International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 15(2023) pp. 97-108
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Submitted: 25 Nov., 2022; Accepted: 8 April, 2023; Publish: 9 June, 2023

Stable Matching based Resource Allocation for Large-Scale Fog Computing

Khaled Zeraoulia[1], **Ayoub Hammal**[1], **Mehdi Lerari**[1] **and Youcef Hammal**[1]

[1]LSI Laboratory, Faculty of Computer Science,
USTHB University, Algiers, Algeria
{*zeraoulia, yhammal*}*@usthb.dz*, {*ayoub.hammal, mehdi.lerari*}*@etu.usthb.dz*

Abstract: **Fog computing is used to expand Cloud computing services at the network edge. A key step in the process of improving Fog services is the management of Fog resources. Fog resources are usually dynamic, heterogeneous, latency-constrained, and bandwidth-constrained as compared to Cloud resources. Improving the performance of Fog systems requires addressing Fog resource management. Also, in large-scale Fog networks, resource management still faces more difficulties despite these improvements. Towards this end, this paper proposes a new resource allocation technique in a large-scale Fog network to optimally serve the service requests generated by a set of IoT objects. In this technique, we exploit the proven stable and efficient Gale Shapley matching algorithm in large-scale Fog computing networks. IFogSIM is used to demonstrate the effectiveness of this approach.**

Keywords: Resources Allocation, IoT, Fog Computing, Cloud Computing.

I. Introduction

The growth of the Internet of Things (IoT) [1, 2, 3, 4, 5] is increasingly being driven by this vision of a more comfortable everyday living, an effective economy and governance, safe highway traffic, environmentally friendly energy supply, and a healthier way of life. Impressive predictions have been made regarding the effects of IoT on the internet and economy; some estimate that by 2025, there will be up to 100 billion linked IoT objects and an impact of more than 11 billion on the global economy. However, the exponential increase in the number of connected devices and the massive amount of data they produce have brought to light some issues with Cloud computing, specifically those with its centralized architecture. Moreover, data created by IoT or user devices in a smart city is typically sent to clouds located far away from the user devices for processing and archiving. Because it is expected to result in increased communication latencies when billions of devices are connected to the Internet, this computing model is not appropriate for the future [6].
An alternative processing paradigm to the Cloud model is suggested to address the workload of users and IoT as well as allow for the closer placement of computing resources. This new emerging processing approach is called Fog Computing [7]. This emerging concept creates an architecture between IoT devices and the Cloud by offering data and resource management, processing, and storage capabilities close to the IoT devices [8, 9, 10, 11, 12]. In order to satisfy the demands of network-sensitive applications, the current research suggests various ways to improve resource management, network performance, reduce excessive surcharges, and reduce latency [7, 8, 13, 14]. However, despite the significant advantages provided by these solutions, the paradigm has limitations such as: congestion-constrained, IoT-Fog workload allocation problem. It is worth noting that these limitations are particularly evident in large-scale Fog environments. This makes thus resource management in large-scale Fog environments extremely challenging. Moreover, a resource allocation strategy for this environment becomes a difficult task that requires additional effort. To enhance resource allocation techniques in such systems, clustering and profiling Fog nodes are investigated. In part, this is due to the fact that most IoT requests and data processing can be carried out within clusters [15]. Profiling Fog nodes are also easily deployable on large-scale systems that require more heterogeneous and massive processing. Therefore, this paper proposes a new resource allocation technique in a large Fog environment, aiming to efficiently serve massive data generated by IoT devices. The proven stable and efficient Gale-Shapley matching algorithm [16, 17] is used in this proposal to demonstrate that our resource allocation technique is capable of dealing with large-scale Fog networks.
The remainder of the paper is organized as follows: in Section 2, we define the key concepts of the problem and present the state of the art in this field. We will then describe our technique, the execution scenarios, and the different exchanges between the entities in our environment in a third section. The fourth section of the paper includes an implementation of this technique within the simulation environment IfogSim, as well as a discussion and analysis of the results. Section 5 concludes with some concluding remarks and perspectives.

II. Backgrounds

Firstly, this section explains the relationship between Fog computing and the Internet of Things. After discussing different Cloud migration strategies, we discuss Fog migration strategies.

A. *Internet of Things (IoT)*

Internet of Things (IoT) describes a network of physical devices - often heterogeneous - interconnected, and whose main function is to collect, exchange, and interact with external data [5]. An intelligent infrastructure can self-organize, share information optimally, and react to environmental changes [4, 18, 19]. The computing power and the energy consumption of IoT objects in this network are limited. However, manufacturers in the field of communications and networks are concentrating their efforts on the production of sensors and IoT devices with different functionalities in order to meet the needs of various industries. In recent years, smart objects have become more accessible. Thus, IoT devices are used in several domains, such as security, surveillance systems, smart homes, autonomous car systems, and medical monitoring. IoT networks are composed of four layers, namely: sensors, networks, management, and applications. Layers utilize a variety of technologies and provide a variety of services [5]. Internet of Things (IoT) devices produce a large volume of data, challenging the traditional Cloud computing infrastructure. With the increase in connected devices, cyber-criminal attacks are becoming more diverse. The IoT network architecture may be negatively affected by the injection of misinformation into an IoT network.
Additionally, most time-sensitive IoT applications require adaptive behavior. The use of Cloud servers to process these IoT applications may not be efficient [20, 18, 19].

B. *Fog Computing*

The Fog Computing approach has become one of the most widely used methods for improving IoT-Cloud communications. It has been defined as a promising approach to address the problem of managing the large data bandwidth requirements and fast response times of IoT devices [20, 21]. According to CISCO [9], Fog computing describes: ”a highly virtualized platform that provides computing, storage and networking services between end-devices and data centers of traditional Cloud computing, usually, but not exclusively, located at the edge of the network”. The OpenFog Consortium [12] also defines it as: ”Across the Cloud-to-IoT, it delivers computing, storage, control, and networking functions closer to IoT devices.” The term also refers to a distributed hardware and application infrastructure that aims to store and process data from different connected devices in order to replace the Cloud for certain processing. The main idea of Fog computing is the instrumentalization of the various equipment which consists of the network nodes (routers, switches, gateways, etc.), a distributed data processing and storage center which is both intermediate to the Cloud and at the same time near the ends of the network. The ability to create a layer near the data being processed reduces transfers entering and leaving the Cloud, resulting in a reduction in latency and, therefore, in the time it takes to perform different processing operations.

