IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 27, 2020, accepted July 2, 2020, date of publication July 8, 2020, date of current version July 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3007944

Hierarchical Load Balancing and Clustering Technique for Home Edge Computing

CHEIKH SALIOU MBACKE BABOU[1,2], DOUDOU FALL[2], SHIGERU KASHIHARA[3], (Member, IEEE), YUZO TAENAKA[2], (Member, IEEE), MONOWAR H. BHUYAN[4], (Member, IEEE), IBRAHIMA NIANG[1], AND YOUKI KADOBAYASHI[2], (Member, IEEE)

[1]Faculty of Science and Technology, Cheikh Anta Diop University, Dakar 5005, Senegal
[2]Division of Information Science, Graduate School of Science and Technology, Nara Institute of Science and Technology, Ikoma 630-0192, Japan
[3]Faculty of Information Science and Technology, Osaka Institute of Technology–Hirakata, Hirakata 573-0196, Japan
[4]Department of Computing Science, Umeå University, 901 87 Umeå, Sweden

Corresponding author: Cheikh Saliou Mbacke Babou (cheikhsalioumbacke.babou@ucad.edu.sn)

This work was supported in part by the Industrial Cyber Security Center of Excellence (ICS-CoE) Core Human Resources Development Program, and in part by Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number JP18H03234.

ABSTRACT The edge computing system attracts much more attention and is expected to satisfy ultra-low response time required by emerging IoT applications. Nevertheless, as there were problems on latency such as the emerging traffic requiring very sensitive delay, a new Edge Computing system architecture, namely Home Edge Computing (HEC) supporting these real-time applications has been proposed. HEC is a three-layer architecture made up of HEC servers, which are very close to users, Multi-access Edge Computing (MEC) servers and the central cloud. This paper proposes a solution to solve the problems of latency on HEC servers caused by their limited resources. The increase in the traffic rate creates a long queue on these servers, i.e., a raise in the processing time (delay) for requests. By leveraging, based on clustering and load balancing techniques, we propose a new technique called HEC-Clustering Balance. It allows us to distribute the requests hierarchically on the HEC clusters and another focus of the architecture to avoid congestion on a HEC server to reduce the latency. The results show that HEC-Clustering Balance is more efficient than baseline clustering and load balancing techniques. Thus, compared to the HEC architecture, we reduce the processing time on the HEC servers to 19% and 73% respectively on two experimental scenarios.

INDEX TERMS HEC-clustering balance, resource allocation, processing time, delay, clustering, load balancing, hierarchical resource management, home edge computing (HEC), edge computing, graph theory.

I. INTRODUCTION

Edge Computing has been adopted at the expense of a central cloud to allow mobility and proximity of the users. Besides, with this extension of the cloud system, users' requests will be processed within the users vicinities. At the same time, a major problem will be solved, concerning certain types of applications that are very sensitive to latency.

However, the explosive emergence of Big Data and the advent of certain real-time applications, which require a response time of less than 1 ms [1], [2], have marked the world of Information and Communication Technologies (ICT). Consequently, the problems of resource management and delay for these applications are becoming more and more recurrent. In addition, on Edge Computing architecture, the distance between the user equipment and the base station was not in step with the requirements of these real-time applications [3]. This posed a problem for applications ultra-low latency compared to network communication, because the equipment where these real-time applications are stored can be very far from the base station.

Hence, several types of architecture related to that of the edge computing have been proposed. Among them, Home Edge Computing (HEC) [3], which is a three-tier architecture-consisting of the central cloud, Multi-access Edge Computing (MEC) server, and HEC server. Compared to other existing Edge Computing architectures, the HEC can significantly reduce network latency and thus meet the requirements of certain types of real-time applications (e.g. Virtual Reality, Augmented Reality, etc.). However, the

The associate editor coordinating the review of this manuscript and approving it for publication was Md. Arafatur Rahman.

processing time on the HEC architecture increases considerably, which will pose a real problem for these real-time applications [3]. This is due to the limitation of resources (CPU, RAM and Storage) in the Home Edge Computing (HEC) servers or local servers which constitute the third level of architecture that interact directly with applications. Besides, with the overload of HEC servers which is caused by the poor distribution of workloads on the architecture during peak hours or in areas where traffic is very high, certain tasks, sensitive to latency, cannot be accepted all the time or will take a long time to be executed. This situation causes a load balancing problem on the HEC architecture. Indeed, the proposal of this new architecture of the Edge Computing, i.e., HEC, allows a considerable amount of reduction in network delay due to the proximity of the HEC servers but create on these servers a very high processing time [3]. Moreover, this increases queue size caused by this limitation of resources on HEC servers promotes an increase in the processing time of the applications, especially those which are very sensitive to latency, even if the HEC server is close to the users. We also note that in the architecture of the HEC, there are some of the nodes which are used less than the others according to the density of the users. This creates a poor load distribution on the whole HEC architecture.

