setup

In our simulations, we varied the infrastructure configuration according to various objectives, and for each one, we used *iFogStor* as a reference strategy for comparison. As shown in Table 4, these configurations are the following:

- We used a small infrastructure encompassing 10 Fog nodes in order to compare our proposed heuristics with the *Exact* method which is not scalable, and this, using the *Zoned* workload with all data criticality levels defined previously.
- We used a medium sized infrastructure involving about 50 Fog nodes in order to test the scalability of *iFogStorS* using the *Zoned* workload with all data criticality levels. However, we cannot test *iFogStorS* with large number (hundreds) of Fog nodes due to its limited scalability.
- Finally, we evaluated a large infrastructure with up to 1,000 Fog nodes to test the scalability of *iFogStorP* using both *Zoned* and *Distributed* workload with various data criticality levels.

In these simulations, we have fixed 100 sensors per GW. Table 5 shows the considered network latencies between nodes, and Table 6 illustrates available storage capacity on each type of data host [16, 24].

As aforementioned, in our work, the data consistency is represented by the number of replicas responding to a given read/write request. In our experiments, we varied the number of replicas of each datum from 1 to 5 [16, 18] except for the 3-Replicas strategy, for which it is fixed to 3. Data management strategies find the best combination between the total number of replicas and QW and QR that minimize the overall service latency while respecting its consistency requirement.

In our experiments, data exchange is achieved on a packet basis. We set the size of data packets generated by sensors to 96 bytes and data packets generated by IoT services to 960 bytes [24]. For service data sharing, we varied the number of consumers sharing data randomly from 1 to 10 for each datum. The evaluation was done on a server with 48 *Xeon E5-2650* CPU cores clocked at 2.2 GHz and 352 GB of RAM.

5.3 Results and Discussion

Note that *Exact* and *iFogStorS* strategies were not evaluated with all infrastructure configurations due to their limited scalability (see Figures 11, 12, 13 and 14).

5.3.1 Latency time

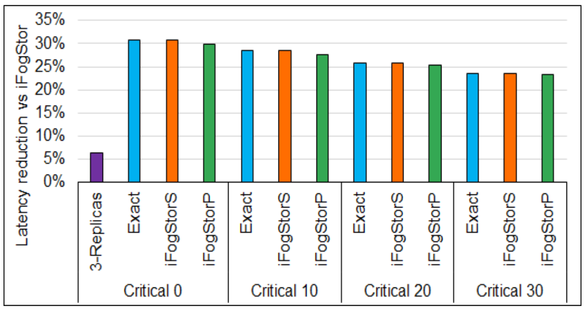
Figure 11 (a), (b), (c) and (d) show the latency reduction of our data management strategies compared to *iFogStor*, using infrastructure configurations involving 10, 50 and 1000 Fog nodes and applying various data criticality and data workloads. We remember here that for the purpose of simplifying the evaluation part, the *3-Replicas* strategy is evaluated only with *Critical 0* which represents its best case of latency. We observe that:

a) In Figure 11 (a), when there are no critical data (*Critical 0*), the *3-Replicas* strategy reduces the service latency by only 6%, whereas the *Exact*, *iFogStorS* and *iFogStorP* strategies reduce the service latency by about 30% compared to *iFogStor*, the one-copy placement strategy. Also, the service latency is reduced by 23% when increasing the level of data criticality to 30% (i.e. *Critical 30*) for all proposed strategies (i.e. *Exact*, *iFogStorS* and *iFogStorP*) compared to *iFogStor*. This performance degradation is induced by the implementation of a strong consistency protocol for 30% of data, which delays read/write requests when replicas are locked
b) When increasing the number of Fog nodes to 50 (see Figure 11 (b)), we observe that the service latency reduction for *3-Replicas* remains unchanged (about 6%), while for *iFogStorS* and *iFogStorP* it is 25% and 19%, respectively, when there is no critical data, and it is about 19% and 13%, respectively, with 30% critical data.
c) With an infrastructure of 1000 Fog nodes, we see a difference in the enhancement of the *3-Replicas* strategy with regards to iFogStor when the type of data workload is changed from *Zoned* to *Distributed*, from 2% to 8%, respectively (see Figure 11 (c) and (d)). Tn fact, this is not a performance degradation of the 3-Replicas strategy, but rather a better behavior of iFogStor. Indeed, both strategies enhance the performance for the *Zoned* workload compared to the *Distributed* one because of lower network latencies. However, with 3 replicas, the synchronization cost is higher, thus the enhancement of *Zoned* over the *Distributed* workload is smaller. This reduces the enhancement of 3-replicas with regards to *iFogStor* in case of *Zoned* workloads. Compared to *iFogStorP*), in case of *Zoned* workload, the number of replicas is higher (for *3-Replicas*) which generates additional latencies. Indeed, Figure 13 (c) shows that *iFogStorP* uses only 2.2 replicas on average (against 3 for *3-Replicas*) to have a reduction of 19% of the service latency. For the *Distributed* workload, *iFogStorP* uses about 2.8 replicas on average (see Figure 13 (d)). For the *iFogStorP* strategy, the latency reduction is stable whatever the workload as the number of replicas is adjusted for each data criticality level (see Figure 11 (c) and (d)). Indeed, the service latency is reduced by about 17% when there is no critical data and 13% when 30% of data are subject to a strong consistency protocol. The more the data are critical, the higher is the synchronization latency.

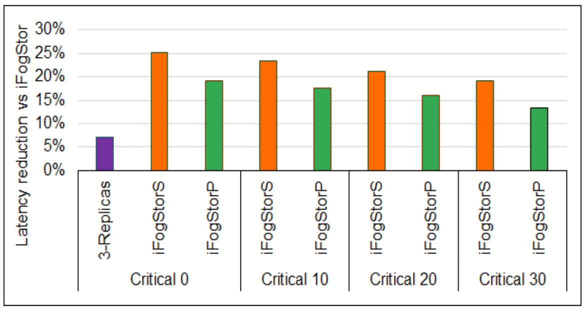
5.3.2 Computation time

Figure 12 shows the total computation time and the time duration of each phase for the different strategies using 10, 20 and 1000 Fog nodes in the simulated infrastructure. We recall that the computation time encompasses (i) the one-copy placement problem formulation and solving times for *iFogStor*, (ii) the P-median solving time for the *3-Replicas* strategy, (iii) the simulation time for the *Exact* strategy, (iv) the shortest path nodes computation time and the simulation time for *iFogStorS*, and (v) the shortest path nodes computation time, the *P-median* resolving time and the simulation time in case of *iFogStorP*, see Table 3.

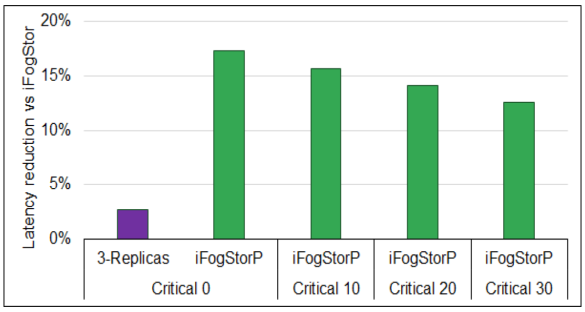
One can observe that the computation time of the *Exact* and *iFogStorS* strategy is very high compared to *iFogStor*, *3-Replicas* and *iFogStorP*. For instance, as shown in Table 7, it is about 200 seconds and 10000 seconds for *iFogStorS* and *Exact* respectively, when using only 10 Fog nodes. This proves their limited scalability. Also, we notice that the computation time of the *3-Replicas* strategy increases faster than that of *iFogStorP*. In fact, the computation time of the *3-Replicas* strategy increases from 0.19 second when using 10 Fog nodes to 4429.49 seconds when using 1000 Fog nodes, whereas the computation time of the *iFogStorP* increases from 1.38 second when using 10 Fog nodes to 193.43 seconds when using 1000 Fog nodes (see Table 7), making a speed up of 22.5 times compared to that of the *3-Replicas* strategy (in case of 1000 Fog nodes).