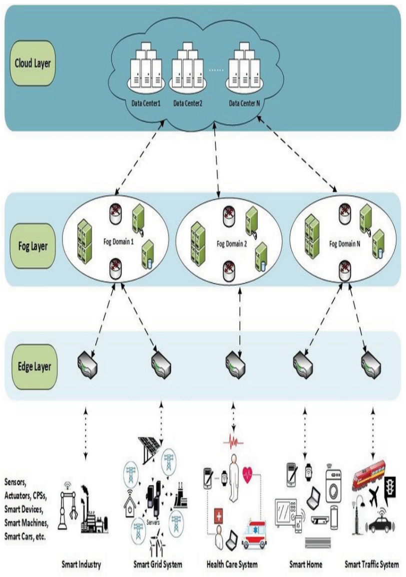

Figure. 1: Architecture of Fog Computing

1) Architecture

Most of the literature follows the same architectural scheme of Fog networks(see Figure 1), namely the three-layer model [11] is presented as follows :

- IoT layer: This layer refers to all devices at the network end, such as smart vehicles, smart-phones, drones, sensors, etc. It is mainly composed of IoT devices, whose role is the collection and transmission of data to the upper layer for storage or processing.
- Fog layer: This layer consists of Fog nodes, which are defined as ”physical or logical elements that implement Fog computing services” [11]. The set of nodes constitutes a distributed processing and storage system connected to both lower and upper layers through gateway nodes.
- Cloud Layer: This layer represents a centralized Cloud infrastructure. It is composed of high level hardware resources and provides different services. Unlike a classic Cloud architecture, some processing and services that are offloaded from the Cloud layer to the Fog layer in order to balance the workload and increase efficiency and reliability.

2) Advantages

Fog Computing has many benefits which may be highlighted as follows:

- Security and Privacy: In Fog networks, privacy and security are both crucial. Security is the defense against threats or danger, whereas privacy is the ability to control how your information is viewed and used.

- Productivity: With the appropriate tools, developers can easily create these Fog apps. It can be deployed at any time after development is complete.
- Cognition: This comes from the fact that the Fog infrastructure is aware of the needs and requirements of users. It thus distributes resources more finely according to each user, unlike the Cloud.
- Agility: It refers to a capacity for quick adaptation to technical changes.
- Latency: Because of its proximity to end users, the Fog has the ability to handle applications that require short and stable latencies and avoiding difficulties associated with centralized systems.
- efficiency: Resulting from the integration of the many devices in the network (with their processing and storage capacity), this concept enhances the system's capacity and overall efficiency.

3) *Challenges in Fog computing environments*

Fog computing has many advantages, but it remains a relatively new paradigm that must deal with energy management, heterogeneity, security, and Fog resource management. The latest is a crucial prerequisite in the enhancement process of Fog services. Fog resources are frequently dynamic, heterogeneous, latency-constrained, and bandwidth-limited, compared to Cloud resources.

4) *Migration in Fog Computing environments*

As aforementioned, addressing Fog resource management is essential to optimizing the performance of large-scale Fog systems. According to [22], managing resources on a large scale while providing performance isolation (by using virtual machine or container) and efficient use of the underlying hardware is a key challenge for any resource management software. Virtual Machines (VMs) are required to effectively manage multiple resources while providing differentiated quality of service to VM groups. We can consider them the key issues in building Fog-scale resource management systems.
Virtualization introduces a layer of software abstraction between the hardware and the operating system or applications running on it. By separating logical resources from underlying physical resources, virtualization enables the flexible assignment of workloads between physical machines. Virtual instance migration (Virtual Machine or Container) is considered as the process of copying and moving the state of the latter from one physical host to another [23]. Virtual instance migration plays an essential role in Fog environments since it guarantees the continuity of services, regardless of the requirements for mobility expressed by objects connected to this environment.
There are two major software virtualization techniques used in service-oriented architectures: virtual machines and containers. Their major differences are their scalability and portability [23, 24, 25, 26] (see Figure 2).

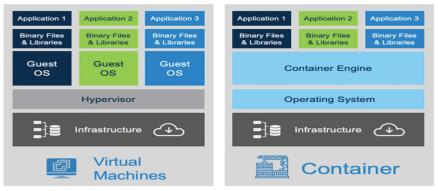

Figure. 2: Architecture of virtual machine and of container

C. *Related Work*

We present in this subsection a taxonomy of similar works focusing on the resource management of Fog nodes, as well as the various challenges encountered.
There are specific characteristics of Fog computing that make it suitable for applications, which require low latency, mobility, real-time analysis, and interaction with the Cloud [27]. In order to ensure a reliable Fog computing environment, it is imperative to optimize the utilization of deployed resources. This environment requires resource-constrained Fog nodes to operate in a highly dynamic environment. As a result, they require adaptive resource and task management. Literature describes Fog resource management methods as "latency" or "workload enhancement" techniques [27, 6, 2, 7, 8]. In order to improve their proposals, most of these methods use heuristics or meta-heuristics, model-based techniques (e.g., approximation, Markov), Machine Learning, Deep Learning, and Game-Theory.
According to [28, 29], Fog resource management provides load balancing, dynamic provisioning, and auto-scaling services via efficient node deployment techniques and virtualization.
Fog environments also face a complex problem of resource management. Consequently, it cannot be considered as a single issue. To simplify the resource management problem, we proposed to view it along six axes: application placement, task scheduling, task offloading, load balancing, resource allocation, and resource supply (see Figure 3).
Application placement: A problem of application placement refers to finding a way to associate an IoT service with a Fog node while meeting Quality of Service (QoS) requirements. More formally let S be an IoT service with QoS requirements Q, and let N be the set of Fog nodes. A solution to the application placement problem consists in associating to the service S a Fog node N_i of N satisfying the QoS requirements Q, while optimizing a set of objective functions O. Multi-valued relationships are possible, so an IoT service can be placed on one or more nodes, and a node can host a variety

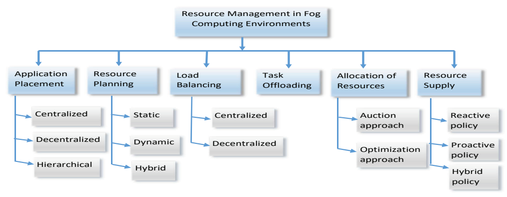

Figure. 3: Resource management domains and their related approaches

of services. Approaches based on Broker management can be organized into 3 categories which are: centralized, decentralized and hierarchical approach.