From this fact, we propose our method called HEC-Clustering Balance technique that significantly reduce the processing time of requests sent by users while efficiently utilizing resources of HEC server, MEC server, and the central cloud on the three-tier architecture of HEC. Our technique also allows to use the HEC-Servers which are not solicited all the time. The proposed method handles the Cloud-MEC-HEC structure as a graph. Each MEC, called cluster-head, manage a sub-graph, i.e., cluster. Upon each cluster (3-TIER), there is a cluster-head that constitutes the upper layer or 2-TIER of the HEC architecture. Then the cluster-heads are connected together on the one hand and with the central cloud on the other hand.

In this paper, our proposed methods, namely, clustering and hierarchical resource allocation technique, allow to simultaneously solve two problems encountered on the HEC architecture. First, the limitation of resources on these HEC servers which causes an increase in latency and possibly a long queue, consequently, the requirements of latency-sensitive applications are not met. Then, a poor load distribution is noticed on the HEC architecture because, when some servers are overloaded, others synchronously remain almost unused. Hence, we need to balance the workload on the overall architecture of the HEC. Thus, the combination of these two techniques increases the performance of HEC architecture on the hierarchical and distributed processing of requests. Besides, we use RAM, MIPS, CPU and storage as performance evaluation parameters in this proposed method. Thus, we validate our proposed technique using simulation and compared with existing techniques about clustering on the one hand and load balancing on the other hand.

In summary, our research objectives were describe as follow:

1) Present and discusses different resource management techniques in the Edge Computing environment, namely, resource discovery technique, Benchmarking technique, data placement technique, computing technique, clustering and load balancing techniques.
2) Present a resource management framework and performance evaluation for the new architecture of Home Edge Computing (HEC).
3) Proposes a clustering method to pool resources at the level of HEC servers to remedy the problems in [3].
4) Proposes a hierarchical Load Balancing technique in this HEC architecture at three levels to meet the requirements for certain real-time applications.

This paper is organized as follows. In Section II, we explain the related works. In section III, we talk about the architecture we want to work on, namely, the HEC, with more details, and we propose a framework for the HEC architecture allowing us to have a global view on resource management techniques. Section IV discusses the Hierarchical resource allocation proposal. In the section V, we describe the Simulation results. Finally, in Section VI, we discuss the conclusion and future works.

II. RELATED WORK

In the Edge Computing system, several resource management algorithms (techniques) have been proposed to reduce the communication delays (latencies) between the user and the Edge System [7]. Thus, these resource allocation techniques on the Edge system can be classified into five categories: Resource Discovery, Benchmarking, Data Placement, Computing, Clustering and Load Balancing. We have categorized these techniques based on certain criteria for resource allocation in Edge Computing networks such as: identifying available resources (Resource Discovery), defining resource performance for decision making to maximize performance in deployments (Benchmarking), identification of appropriate resources for deploying a workload (Data Placement), calculation decisions based on scheduler performance and traffic type priority (Computing), reuse of residual resources thanks to the pooling of nodes (Clustering), and the distribution of workloads between resources (Load Balancing).

Thus, several authors have made publications to solve the issues related to the reduction of the execution costs on the latency about communication (network) and processing (server). These problems were more and more recurrent with the advent of Edge Computing. In addition to the overload, we are seeing, with these edge servers, an inequality related to the utilization rate of these edge servers as a function of their location [6].

Thus, in this section, we will talk about different resource allocation techniques in the edge computing according to the classification mentioned below but also corresponding to the problems that we try to solve.

A. RESOURCE DISCOVERY TECHNIQUES

These types of algorithms make it possible to identify the different resources available on the Edge Computing systems in order to perform the distributed calculation [7]. Furthermore, it is a big challenge to identify an available resource on edge systems [8], [9], [11].

Do *et al.* [12] proposed a distributed solution for joint resource allocation and minimizing carbon footprint problem [13]. They formulated the problems as a general convex optimization, where the location diversity of requested video streaming utility and costs are modeled. Unfortunately, the carbon footprint formulated proposal does not provide solutions in case of overload of this centralized Data Center or if Fog Computing Nodes (FCN) are out of service.

Hassan *et al.* [14] proposed an approach for the secure uploading of applications dynamically. Their approach takes into account the availability and proximity of resources. Moreover, the execution costs (delay) of applications will be uploaded on these edge servers. They explore the potentials of the fog computing system for offloading and storage expansion. However, their solution cannot guarantee anytime available resources in the case where the fog server is overload. That means, the performance goes down slowly.