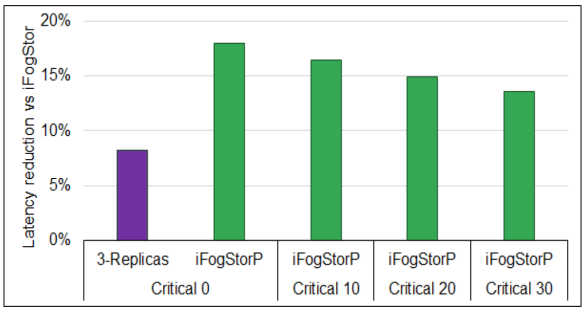
(a) Latency-reduction-10.



(b) Latency-reduction-50.



(c) Latency-reduction-1000-Zoned.



(d) Latency-reduction-1000-Distributed.

Fig. 11: Latency results.

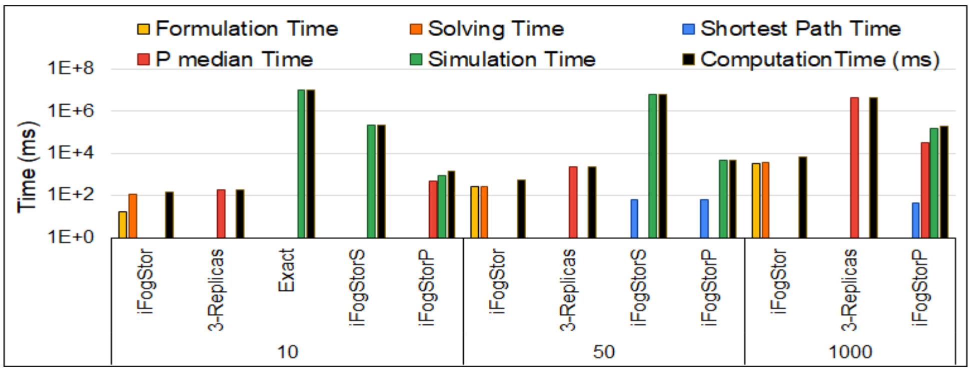


Fig. 12: Computation time results.

Table 7: Computation time results.

	Computation time (s)		
	10 Fog nodes	50 Fog nodes	1000 Fog nodes
iFogStor	0.14	0.55	7
3-Replicas	0.19	2.26	4429.49
Exact	10197	Not evaluated	Not evaluated
iFogStorS	210.96	6044.44	Not evaluated
iFogStorP	1.38	5	193.43