- In the centralized approach, the broker needs to have a global view of the entire Fog environment in order to make optimization decisions for the entire system. This approach does not guarantee effective optimization due to the difficulty of obtaining all the information from all the entities of the Fog, as well as poor fault tolerance due to its architecture centralized.
- While the decentralized approach, it consists of a set of local optimization which makes it very interesting in terms of scalability.
- As for the hierarchical approach, the idea is to link and coordinate the various local managers so that they can collaborate with each other and thus benefit from the advantages of the previous two approaches.

Resource planning: In Fog environments, an IoT service can be placed on multiple nodes, and each service can be divided into several sub-services. Let a set of sub-services $S = \{S1, ..., Sn\}$ (with different requirements in terms of QoS) to place on a set of nodes $N = \{N1, ..., Nm\}$ (having different processing capacities). Resource planning consists in finding an optimal assignment of the different sub-services S_i to the different nodes N_j according to the objectives considered by the scheduling policy (for example, minimizing the execution time).
The following approaches are used in resource scheduling:

- Static: This approach allocates nodes to the various sub-services in a static manner, that is, the decision is already made even before the request is submitted, which implies prior knowledge of all the relevant information.
- Dynamic: unlike the previous one, the allocation process is not fixed in advance, decisions are made once the requests have been formulated.
- Hybrid: it consists of a combination of the two previous approaches in order to respond to the diversity of types of application.

Task Offloading: In task offloading, tasks that cannot be executed locally due to lack of resources are transferred to nodes with the necessary capabilities. Considering IoT devices have limited hardware and energy resources, resource-intensive tasks such as graphics calculations, augmented reality, etc., often require external entities, such as the Fog or the Cloud.
There are three main components to task offloading:

- IoT devices: whose role is to specify how applications should be partitioned, then determine which part should be run locally, and which part should be discharged.
- Communication links: they make it possible to ensure the transfer of tasks, and therefore the quality of the transfers depends on the physical capacities of the links.
- Fog nodes: these have a lower capacity than the Cloud, but greater than IoT devices.

Task offloading may also occur to provide load balancing, minimize latency, power efficiency, etc.
Load balancing: Load balancing [30] consists in distributing the excessive loads on the different Fog nodes according to a certain strategy, in order to ensure that no Fog node is overloaded or underloaded, improving thereby the overall performance of the system. However, in reality the load balancing mechanisms encounter many challenges, mainly the problem of latency which is due to the continuous migration of different processes. Balancing strategies are implemented according to either a centralized or a decentralized architecture:

- The centralized approach relies on a central controller, thus requiring global and real-time knowledge of the status of the various nodes. This approach is therefore difficult to implement due to the difficulty of continuously knowing the states of the various nodes of the system, besides its low tolerance to faults of its centralized architecture.
- In the decentralized approach, a decentralized controller coordinates the various local controllers, ensuring greater scalability.

Allocation of resources: The resource allocation problem in Fog environments can be considered as a double matching (mapping) problem because Cloud Servers and Fog Nodes are paired for users whereas the user and Fog Nodes are paired for Cloud Servers. In other words, the users must take into consideration the relationship between Fog nodes and Cloud servers, and Cloud servers must take into consideration the relationship between nodes and users. Techniques for allocating resources can be categorized into two categories:

- ”auction-based” technique: clients submit their requests for resources to the broker with a request pricing system, and the resources will be allocated at most bidding, using auction mechanisms calculated using various mathematical techniques.

- ”optimization-Based” technique: it consists of finding the optimal combination (Cloud servers, Fog Node, user) for each user by performing optimizations of objectives function, such as minimizing response time, maximizing QoS, etc.

Resource supply: Since the workloads of various applications fluctuate constantly, the risk of resources over-provisioning or under-provisioning is high. The over-provisioning problem consists in allocating an amount of resources greater than the actual workload of an application. (And vice-versa for the problem of under supply). Dynamic resource provisioning is essential to allowing continuous adaptation to workloads in a constantly changing environment. Dynamic sourcing policies are categorized into 3 policy kinds:

- Reactive policy: it consists of responding only to received requests, with no attempt to predict future requests.
- Proactive policy: it is based on prediction techniques to anticipate future changes in workloads and adapt decisions, accordingly.
- Hybrid policy: it therefore adopts the two previous policies, the reactive policy is often used to supply resources to a new request that arrives in the system, while the proactive policy makes it possible to anticipate future changes in demand.

1) Proposed architectures

The different resource management approaches in Fog environments have been classified according to their 3-type architectures [17]:

- Data flow architectures: These types of architectures are based on the direction of workloads transfer, for example, workloads can be transferred from the user to the Fog node or from the servers Cloud to nodes.
- Control architectures: These architectures are based on the way the resources are managed at the system level, for example, a central controller or algorithm can be used to manage a set of nodes.
- Tenancy architecture: this architecture is based on the ability of different nodes to host several applications, for example, one or more applications can run on a Fog node.

III. Stable Maching Based Resources Allocation in large scale Fog Environment (SMRA)

According to related research on improving and optimizing resource management, resource allocation is a key issue that needs to be investigated to achieve optimal resource management. Over the past few years, academia and industry have paid considerable attention to the objective of optimizing resource allocation. The development of an efficient resource allocation management model is therefore crucial for an efficient large-scale Fog infrastructure. The purpose of this paper is to develop a dynamic, effective and scalable resource planning model to enable IoT devices to allocate different resource demands to different Fog nodes based on their resource demands. This problem was inspired by an article [31] that addressed the same issue, with the authors interested in providing resources in a Cloud environment. This process can be divided into three principal steps, namely :

1. Identification of affected nodes (i.e. under-utilized or over-utilized nodes).
2. Selection of VMs to migrate.
3. Reassigning VMs to underutilized nodes.