B. BENCHMARKING TECHNIQUES

The resource allocation management of the Edge Computing system using the Benchmarking technique is very difficult to implement. It is essentially based on the evaluation of power, CPU, and memory on Edge system resources. Edge Computing Benchmarking is divided into three classes: an analysis based on functional properties assessments, applications, and integrated Benchmarking. Thus, the challenge at the Edge Computing system in parallel with the performance analysis using the Benchmarking technique can be seen under three levels, namely, the performance test of specific using Enos [15] approach to collect a performance metric, the execution of additional applications, which take much more time and the integration of the Cloud system with that of Edge Computing on the evaluation of resources. However, there is still a lack of research for performance evaluation based on Benchmarking techniques. As mentioned in [16], authors are not able to provide techniques to perform the live container migrations between entities (server) in case of overload. That means, the network based on the Benchmarking method will still be efficient. Moreover, the mobility of edge entities in several IoT/Edge use cases is not taken into account in this method. Such a scenario will impact latency during communication between the user equipment and edge server [17].

C. DATA PLACEMENT TECHNIQUES

One of the challenges in the Edge Computing system is the management and location of the data from the user. Unlike the traditional central cloud system, the Edge System is resource-constrained. The existing data placement techniques take into account limited resources on the Edge System [21] but do not consider certain criteria namely, mobility. That is why authors have tried to solve the aforementioned issue by classifying existing techniques into dynamic techniques sensitive to certain conditions, and iterative techniques. Thus, the iterative techniques are subdivided into a hierarchical iterative method that performs allocation based on resources (CPU, memory, and bandwidth) and, iterative method according to several constraints namely, resources, quality of service (QoS), latency, type of applications, etc. [23], [24]. However, Taneja [24] proposed a module mapping algorithm for the efficient utilization of resources in the network infrastructure for IoT applications. Unfortunately, the efficiency of this technique is only for the static part of network performance, the dynamic network characteristics (network connectivity, failure of nodes, etc.,) and scheduling policies of resources remain to be improved.

D. COMPUTING TECHNIQUES

Due to the limited resources available to Edge Network servers, the recurring problem in system is resource management. To this end, several research projects have been oriented in this direction to improve these techniques at the Edge Computing level. These algorithms make it possible to perform resource sharing calculations in the edge computing system. Thus, they can be subdivided into 3 types of algorithms namely, Resource sharing, Task Scheduling, Offloading and Load Redistribution [10]. Authors [30] try to solve task offloading problems in the edge network based on the Heuristic offloading decision algorithm. However, they still did not solve the issue of joint optimization of communication and computation resources in the edge networks.

E. CLUSTERING TECHNIQUES

Several techniques for allocating resources based on the cloud federation (Cluster) in the Edge Computing system have been proposed to minimize execution times.

This Paper, Chen *et al.* [37] proposed a clustering technique to group the proximate user devices to minimize computing latency. For that, they exploit computing and storage resources via joint task offloading and proactive caching. These results show that this method can gain up to 65% about performance delay but still improve the traffic intensity to local computing like in the cloudlet. From that point, the delay can be improved by processing the task close to the user like in the cloudlet cluster.

Bouet *et al.* [32] proposed an optimization technique using the Geo-Clustering approach with the concept of the Mixed Integer Linear Programming (MILP) formulation. For that, they proposed an algorithm that, based on the spatial distribution of the communications, satisfies the request according to their requirement according the types of application on the MEC server. In addition, the algorithm takes into account the maximum capacity of the server expressed in terms of resources (CPU, storage, ...). To evaluate these methods like clustering and MILP, they use the dataset of mobile communication. They show that the clustering takes into account

the spatial distribution of the communications and enables to largely offload the core. They evaluate the MEC partition and show that there is no saturation in the MEC server by using in the latter both techniques clustering and balance in the MEC server. However, they still exploit several aspects thus as group communication and latency according the types of applications.

F. LOAD BALANCING TECHNIQUES

Load distribution algorithms allow us to balance the overall workloads of the edge computing. These algorithms also allow networks that implement QoS to be able to distribute workloads based on certain criteria such as traffic priority, etc.

To ensure high availability on the network side but also on the system of the Cloud Computing system, load balancing mechanisms must imperatively be implemented. These mechanisms constitute a real challenge to take into account the techniques of resource allocation. However, load balancing algorithms use primarily four techniques: Particle swarm optimization [19], Cooperative load balancing [6], Graph-based balancing [18], and the use of Breadth-based searching [20].

Beraldi *et al.* [32] proposed a cooperative schema between data centers, called CooLoad. This technique allows the collaboration of two data centers at the edge of the network concerning the processing of requests. In other words, the CooLoad allows an overloaded data center to be able to transfer future requests to the other data center that has more resources based solely on CPU usage. For this, they defined a mathematical model to propose an effective cooperation strategy to show how the blocking probability and the service time can be reduced at the same time.