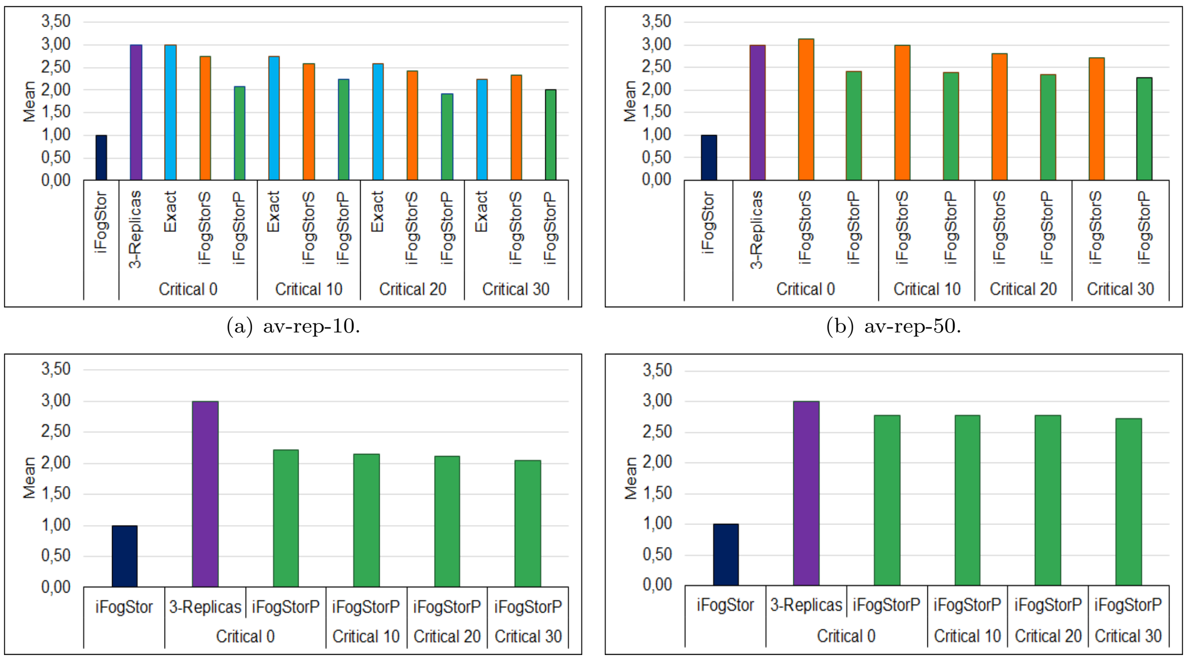
5.3.3 Number of replicas

Figure 13 (a), (b), (c) and (d) show the average number of replicas per datum when using each strategy. We recall that *iFogStor* and the *3-Replicas* places one replica and 3 replicas per datum, respectively. We observe that:

a) In Figure 13 (a), when applying the *Exact* strategy, the average number of replicas is approximately 3 when there is no critical data. This number is lower when increasing the amount of critical data. For instance, it is 2.3 when using *Critical 30*. In fact, when there are no critical data, there is no replicas synchronization latency. Thus, the number of replicas is higher in order to minimize the data access latency. On the contrary, increasing the amount of critical data reduces the number of replicas as the synchronization latency grows higher.

b) Also, the average number of replicas using *iFogStorP* is lower than that of *iFogStorS* which is smaller than that of the *Exact* solution, see Figure 13 (a) and (b). This can be explained by the fact that *iFogStorP* has fewer node options to store replicas (the medians of the shortest paths nodes), as a consequence, it stores fewer replicas in comparison to *iFogStorS* and *Exact* strategies which have a broader vision of the infrastructure. This is why *Exact* and *iFogStorS* place more replicas and reduce further the service latency than *iFogStorP*.

c) In Figure 13 (c) and (d), the average number of replicas of *iFogStorP* on the *Zoned* workload is lower than that in the *Distributed* one as geographically spread consumers require more replicas than closer ones in order to reduce their latency. Indeed, the average number of replicas of *iFogStorP* is about 2.25 when using the *Zoned* workload, and about 2.7 about when using the *Distributed* one.



(a) av-rep-10. (b) av-rep-50.

(c) av-rep-1000-Zoned. (d) av-rep-1000-Distributed.

Fig. 13: Replicas number results.

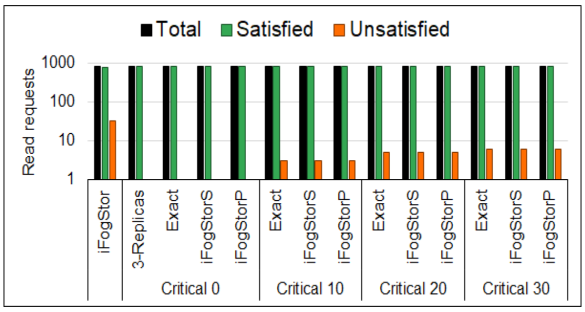
5.3.4 Unsatisfied read requests

Figure 14 (a), (b), (c) and (d) show the number of satisfied and unsatisfied read requests and the total read requests of each data management strategy using the different configurations. We notice that the distribution nature of the used workload does not have a considerable impact on the amount of satisfied requests. This is why we present the results using a single workload distribution which is the *Distributed* one, see Figure 14. We observe that:

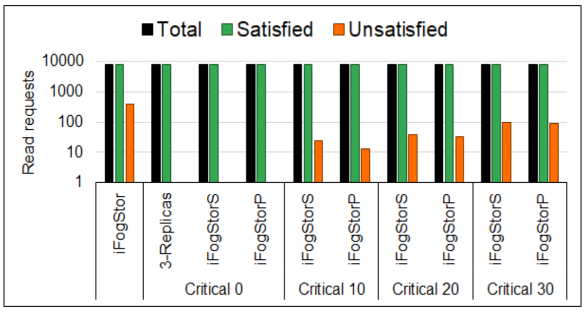
a) In Figure 14 (a), using *iFogStor* (one copy), the system does not satisfy about 3% of the read requests, whereas, using the *3-Replicas*, *Exact*, *iFogStorS* and *iFogStorP* strategies, the system responds to almost all read requests. For instance, it rejects (in the worst case) less than 0.8% of read requests with 30% criticality. In fact, *iFogStor* uses only one copy per datum and having concurrent read and write requests in this case has a higher probability. In the contrary, the other strategies use several replicas allowing to write in some replicas and to read from others. The proportion of unsatisfied requests grows higher with data criticality rate as data locking happens more frequently when data criticality is higher.

b) When increasing the number of Fog nodes to 50 (see Figure 14 (b)), the system responds to almost all read requests and the percentage of unsatisfied read requests is always under to 1% for all strategies.

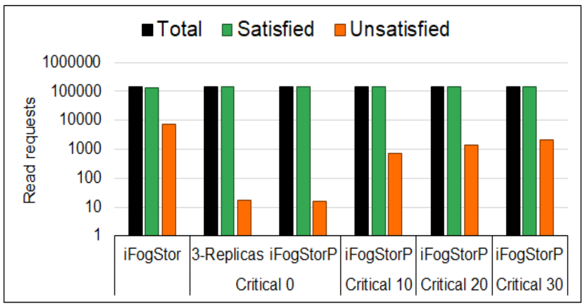
c) When increasing the number of Fog nodes to 1000 (see Figure 14 (c) and (d)), we observe that the distribution nature of the used workload does not have a considerable impact on the amount of satisfied requests, and both figures show approximately the same results. Moreover, when using *iFogStor* the system rejects more read requests due to the high concurrency of access operations on a single data copy. For instance, 10% of the total read requests are rejected by *iFogStor*, whereas, when using *3-Replicas* and *iFogStorP*, the percentage of unsatisfied read requests is 0.03% with the workload *Critical 0*. So, if we ignore failure and congestion, we can conclude that the system availability (for read requests) is 30 times higher using *3-Replicas* and *iFogStorP* than using *iFogStor*. Moreover, considering failure and congestion, the system availability using *3-Replicas* and *iFogStorP* is better than *iFogStor* because this latter uses only one copy which will be unavailable if its host node is offline.



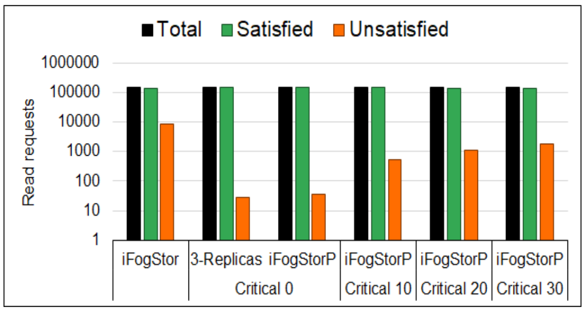
(a) read-10.



(b) read-50.



(c) read-1000-Zoned.



(d) read-1000-Distributed.

Fig. 14: Read requests results.

5.4 Summary

Our experimental work can be summarized as follows:

1) In case of small infrastructures *iFogStorS* can be used in order to reduce the service latency. In fact, it reduces the service latency by 21% to 30% (while *3-Replicas* reduces the service latency by only 6%) compared to *iFogStor*. To achieve this latency reduction, *iFogStorS* stores on average between 2 and 3 replicas per datum depending on the amount of critical data in the system. *iFogStorS* (as for 3-Replicas) also offers a better availability by satisfying almost all read requests while *iFogStor* rejects 3% of reads. We notice that the computation time of *iFogStorS* is about 1000 times greater than *iFogStor* and *3-Replicas* as they did not use simulation to compute data placement. We notice that, if we want to speed up the computation time by sacrificing performance a little, we can use *iFogStorP* which has a computation time less than 2 seconds (0.2 second for *3-Replicas* and 1.3 second for *iFogStorP*).
2) In case of large scale infrastructures encompassing up to thousands of Fog nodes, *iFogStorP* can be used to reduce the service latency by 13% to 18% (while *3-Replicas* reduces the service latency by 2% to 8%) compared to *iFogStor*. Moreover, using *iFogStorP* (as for 3-Replicas) the system can answer 99% of read requests against 90% using *iFogStor*. To achieve this performance, *iFogStorP* stores on average 2.5 replicas per datum which makes the system twice more resilient against data loss and

node failures which are very common in Fog infrastructures, as compared to *iFogStor*. Concerning the computation time, we showed that *iFogStorP* is about 22.5 times faster than the *3-Replicas* strategy.