We are particularly interested in the third step, where the authors propose a matching problem for allocating VMs to the correct nodes. Our paper proposes, in a similar way, a new resource allocation technique for large-scale Fog networks to better serve queries generated by IoT devices. This technique, called ”‘pure Stable Matching based Resources Allocation (SMRA)”’, is based on the algorithm of Gale-Shapley [17, 16]. It is detailed in what follows.
Hypotheses:
in order to focus on the issue of matching service requests to Fog nodes, we first have to formulate some assumptions about the physical topology of the large scale Fog infrastructure.

- The topology is static during operation: once the solution is implemented, the topology does not undergo any change.
- The topology follows a matrix organization: Fog nodes are organized in well-defined, distinct levels with the same number of nodes in each level.
- The topology is interlaced between the levels: each cluster Fog node is physically connected to all higher level nodes.
- Each node knows all its parent and child nodes.(i.e. respectively upper level nodes and lower level nodes)
- In this model, we have at least three types of requests which are:
  - The request, which corresponds to a request for services issued by the IoT devices, it contains all the details of the request.
  - the outcome request, which is generated after a service request is completed, is issued to the sender IoT object.
  - The token request, which represents the token circulating between the different gateway nodes following round-robin policy.

1) SMRA architecture

Our allocation technique SMRA is divided into four interrelated features, which we describe below.:

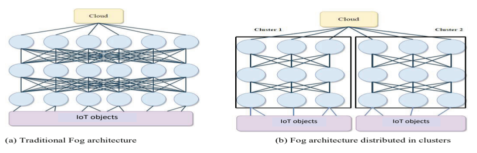
Cloud
IoT objects
(a) Traditional Fog architecture
Cloud
Cluster 1
Cluster 2
IoT objects
IoT objects
(b) Fog architecture distributed in clusters

Figure. 4: Fog architectures

- Clustering: by grouping the Fog nodes into groups (Cluster) based on their geographical proximity.
- Profiling: by assigning different profiles or roles to the different nodes of a cluster.
- Batch processing: requests are grouped and processed in batches according to their arrival orders.
- Matching: the process of associating and redirecting requests to the adequate Fog nodes.

2) Clustering

As mentioned before, a classic Fog infrastructure consists of 3 overlapping layers (see Figure 4(a)):

- The IoT layer, which represents the set of IoT objects that make service requests.
- The Fog layer, which represents all the Fog nodes located at the intermediary between the IoT objects and the Cloud.
- The Cloud layer, which represents the traditional Cloud infrastructure.

SMRA starts by vertically splitting up the Fog layer of the infrastructure into a set of clusters (see Figure 4(b)), i.e. each cluster brings together a set of interconnected Fog nodes.

3) Profiling

Once the infrastructure is distributed into clusters, SMRA assigns profiles to the nodes constituting each cluster (see Figure 5) by using two profiles defined as follows:

- The Gateway Node profile (in red), which is adopted by the nodes belonging to the level that is directly connected to the IoT layer.
- The Node Fog profile (in blue): which is assigned to the nodes of the remaining intermediate levels between the gateway node layer and the Cloud.

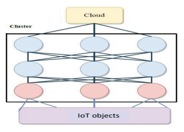

Figure. 5: Diagram representing the profiles of the nodes within a cluster

4) Scenario Execution

Since the clusters are independent, generalizing this operation to all clusters defined by the infrastructure is thus possible since it indistinguishably applies to anyone of them. Different IoT devices send service requests to the cluster gateway nodes of the cluster they belong to. These nodes keep then their requests in a queue, waiting for them to be matched and thereafter sent them to their selected destination.
We define a token circulating from one gateway node to another according to the round-robin policy; Whenever any Gateway node receives a token, it matches requests in its queues with Fog nodes in its cluster. Thereafter, it sends each request to the appropriate Fog node, returns the token to the next gateway node, and so on (as depicted by the flowchart of Figure 6).

5) Algorithmic description of the scenario

The SMRA is designed as an event paradigm, i.e. an algorithm based on events, where each event is associated with a procedure called ”Routine”. SMRA has 2 main routines which are: - The Routine associated with the gateway nodes - The Routine associated with the Fog node
Routine associated with gateway nodes:
This routine is executed at the level of the gateway nodes each time a request is received. The pseudo-code of this routine is as follows:
Matching procedure:
For the implementation of this procedure, we opted for the use of the Gale-Shapley algorithm [17], which is an algorithm designed to solve the problem of stable marriages. It has interesting properties [17] which are:

- Good performance due to its quadratic complexity (O(n2)).
- Convergence: any request will be associated with a node at the end of the execution.

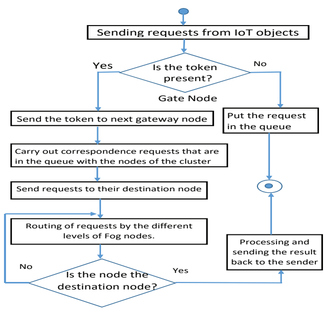
Sending requests from IoT objects
Yes
Is the token present?
No
Gate Node
Send the token to next gateway node
Put the request in the queue
Carry out correspondence requests that are in the queue with the nodes of the cluster
Send requests to their destination node
Routing of requests by the different levels of Fog nodes.
Processing and sending the result back to the sender
No
Is the node the destination node?
Yes

Figure. 6: SMRA Execution Scenario Flowchart

- The pairs (request, node) resulting from this algorithm are stable.
- The resulting configuration is optimal compared to all other stable solutions

This algorithm requires the definition of a preference relation, called "Adequacy Relation" or AR, which is associated with each request and each node: To achieve this objective, we define a distance between the request and the node, which is calculated as follows:

Symbol	Definition
C_u	$MIPS$ (million of instructions per second) used
C_{dem}	$MIPS$ required by the service to be executed
C_fd	total amount of the node $MIPS$
C_r	$MIPS$ Available

$$Distance = \begin{cases} \frac{C_u + C_{dem}}{C_f d} & \text{if } C_r \geq C_{dem} \\ -1 & \text{else} \end{cases}$$

The matching relationship is defined by using the minimum distance between the node and the request. That is, let $D = \{D_1, D_2, ..., D_n\}$ be a set of demands, and let $N = \{N_1, N_2, .., N_n\}$ be a set Opf nodes. We say that the node N_j is the best suited to the request D_i if and only if: $Distance(D_i, N_j) = Min(Distance(D_i, N_m)), \forall m \in \{1, .., n\}$.