Xu *et al.* [31] proposed a method of dynamic resource allocation for load balancing called DRAM in an edge computing environment. This method allows achieving a very high level of load balancing for all types of computing nodes in the fog computing platform according to its capacity and type of services. For this, their method has been classified into four main steps. unfortunately, this method can't analyze the negative impact of the service migration like the traffic for different types of computing nodes, the cost for service migration, the performance degradation for the service migration, and the data transmission cost.

Liu *et al.* [22] proposed the Particle Swarm Optimization (PSO) algorithm in order to reduce the energy consumption and improve the computing resource allocation in the Edge systems. This Computing resource allocation include two parts. The first one is the power control scheme based on the potential game theory which will allow to reduce the energy consumption of MEC networks. The second one is the computing resource allocation scheme based on linear programming. The latter allows them to improve the average computing resource coefficient of MEC networks.

Li *et al.* [25] proposed an Edge Computing IoT (ECIoT) Architecture to address challenges like mass connections, big data processing, huge power consumption about IoT Platform. Also, in order to improve the performance in the ECIoT Architecture, they developed, an approach based on the Lyapunov stochastic optimization. For more flexibility and scalable in the ECIoT, they make in the network architecture the Software Defined Network Technology. Moreover, by utilizing SDN-Based architecture, the network can be easily managed and also the network capacity can be increasing (With service management).

Alsaffar *et al.* [26] proposed a collaborative platform likes resource allocation methods with 2 levels, Fog and cloud paradigms. This proposed algorithm depend of linearized decision based on tree conditions like VM capacity branches (enough or not), the completion time (now or later) and the service size (small or large). With this methods, they were considerably reduced the total overhead for big data processing in the fog-cloud system. They also proposed resource allocation algorithm based on Service Level Agreement (SLA) and Quality of Service (QoS) in order to optimize the big data distribution and in fog and Cloud systems.

Kelaidonis *et al.* [27] offer an architecture called Edge Cloud-IoT to bring innovation in 5G infrastructure with the multilayered approach for managing services in the distributed cloud environment. Thus, the proposed solution brings a new wave of technological innovation depending on the services and applications but presents certain shortcomings such as the migration of services and the aggregation of data in environments close to the user (clustering).

Maheshwari *et al.* [28] present an analysis of the evolution of the performance of a decentralized cloud system on very latency-sensitive applications in order to have a better understanding of the basic parameters for a good local cloud design such as resource management, bandwidth, latency, and the distance between the central and the edge cloud. However, authors did not consider the impact of mobility and the distribution of tasks on the Edge cloud.

Pahl *et al.* [29] proposed a clustered architecture using the Raspberry Pi as a local server. Thus, the main objective is to set up a gateway between the terminals (IoT devices) and the central cloud for certain types of the application whose processing do not require to be performed in the central cloud. In addition, they proposed an analysis of a container and cluster management system on Raspberry Pi. However, their evaluation shows that effective data and network management still needs to be improved.

III. OVERVIEW ON HOME EDGE COMPUTING

A. HOME EDGE COMPUTING CONCEPT

Home Edge Computing (HEC) [3] is an architecture for having a storage and data processing device near the users (HEC Server); it also allows us to set up a micro-cell at the user to reduce the workload at the base station located in the MEC and improve system performance. The HEC architecture comprises three levels of cloud namely local cloud or Home Server (3-TIER), edge cloud (2-TIER), and

central cloud (1-TIER) (Fig. 1). The term "Home" in Home Edge Computing does not restrain our work to the homes of the users, we take into account other places where the users could connect to the Internet such as: Building, shopping malls, hospitals, etc. Thus, HEC is a new architecture of the edge computing system that is more in proximity to the users compared to Cloudlet, Fog Computing and Multi-access Edge Computing. HEC is a concept proposed to solve the latency problem still present in MEC for certain types of applications that have very high needs in term of resources and have to be processed with relatively reduced delays. With this concept, we will no longer need to go to the MEC or the central cloud for some queries that do not require a lot of computing resources for their processing. For instance, Smart Health allowing us to have real-time information from the patient on the variation of his heart rate to detect possible anomalies. However, thanks to the synchronization with the edge computing and that of the central cloud, in case of unavailability of the resources on the Home Server, the latter will automatically launch a request, in a hierarchical way, to these two systems which potentially possess resources for satisfying the computational needs of the task.