6 Related work

In [21], the authors have proposed *FogStore*, a strategy for managing replica placement and data context-sensitive consistency in Fog computing. *FogStore* subdivides the Fog infrastructure to failure groups then it chooses the "N" closest ones to each data producer to place "N" replicas for which the number is fixed by the user. However, *FogStore* neither considers consumers data access nor replicas synchronization latencies to place replicas. For the data consistency management, *FogStore* defines several consistency levels according the number of replicas that should respond to read requests (e.g. one, all, quorum). Then, it maps each incoming consumer read request to the appropriate consistency level based on the consumer context, in particular the location.

In [23], *Xie et al.* proposed a strategy for short-latency and low-overhead data placement and retrieval services for Edge computing, called Greedy Routing for Edge Data (GRED). This strategy computes the data placement by hashing data identifiers which are defined by a string of characters. In order to make data replication, the authors suggested a naive algorithm that concatenates the serial number of the data copy (i.e. replica) with the data identifier to form a new string. By hashing the new string, the placement of the corresponding data copy is calculated. Nevertheless, this strategy does not consider the data access latency in replica placement computation and does not manage data consistency.

In [24], *Huang et al.* have proposed *MultiCopyStorage*, a latency-aware multiple data replicas placement strategy for Fog infrastructures. The experiment result demonstrates that *MultiCopyStorage* strategy reduces the overall latency by 6% compared to *iFogStor*. Also, the authors have proposed *iFogStorM*, an exact replica placement strategy based on MILP model. However, in this work, data consistency management was not addressed and the replicas synchronization latency was not considered when calculating the data placement. Indeed, replicas synchronization may add a high latency overhead to the system due to the large distribution of Fog nodes.

Aral et al. proposed a distributed data dissemination approach in Edge/Fog computing [19]. Their approach is two-fold. First, they proposed a decentralized, dynamic, and online algorithm for creation/replacement/removal of data replicas based on Uncapacitated Facility Location Problem formulation (UFLP) [27]. Second, they proposed a low overhead messaging methodology to notify edge/Fog entities about nearby replicas to send them their future data requests, instead of remote cloud storage. Similar to the two previous studies, data consistency and replicas synchronization latency were not addressed in this work.

In [16], authors proposed *iFogStor*, a one-copy data placement strategy to minimize the data access latency in Fog infrastructures. The authors have formulated the data placement problem as a Generalized Assignment Problem [25] using integer programming model. As GAP is NP-hard, the authors have proposed two heuristics based on *Divide and Conquer* to reduce the placement computation time [16, 18]. Both heuristics subdivide the infrastructure into several parts generating thereby several sub-problems with a lower complexity. The global data placement problem is built by aggregating sub-problems solutions. However, in these approaches, there were no data replication considered.

To the best of our knowledge, there are no other research proposed to manage data placement, replication or consistency in Fog computing environment.

7 Conclusion

In this paper, we proposed an exact method and two heuristic-based strategies, *iFogStorS* and *iFogStorP*, to manage data replication and consistency in a Fog infrastructure. The first heuristic is dedicated to small infrastructures including several tens of Fog nodes, whereas the second one is dedicated to large infrastructures including up to thousands of Fog nodes. Both strategies choose for each datum, the number of replicas and their location while reducing the sum of data access and replicas synchronization latencies.