Routine associated with the Fog node:
This routine is executed at Fog nodes each time a request is received.

IV. Implementation and experimental results

A. Introduction

The previous section presented all the theory behind our resource planning solution for managing resources in the Fog environment. To evaluate performance and results, we need tools that allow us to quantify them. In this section, we first provide an overview of the different tools used to simulate

Algorithm 1 Routine associated with gateway nodes

```
function HANDLE(Input $Data$: ListRequests, ListNodes)
    if The event received is of the token type then    ▷ call the Matching procedure
        Matching(fileRequests, listNodes);    ▷ then perform the request routing
        for all request $D_i \in fileRequests$ do
            if listParents.contains($D_i$.destination) then
                send(request, $D_i$ .destination);
            else
                send(request, DefaultParentNode);    ▷ update default node for load balancing between links
                DefaultParentNode ← nextParentNode()
            end if
        end for
    else
        if The event received is of type result then    ▷ send the result to the concerned IoT device
            send(result, result.destination);
        else    ▷ the event is therefore of the service request type
            push(fileRequests, Request);
        end if
    end if
end function
```

Algorithm 2 Matching procedure

```
function HANDLE(Input $Data$: ListRequests, ListNodes)    ▷ reset all requests to null destination.
    for all Event $D_i \in fileRequests$ do
        $D_i.destination \leftarrow null$;
    end for
    while ∃ an unassigned request $d$ that can be offered to a node do
        $n$ ← the node the best suited to $d$ such that $d \notin node.rejectedrequests$;
        if n = null then    ▷ if no node can process the request, it is delegated to the Cloud.
            $d.destination \leftarrow Cloud$;
            sendToCloud(d);
        else
            if n is free then
                d.destination ← n;
            else    ▷ case n is not free and d is more adequate than the request associated with n
                if $distance(n, n.request) \geq distance(n, d)$ then    ▷ reset to null the destination field of the request associated with n so that it can be handled again.
                    $n.request.destination \leftarrow null$;    ▷ add old request attached to n into the list of requests rejected by the node n so that it cannot be offered again to n
                    n.rejectedrequests.add(n.request);    ▷ the new request associated with n is replaced by d
                    $n.request \leftarrow d$;
                    $d.destination \leftarrow n$;
                else    ▷ request d does not fit with the node n, so it is rejected by n.
                    n.RejectedRequests.add(d);
                end if
            end if
        end if
    end while
end function
```

Algorithm 3 Routine associated with Fog nodes

```
function HANDLE(Input $Data$: ListRequests, ListNodes)
    if Event received is of type request then
        if request.destination is the node itself then    ▷ the destination of the result is the request source .
            $result \leftarrow execute(request)$;    ▷ thereafter route the result to its destination.
            if listChildren.contains(result.destination) then
                send(result, result.destination);
            else
                send(result, DefaultChildNode);
                DefaultChildNode ← nextChildNode();
            end if
        else    ▷ the node in question is not the destination.
            if listParents.contains(request.destination) then
                send(request, request.destination);
            else
                send(request,DefaultParentNode);
                DefaultParentNode ← nextParentNode();
            end if
        end if
    else    ▷ the request received is therefore a result request.
        if listChildren.contains(result.destination) then
            send(result, result.destination);
        else
            send(result, DefaultChildNode);
            $DefaultChildNode \leftarrow nextChildNode()$;
        end if
    end if
end function
```

this solution and the various elements added to it. As a final step, we compare the results with those of the various classic management policies: "First fit", "Best fit", and "Worst fit".

B. Development Tool

In order to be able to simulate the proposed solution, we opted for the "IFogSim" simulator (see Figure 7). The latter operates according to the event paradigm, which is in line with our solution. With IFogSim, an application developed exclusively in Java, we can measure the impact of management on technical factors by analyzing data, energy consumption, lead times and network status.

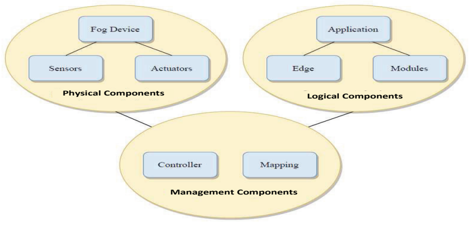

Figure. 7: Diagram representing interactions of the different components of IFogSim

C. Design, development and Tests

In order to implement the proposed technique, we add new classes necessary for the implementation of our solution, but we also proceed to the modification of some main classes of the IFogSim in order to adapt it to our needs (see Figure 8). The manipulations carried out to implement this solution are listed below:

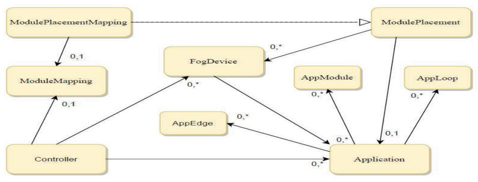

Figure. 8: Diagram representing relationships between the main classes of IFogSim

1) Creation of a ClusterFogDevice class:

This class represents any Fog node of the cluster. It extends the predefined FogDevice class and additionally includes the following attributes:
— parentsIds: which is a list that contains the identifiers of the nodes at the top level.
— isNorthLinkBusyById: which is a list of booleans where each element indicates whether the upper link in question is

busy.
— northTupleQueues: which is a list of tuple queues that associates each upper link with a queue of tuples where the tuples (related to tasks required by sensors) to be sent on this link will be stored if the latter is busy. Each tuple is generated by a sensor to require some task.
Additionally, the processTupleArrival method, which is the method executed by the node each time a tuple arrives, has been redefined to implement the routine associated with a Fog node that is described in the design.