The architecture for Home Edge Computing is depicted in Fig. 1. It is composed of three levels: HEC Server (3-TIER), MEC Server (2-TIER) and Central Cloud (1-TIER). In the environment of the HEC, all equipment will be wired or wirelessly connected to the local cloud. The Home Server serves, at the same time, as a gateway for this equipment outside the local network because it will be installed and managed by the Internet service provider. Thanks to the proximity of the HEC, the latency-sensitive queries will be rapidly processed. Moreover, for its connection to the rest of the network, we will have to use the system FTTx (Fiber To The xHome, Office, etc.) because it has an ultra-low latency and its flow can reach up to 1Gbps, which a bandwidth of latency requirements of potential 5G use cases, like latency-sensitive applications [3]. Thus, for its operation (Fig. 2), if a request leaves the user's equipment, it is loaded in the box provided by the Internet provider. The HEC Server inside the box will handle the customer's request. On this HEC server, if the request cannot be processed, the system will hierarchically transfer the request to the cloud (MEC Server or Cloud) [4], [5]. In addition to lightening the load of the MEC, HEC allows having micro-cells which can also help process latency-sensitive applications. With the Home Edge Computing (HEC) architecture, we can see that the only difference with MEC is the Home Server hosted by the customer. The Home Server has to be tiny, non-cumbersome and transparent to the customer. It must be able to fit within the box that the customer received from the internet service provider i.e. it has to be a mini computer e.g. Raspberry Pi.

B. HOME EDGE COMPUTING SYSTEM FRAMEWORK FOR MANAGEMENT

In this subsection we propose the framework (Fig. 5) provides a global view of the different ways of allocating resources in

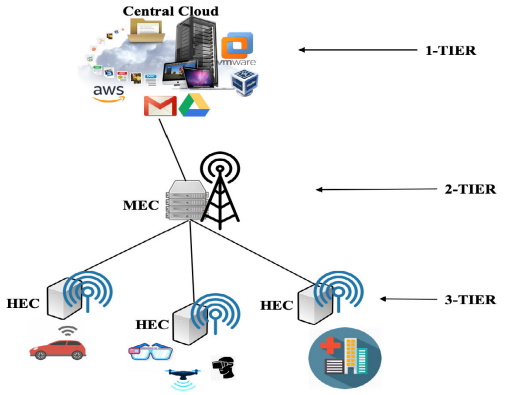


FIGURE 1. Overview on Home Edge Computing (HEC) architecture.

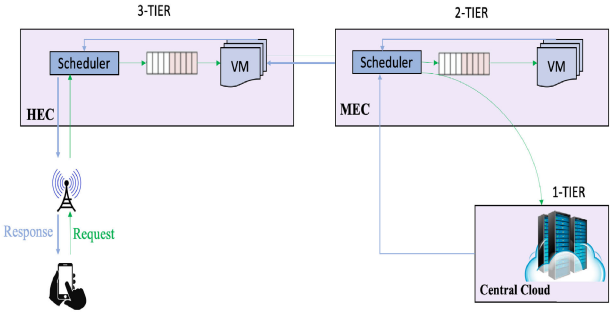


FIGURE 2. Resources allocation on HEC architecture.

HEC architecture. Thus, this framework consists essentially of three parts namely the architecture layer, the data flow control layer and the software control layer. Each time a request is received on the architecture layer, it will either be processed on one of these three types of servers (HEC, MEC or Cloud). This is possible thanks to the control layer which transfers data flows from the 3-TIER level to the 1-TIER level according to the availability of resources. Finally, the software layer is responsible for centralizing and automating the management of these data flows with the virtualization of the entire HEC architecture. We talk in detail about the different features that make up each layer of this framework.

1) Dataflow Control layer
   a) *HEC Dataflow:* On the HEC architecture, the movement of workloads is from the user to the central cloud, passing of course respectively by the HEC and the MEC. In other words, if a request is sent by the user, it will be received first by the HEC (or HEC cluster). If the capacity on this HEC is sufficient, the task will be processed on it. In case of insufficiency, the request will be transferred to the MEC level. In this case, the same treatment as that at the HEC will be performed. Thus, if the task cannot be processed on the MEC, it will be transferred directly to the central cloud where it will be processed because it is assumed that these resources are unlimited (Fig. 2). In

the same way as with existing Edge Computing systems, HEC workloads can be aggregated, shared, or offloaded. For each of these techniques, several solutions have been proposed.

b) *HEC Control:* This is the general resource control of the three-tier architecture of the decentralized cloud system but in particular that of the HEC. Control can be established either centrally (using a controller) or decentralized (e.g. using game theory, blockchain, etc.). Thus, the resource control of this architecture will be at the third level, that is, the level closest to the HEC server. Therefore, it will be responsible, for example, to manage the resources available on each cluster in order to satisfy the tasks sent by the client, to be able to automatically update the system each time a request is satisfied, to take into account the constraints (i.e. QoS, real-time applications) for each traffic type, etc.

c) *HEC Tenancy:* This architecture allows us to better know how the applications are spread over the entire HEC architecture. In other words, depending on available resources, an entity or application can use some or all of it. Thus, the tenant-based resource management architecture can be classified into two types (similarly defined on the Edge system), namely the single-tenant system or the multi-tenant system. These systems are respectively defined as the exclusive use of resources by an entity and the use of multiple entities for the same resource. For the HEC architecture, which is considered a distributed architecture where all types of applications can be hosted, the Multi-tenant method is best suited as an architectural classification method for resource management.