In this work, we investigated and gave solutions to the problem of replica placement in a Fog infrastructure considering the service latency and data consistency. However, geo distributed data storage systems operate in a trade-off between latency, consistency, availability and network partitioning tolerance [21]. As a continuation of the work presented in this paper, we plan to adapt the proposed solutions to consider also the availability and partitioning tolerance when placing data replicas. Furthermore, to the best of our knowledge, currently there is no large-scale platform managing the availability and partitioning tolerance in Fog computing. Thus, we plan to further extend the *iFogSim* simulator to address this lack of management platforms. Also, by adding the support to the availability management and partitioning tolerance, the *iFogSim* simulation time would be high. Thus, it is interesting to make *iFogSim* able to manage simulations in a parallel way whether on a single machine or in a cluster in order to reduce the simulation time in case of large scale infrastructures.

The Java source code used in this work is made available in:
https://github.com/medislam/iFogSimWithDataConsistency

References

1. A. Kertesz, T. Pflanzner, T. Gyimothy. A Mobile IoT Device Simulator for IoT-Fog-Cloud Systems. Journal of Grid Computing 17, 529–551 (2019). https://doi.org/10.1007/s10723-018-9468-9
2. P. Kochovski, V. Stankovski, V. Gec, et al. Smart Contracts for Service-Level Agreements in Edge-to-Cloud Computing. Journal of Grid Computing 18, 673–690 (2020). https://doi.org/10.1007/s10723-020-09534-y
3. M. Ghobaei-Arani, A. Souri, A. Rahmanian. Resource Management Approaches in Fog Computing: a Comprehensive Review. Journal of Grid Computing 18, 1–42 (2020). https://doi.org/10.1007/s10723-019-09491-1
4. L. Horwitz, The future of iot miniguide: The burgeoning iot market continues. Cisco (2019).
5. Tunable consistency in cassandra, http://cassandra.apache.org/doc/latest/architecture/dynamo, accessed: 2020-03-09.
6. E. Tyleckova, D. Noskievicova, The role of big data in industry 4.0 in mining industry in serbia, System Safety: Human - Technical Facility - Environment. doi.org/10.2478/czoto-2020-0020 (2020).
7. S. Sarkar, S. Chatterjee, S. Misra, Assessment of the suitability of fog computing in the context of internet of things, IEEE Transactions on Cloud Computing PP (99), pp. 1–1. doi:10.1109/TCC.2015.2485206 (2015).
8. F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things. MCC '12: Proceedings of the first edition of the MCC workshop on Mobile cloud computingAugust, pp. 13–16 doi.org/10.1145/2342509.2342513 (2012).
9. A. Coutinho, F. Greve, C. Prazeres, J. Cardoso, Fogbed: A rapid-prototyping emulation environment for fog computing, in: IEEE International Conference on Communications (ICC), IEEE, pp. 1–7 (2018).
10. O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, G. Pavlou, On uncoordinated service placement in edge-clouds, in: IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 41–48. doi:10.1109/CloudCom.2017.46 (2017).
11. O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, Towards qos-aware fog service placement, in: IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pp. 89–96. doi:10.1109/ICFEC.2017.12 (2017)
12. M. Taneja, A. Davy, Resource aware placement of iot application modules in fog-cloud computing paradigm, in: IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1222–1228. doi:10.23919/INM.2017.7987464 (2017)
13. Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, F. Desprez, Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed iot applications in the fog, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC '18, pp. 751–760. doi:10.1145/3167132 (2018).
14. N. Daneshfar, N. Pappas, V. Polishchuk, V. Angelakis, Service allocation in a mobile fog infrastructure under availability and qos constraints, in: IEEE Global Communications Conference (GLOBECOM), IEEE, pp. 1–6 (2018).
15. A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, J. P. Jue, QoS-aware Dynamic Fog Service Provisioning, ArXiv e-prints. arXiv:1802.00800 (2018).
16. M. Naas, P. Raipin, J. Boukhobza, L. Lemarchand, iFogStor: an IoT Data Placement Strategy for Fog Infrastructure, in: IEEE 1st International Conference on Fog and Edge Computing, Madrid, Spain, pp. 97-104, doi: 10.1109/ICFEC.2017.15. (2017).