2) *Creation of a GWFogDevice class:*

This class represents a Fog gateway node, i.e. the node connected directly to sensors and actuators. This class also extends the class FogDevice, and adds the following attributes:
— waitingQueue: which is a list of tuples used to store requests not yet assigned.
— tupleToMatchedDevice: is a list containing the tuples not assigned to a node.
— matchedTupleList: which is a list comprising the tuples assigned to their respective nodes.
— gwDevices: is a list of GWFogDevice, containing the set of gateway nodes.
— isNorthLinkBusyById: which represents the same as in the class.
— northTupleQueues: it also represents the same as mentioned in the ClusterFogDevice class.
— clusterFogDevicesIds: which represents the list of all Fog nodes in the cluster.

3) *Creation of a MatchedTuple class:*

This class describes the tuples once assigned to their respective nodes. It represents an enrichment of the Tuple class by the following attributes: — destinationFogDeviceId: which contains the identifier of the destination node. — destModuleMips: which describes the processing requirements of the destination module. The same procedure was followed to implement concurrent policies.

D. *Results and performance evaluation:*

We evaluate the performance of our solution by measuring the time taken to complete the task in order to determine the effectiveness and the extent of gains made through our resource planning model. So, to evaluate the performance of this solution, we measure the execution time, application execution time, and tuple execution time. Then, we compare the results with those of the classical models implementing the “FirstFit”, “BestFit”, “WorstFit” strategies.
As a result, Fog nodes and generated requests have a certain level of heterogeneity, allowing us to better study the ability of our algorithm to match a variety of requests to nodes with different hardware capabilities.

1) *Vertical scalability*

This part measures the performance of the model based on the variation of the Fog matrix levels. In this experiment, we fix the number of nodes per level at 5 nodes.
Figures 9 and 10 show, respectively, the variation in the average execution time of a tuple (representing a service request)

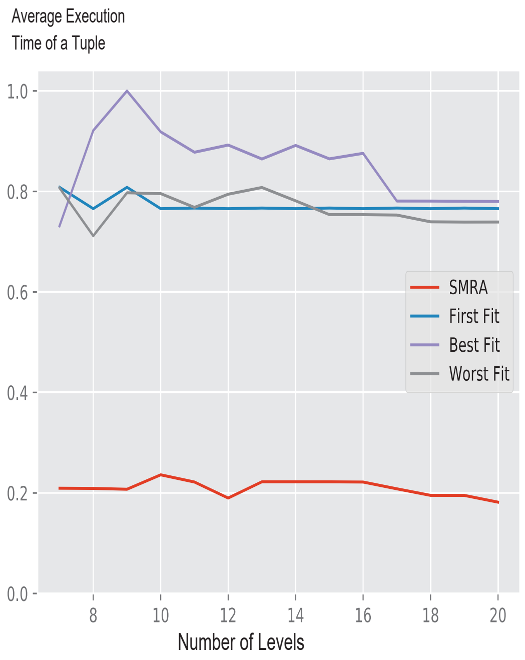

Figure. 9: Average execution time of a tuple according to the number of levels

and the execution time of a complete application according to the number of levels in the architecture.
We see a clear reduction in the tuple processing time of about 75 per cent, compared to the other simulation algorithms. This reduction stabilizes beyond the threshold of 18 levels, This metric is no longer affected by the number of levels. It is also possible to see the impact of reducing the response time of individual tuples on the execution time of an application.

2) *Horizontal scalability*

Here, we evaluate the model performance in relation to the variation in the number of nodes per level of the previously defined Fog matrix. Since each gateway node is connected to a set of IoT objects, the number of tuples generated by IoT objects depends on the number of nodes per level. In this way, increasing the number of gateway nodes will also result in an increase in requests. We set the complexity with relation to ten levels.
Figures 11 and 12 represent, respectively, the variation in execution time of a tuple by the number of nodes per level and the variation in execution time of an application by the number of nodes per level.
Figure 11 shows a linearly increasing execution time for classical strategies, unlike the proposed method which seems to follow a logarithmic trend. For classical strategies, increasing the number of nodes per level significantly impacts the execution time of an application, contrary to the proposed strategy. It seems that the correlation is less significant.
With the ”Best Fit” strategy, the number of nodes per level increases linearly with the execution time of a tuple. In the case of the ”First Fit” strategy, in levels with more than 19 nodes, the execution time stabilizes. ”Worst Fit” strategy shows an increase followed by a phase decrease from 17 nodes per level.

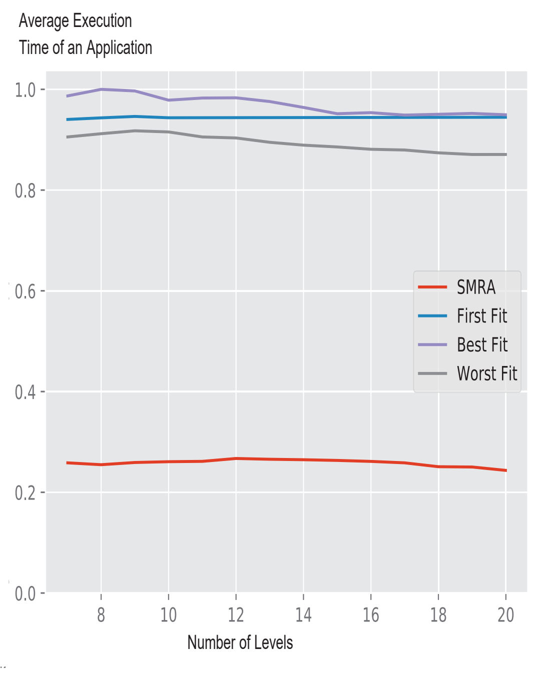

Figure. 10: Average execution time of an application according to the number of levels