2) Software Control layer

The software part for resource management can be classified on the HEC architecture in two parts, namely the Middleware and the Software.

a) *Virtualization systems:* The virtualization that is the main method of resource management has revamped the world of cloud computing because it is thanks to this technique that several machines can be created on the same virtual machine. Thus on the HEC infrastructure, we have the virtualization system, which will allow the creation of virtual machines on physical machines. Then we have the virtualization system based on containers. It plays a similar role that VMs has the difference, it offers a slight treatment during virtualization. Besides, it facilitates the adoption of the performance of architectures whose resource is limited (for example that of HEC).

b) *Network virtualization systems:* Thus, the network virtualization system is a major asset on the cloud compared to the allocation of VMs but also concerning the centralized management of resources. Thus, several techniques have been developed for the control listed resources but also listed network. Among them, the two most used are Software Defined Network (SDN) and Network Function Virtualization (NFV). These methods allow you to manage and control the network and these features.

c) *Middleware:* It is a complementary service to the software platform. It allows them to have a vision of performance on the machine where it is hosted. It also allows to provide operating systems hosted on these servers with additional services. Also, he is responsible for coordinating the calculations distributed on the servers, orchestrating installations (communications, protocols) and deploying virtual machines on the nodes of the edge computing network.

3) Architecture layer

This is the part of the hardware components of the HEC architecture. It is composed of three types of server. First, we have HEC (3-TIER) type servers that are directly connected to IoT devices, thus solving latency problems for real-time applications. Then these HEC servers are directly connected with the MEC servers (2-TIER). Finally, the MEC is directly connected on with the central cloud (1-TIER). Besides, users are directly connected to HEC servers with wireless technology and MEC servers (level 2) for equipment in motion. Thus, the processing of requests from users is more efficient in terms of execution cost (latency) than when it is performed on the server being as close as possible to the user (especially for applications very sensitive to latency).

C. HOME EDGE COMPUTING PERFORMANCE EVALUATION

Based on our experiment, the results of which are described in Fig. 3, we were able to show that the three-level architecture of the HEC makes it possible to considerably reduce the network latency. This low latency is gained due to the reduced distance between end-users devices and local data center (home server). In other words, Thanks to our method and the help of our simulation platform, we were able to considerably reduce the transmission time of requests using HEC, i.e., the Home Server. Thus, the average delay, under the same conditions, went from 94.82% on the MEC to 5.17% at the HEC level (Fig. 3).

Moreover, as the response time does not depend solely on the network delay, we have also taken into account the service time and the processing time. According to Fig. 3, it was possible to significantly reduce the latency between the MEC and the HEC based on the requests launched by the mobile equipment. In Fig. 4, we have a visualization of the

processing time and the service time of the two data centers: the home server and the edge server. Thus, after analysis of the information, we found that, on average, the service and the processing time of the HEC are higher than those of the MEC unlike the situation in Fig. 3. Unsurprisingly, this can be easily explained by the fact that the resources at the MEC level are more important than those of the Home Server according to the MIPS, the number of cores, the RAM, etc.

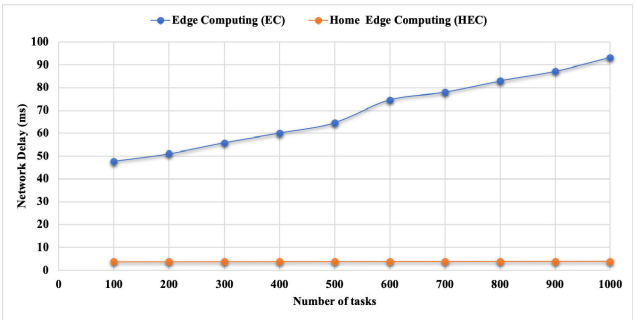


FIGURE 3. Network delay.

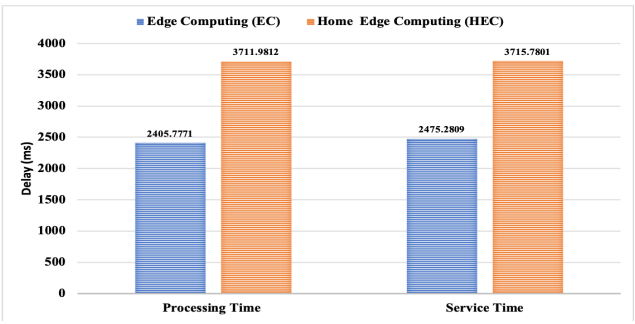


FIGURE 4. Latency between HEC and MEC.