17. M. Naas, J. Boukhobza, P. Raipin, L. Lemarchand, An extension to ifogsim to enable the design of data placement strategies, in: 2nd IEEE International Conference on Fog and Edge Computing, ICFEC, Washington DC, pp. 1–8. doi:10.1109/CFEC.2018.8358724 (2018).
18. M. Naas, L. Lemarchand, J. Boukhobza, P. Raipin, A graph partitioning-based heuristic for runtime iot data placement strategies in a fog infrastructure, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC, Pau, France. pp. 767–774. doi:10.1145/3167132.3167217 (2018).
19. A. Aral, T. Ovatman, A decentralized replica placement algorithm for edge computing, IEEE Transactions on Network and Service Management 15 (2) 516–529. doi:10.1109/TNSM.2017.2788945 (2018).
20. U. Ozeer, X. Etchevers, L. Letondeur, F.-G. Ottogalli, G. Sala¨un, J.-M. Vincent, Resilience of stateful iot applications in a dynamic fog environment, in: Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, pp. 332–341 (2018).
21. R. Mayer, H. Gupta, E. Saurez, U. Ramachandran, Fogstore: Toward a distributed data store for fog computing. arXiv:1709.07558 (2017).
22. S. Goel, R. Buyya, Data replication strategies in wide-area distributed systems, in: Enterprise service computing: from concept to deployment, pp. 211–241 (2007).
23. J. Xie, C. Qian, D. Guo, X. Li, S. Shi, H. Chen, Efficient data placement and retrieval services in edge computing, in: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 1029–1039. doi:10.1109/ICDCS.2019.00106 (2019).
24. T. Huang, W. Lin, Y. Li, L. He, S. Peng, A latency-aware multiple data replicas placement strategy for fog computing, Journal of Signal Processing Systems 91 (10) 1191–1204 (2019).
25. S. R. Balachandar, K. Kannan, A new heuristic approach for the large-scale generalized assignment problem, Int J Math Comput Phys Elect Comput Eng 3 969–974 (2009).
26. H. Gupta, A. Vahid Dastjerdi, S. Ghosh, R. Buyya, iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments, ArXiv e-print (2016).
27. V. Verter, Uncapacitated and Capacitated Facility Location Problems (2011).
28. X. Wu, Data sets replicas placements strategy from cost-effective view in the cloud, Scientific Programming (2016).
29. D. Bermbach, J. Kuhlenkamp, Consistency in distributed storage systems, in: V. Gramoli, R. Guerraoui (Eds.), Networked Systems (2013).
30. B. Kemme, G. Ramalingam, A. Schiper, M. Shapiro, K. Vaswani, Consistency in distributed systems (dagstuhl seminar 13081), in: Dagstuhl Reports, Vol. 3, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2013).
31. S. V. Adve, K. Gharachorloo, Shared memory consistency models: a tutorial, Computer 29 (12) (1996).
32. M. Shapiro, A. Bieniusa, N. Pregui¸ca, V. Balegas, C. Meiklejohn, Just-right consistency: reconciling availability and safety. arXiv:1801.06340 (2018).
33. D. Bermbach, S. Tai, Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior (2011).
34. V. Marianov, D. Serra, Median Problems in Networks. " Foundations of location analysis. Springer, Boston, MA. 39-59 (2011).
35. O. Kariv, S. L. Hakimi, An algorithmic approach to network location problems. i: The p-centers, SIAM Journal on Applied Mathematics 37 (3) 513–538 (1979).
36. R. W. Floyd. 1962. Algorithm 97: Shortest path. Commun. ACM 5, 6 , 345. doi:10.1145/367766.368168 (1962).
37. P. Avella, A. Sassano, On the p-median polytope, Mathematical Programming 89 (3) 395–411 (2001).
38. O. Kariv, S. L. Hakimi. "An Algorithmic Approach to Network Location Problems. II: The p-Medians."SIAM Journal on Applied Mathematics 37 (3) 539–560 (1979).
39. Ibm ilog cplex optimization toolkit, http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud, accessed: 2017-09-20.
40. M. Rabinovich, E. D. Lazowska, An efficient and highly available read-one write-all protocol for replicated data management, in: [1993] Proceedings of the Second International Conference on Parallel and Distributed Information Systems, pp. 56–65. doi:10.1109/PDIS.1993.253072 (1993)