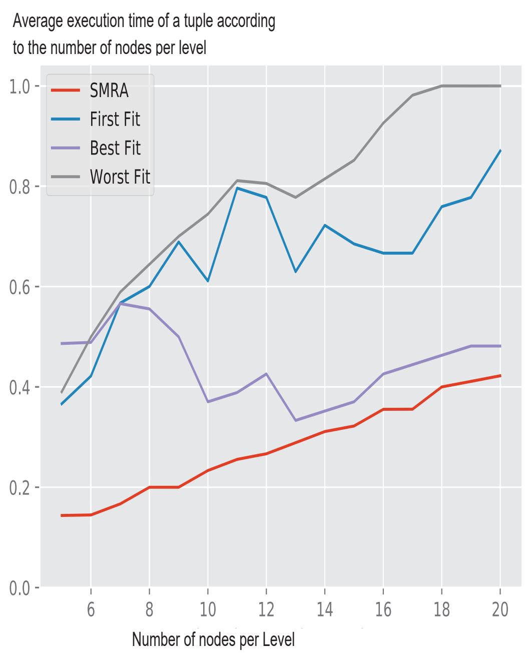

Figure. 11: Average execution time of a tuple based on the number of nodes per level

The proposed strategy also seems to have better results and a more stable execution time, which is a bullish sign.
Clustering schemes are derived from the results of our scalability experiments. As the number of gateway nodes exceeds the number of levels in the Fog cluster, the performance of our previous two experiments drops.

3) Study based on Throughput Request

Our model will be evaluated based on its response time as a function of the delay between each request. In a network, the smaller the delay between requests, the greater the load and congestion. We fix in this experiment the number of levels at 7 levels and the number of nodes per level at 5 nodes (note that $5 \leq 7$).
Figures 12 and 13 represent, respectively, the variation of the execution time of a tuple and the execution time of an application according to the transmission delay between consecutive tuples. According to the classic strategies, the execution time of an application is stable and close to the maximum value, which means that the decrease in workload does not affect the execution time. Conversely, the proposed strategy shows better performance in terms of delay as well as a reactivity to variations in transmission delay.
With our technique, we were able to reduce the execution time of service requests, indicating better resource management and distribution throughout the entire cluster.

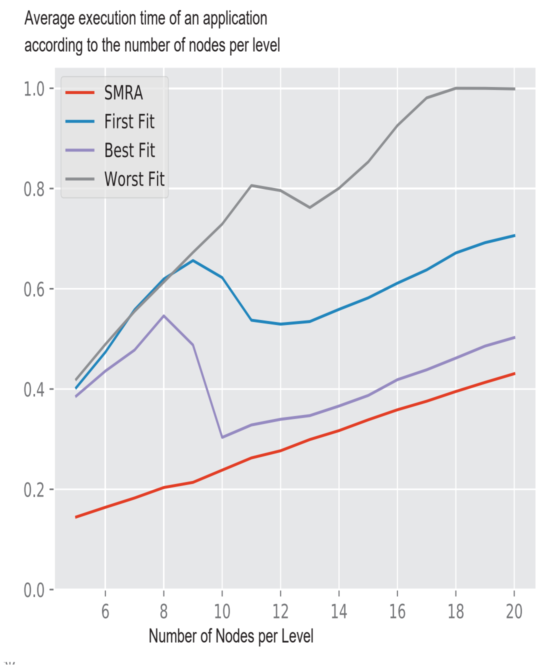

Figure. 12: Average execution time of application based on the number of nodes per level

V. Conclusion and future work

In this paper, a decentralized algorithm is proposed for the fair distribution of batch service requests among Fog nodes.
With our solution, real-time applications with high quality service requirements can expect responsiveness and shorter response times. Additionally, this solution ensures an eq-

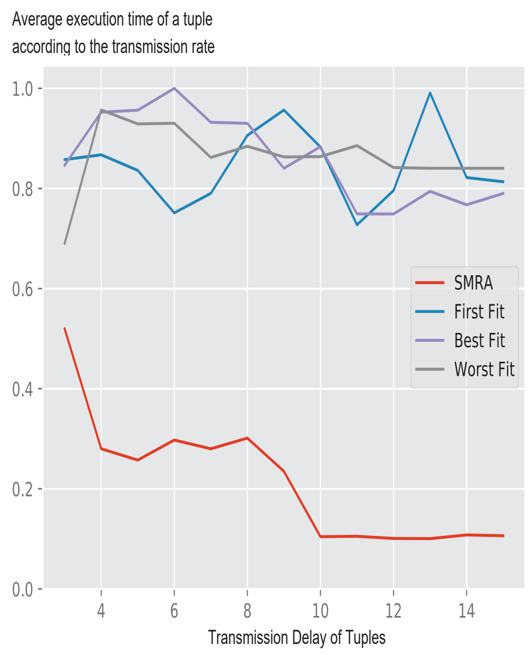

Figure. 13: Average execution time of a tuple according to the transmission rate

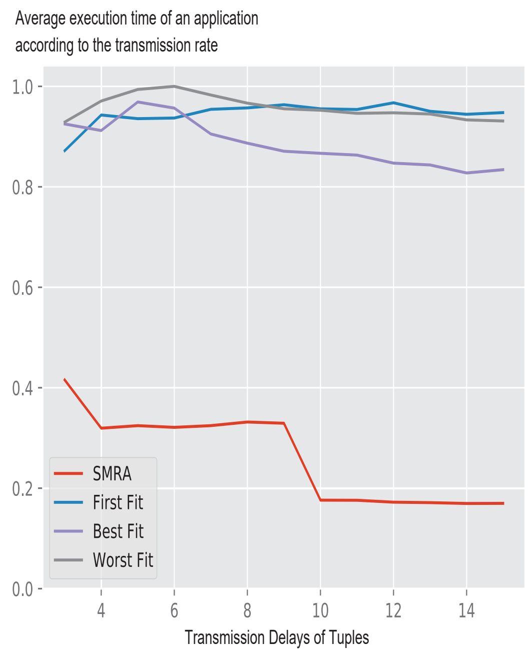

Figure. 14: Average execution time of an application according to the transmission rate

uitable distribution of workloads among nodes in the same cluster, which prevents saturation of the network and critical points from occurring. There are other aspects of the Fog architecture that could be explored to enrich the current solution, which deals only with the resource allocation problem. Many perspectives could be explored, in particular:

- Our technique could enhanced to provide a logical topology of the network by integrating a network discovery mechanism. By doing so, we will be able to include the cost of the network link in the distance function between requests and nodes.
- It can take advantages of Gateway node subscription based solutions to enable dynamic addition and removal of nodes.
- Lastly, in order to efficiently deal with different requests from IoT devices, a priority concept need to be developed and integrated to our proposal.