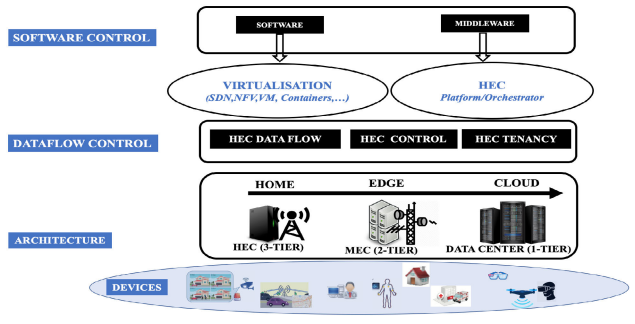


FIGURE 5. Framework proposal on HEC system for resource management.

IV. HIERARCHICAL RESOURCE ALLOCATION PROPOSAL IN HEC ARCHITECTURE

In this section, we describe our proposed solution based on HEC architecture. In subsection IV.A, we model the HEC architecture using graph theory. In the subsection IV.B, we talk about the proposed resource allocation solution in a hierarchical environment.

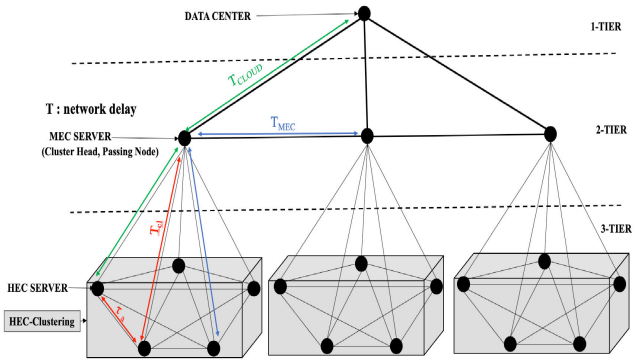


FIGURE 6. Home Edge Computing (HEC) architecture proposal based on clustering.

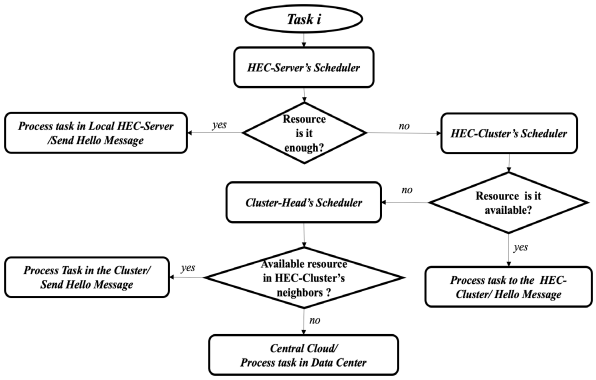


FIGURE 7. Hierarchical task processing model in HEC clustering architecture.

A. SYSTEM MODEL

We model HEC servers as a multi-flow system. For each vertex (HEC server), we associate at least a request di as being the quantity of data of the request. Thus, we consider the HEC architecture (Fig. 6) as being a complete non-oriented graph $G = (X, A, C)$, where $i \in X$ corresponds to the vertex (HEC-Server and MEC-Server) and $a \in A$ define the different edges between the pairs of nodes (x, y), with $(x, y) \in A$. Also, X contains all servers, A is set of fiber link interconnecting them and C represents the maximum capacity of each link. Thus, the number of edges a that can be defined in this interval is determined as follows:

$$i - 1 \leq a \leq \frac{i(i-1)}{2} \tag{1}$$

Besides, each edge a has a capacity C such that:

$$C_a \geq 0, \quad \text{with } a \in A \tag{2}$$

With each vertex $i \in X$, we associate a traffic demand denoted d_i. Let φ, the data flow from vertex i. We define φ_a^k as data flow crossing an edge a with $k \in (1, \ldots, n)$. Thus,

Algorithm 1 :HEC-Clustering Algorithm

Result: Updating the local database on the nodes of the HEC-Cluster after receiving a HELLO message (node id, CPU, RAM, Status, Storage)

```
while HELLO message ∃ do
    if message is from new HEC-Server then
        Add a new instance from HELLO message to the HEC-Server database.;
        Send ACK to the new node. ;
    else if message is from cluster head then
        Remove the node instance in the local database. ;
    else
        Update the information about an instance of the local database of the corresponding node. ;
    end
end
```

the total flow of data φ_a passing through in edge a is defined as follows:

$$\varphi_a = \sum_{k=1}^{n} \varphi_a^k \leq C_a \tag{3}$$

We then assume that there is no waiting queue if :

$$\varphi_a \leq C_a \tag{4}$$

Let τ_a be the cost of the network communication edge a:

$$\tau_a = \frac{d_a}{C_a} \text{ , where } d_a \text{ is current workflow in edge } a \tag{5}$$