References

[1] Kevin Ashton. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[2] Yuanhao Cui, Fan Liu, Xiaojun Jing, and Junsheng Mu. Integrating sensing and communications for ubiquitous iot: Applications, trends, and challenges. *IEEE Network*, 35(5), 2021.

[3] Alem Čolaković and Mesud Hadžialić. Internet of things (iot): A review of enabling technologies, challenges, and open research issues. *Computer Networks*, 144:17–39, 2018.

[4] Somayya Madakam, R. Ramaswamy, and Siddharth Tripathi. Internet of things (iot): A literature review. *Journal of Computer and Communications*, 3(164-173):164–173, 2015.

[5] K. K. Patel, S. M. Patel, and et al. Internet of things-iot : definition, characteristics, architecture, enabling technologies, application & future challenges. *International journal of engineering science and computing*, 6(5):–, 2016.

[6] HoHong Cheo and Varghese Blesson. Resource management in fog/edge computing. *CoRR*, 2018.

[7] Muhammad Fahimullah, Shohreh Ahvar, and Maria Trocan. A review of resource management in fog computing: Machine learning perspective. In *arXiv:2209.03066*, 2022.

[8] Babatunji Omoniwa, Riaz Hussain, Muhammad Awais Javed, Safdar Bouk, and Shahzad Malik. Fogedge computing-based iot (feciot): Architecture, applications, and research issues. *IEEE Internet of Things Journal*, abs/1810.00305, 2018.

[9] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, page 13–16, New York, NY, USA, 2012. ACM.

[10] Cheol-Ho Hong and Blesson Varghese. Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys*, 52(5), sep 2019.

[11] Michele De Donno, Koen Tange, and Nicola Dragoni. Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. *IEEE Access*, 7:936–948, 2019.

[12] C.B. Robert Swanson and al. et al. *Openfog reference architecture for fog computing*. OpenFog Consortium Architecture Working Group, USA, February 2017.

[13] Nacer Edineand Belkout, Khaled Zeraoulia, Mohamed Nasir Shahzad, Lu Liu, and Bo Yuan. A load balancing and routing strategy in fog computing using deep reinforcement learning. In *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, 2022.

[14] Oche Onah John, Muhammad Abdulhamid Shafii, Misra Sanjay, Mohan Sharma Mayank, Rana Nadim, and Oluranti Jonathan. Genetic search wrapper-based naïve bayes anomaly detection model for fog computing. In *Intelligent Systems Design and Applications. ISDA 2020*, 2020.

[15] Nam Giang and Rodger Lea. Developing applications in large scale, dynamic fog computing: A case study. *Software: Practice and Experience*, 50, 2019.

[16] Ayoub Hammal, Mehdi Lerari, Khaled , Zeraoulia, and Youcef Hammal. An efficient resource allocation technique in fog computing environment. In *Intelligent Systems Design and Applications, A. Abraham et al. (Eds.): ISDA 2022, LNNS 646, Springer Nature*, pages 1–11, 2022.

[17] David Gale and Lloyd Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[18] Harshit Gupta, Amir Vahid, Ghosh Dastjerdi, Soumya K., and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *CoRR*, abs/1606.02007, 2016.

[19] Z. Rejiba, X. Masip, and E. Marin-Tordera. A survey on mobility-induced service migration in the fog, edge, and related computing paradigms. *ACM Computing Surveys*, 52:1–33, 2019.

[20] Alsadie Deafallah. Resource management strategies in fog computing environment: A comprehensive review. *International Journal of Computer Science and Network Security*, 22(4):310–328, 2022.

[21] Arani Mostafa, Souri Alireza, and Rahmanian A. Resource management approaches in fog computing: a comprehensive review. *Journal of Grid Computing*, 2019.

[22] Ajay Gulati, Ganesha Shanmuganathan, Anne Holler, and Irfan Ahmad. Cloud-scale resource management: Challenges and techniques. *Proceedings of the 3rd Usenix Conference on Hot Topics in Cloud Computing*, 11, 2011.

[23] Hussein T. Mouftah and Burak Kantarci. *Virtual Machine Migration in Cloud Computing Environments : Benefits , Challenges , and Approaches*. IGI-Global, USA, H. Mouftah and B. Kantarci (Editors)., September 2013.

[24] N. Chandrakala and D. B. Rao. Migration of virtual machine to improve the security in cloud computing. *International Journal of Electrical and Computer Engineering*, 8(-):210–219, 2018.

[25] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito. Container migration in the fog : A performance evaluation. *Journal of Sensors*, 19(-):1488, 2019.

[26] Andrey Mirkin and Alexey Kuznetsov. Containers checkpointing and live migration. In *Linux Symposium, vol. 2, Ottawa, Ontario. Canada.*, pages 85–90, 2008.

[27] Alturki Badraddin, Stephan Reiff-Marganiec, and Charith Perera. A hybrid approach for data analytics for internet of things. In *Proceedings of the seventh international conference on the internet of things*, volume 11, 2017.

[28] R. B. Redowan Mahmud and Kotagiri Ramamohanarao. Application management in fog computing environments : A taxonomy, review and future directions. *ACM Computing Surveys*, 53:1–8, 2020.

[29] R. Mostafa Ghobaei-Arani and A. Souri. Resource management approaches in fog computing : a comprehensive review. *Journal of Grid Computing*, 2019.

[30] Ashish Chandak and Niranjan Kumar Ray. A review of load balancing in fog computing. In *2019 International Conference on Information Technology (ICIT)*, pages 460–465, 2019.

[31] T. Jing V. Wang and Kai-Yin Fok. A stable matching-based virtual machine allocation mechanism for cloud data centers. In *2016 IEEE World Congress on Services*, 2016.