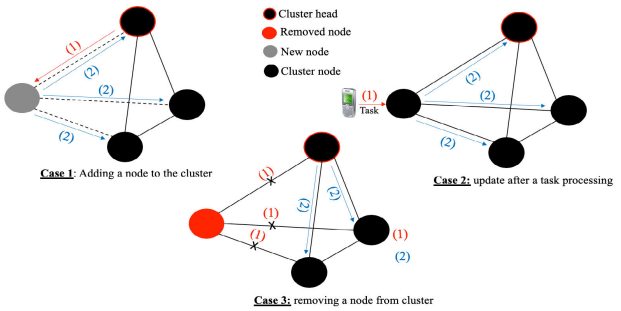


FIGURE 8. Clustering process in HEC architecture.

B. HIERARCHICAL CLUSTER-BALANCE PROPOSAL ON HEC ARCHITECTURE

Algorithm 1 describes the clustering process in the architecture of HEC. The oncoming *HELLO* message is triggered either by adding/removing a new node in the cluster or for updating after processing a new task as show in Fig. 7. In case 2 of Fig. 8, after executing the task, the node sends a HELLO message to all the nodes of the cluster to allow them to update their local database. In other words, the nodes compare the information received from the *HELLO* message with that of the local database of the HEC server. In addition, this local database contains all of the neighboring states, namely id_i, id_{cl}, RAM, CPU, $Status$, and $Storage$.

Algorithm 2 : Hierarchical Cluster-Balanced Algorithm on HEC Architecture

Result: Execution delay D_i of the task from user(s)

```
while Task T_i ∃ do
    if local HEC-Server has enough resource then
        Process to create VM for Task T_i in local HEC-Server;
        D_i ← T_proc ;
        Algorithm 1 (Send updating HELLO message) ;
    else
        goto Cluster Scheduler ;
    end
    if HEC-Cluster has enough resource then
        Process to create VM for Task T_i in the HEC-Cluster;
        D_i ← (n × T_proc + T_cl) ;
        Algorithm 1 (Send updating HELLO message) ;
    else
        goto Cluster-Head Scheduler ;
    end
    if Resources are available on neighboring HEC-Clusters then
        Process to create VM for Task T_i in the neighboring HEC-Cluster;
        D_i ← (n × T_proc + T_MEC) ;
        Algorithm 1 (Send updating HELLO message) ;
    else
        goto Central-Cloud Scheduler ;
        Process to create VM for Task T_i in the Central Cloud;
        D_i ← (T_proc + T_Cloud) ;
    end
end
```

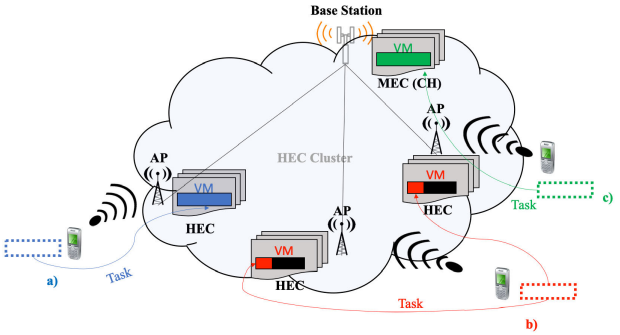


FIGURE 9. Possible outcomes of computation decision according to the available resources in HEC cluster.

Note that the cluster head (MEC server) *ID* (id_{cl}) always takes the lowest cluster number and each time the node is added, the *ID* is incremented. Each time a node is added via a wired link (fiber optic), the MEC gives it an *ID* by incrementing the *ID* from the last node added.

Then, if there is a new node in the cluster as show in case 1 in Fig. 8, the cluster head sends to the new node a copy of its local database which has all node parameters (id_i, id_{cl}, *RAM*, *CPU*, *Status*, and *Storage*). As a result, the new node send a HELLO message to all the nodes of its cluster. Hence, all cluster nodes update their local database by adding an entry about this new node, e.g. the parameters from *HELLO* message.

Finally, in case where a node is remove to the cluster (case 3 in Fig. 8), the cluster head send an update message to all the nodes of the cluster to notify the removal of the node. Upon receipt of this HELLO message, each node in the cluster know deleted node's ID and update its database. Thus, to ensure continuity of nodes' ID, each node compares its ID with that of the deleted node. If the ID of the deleted node is greater than its ID, the node maintains its ID. Otherwise, its ID should be decremented (ID = ID - 1).

In summary, according to Fig. 8, there are three types of HELLO messages, namely updating HELLO message (case 2), adding HELLO message (case 1) and removing HELLO message (Case 3).

In algorithm 2, we define the hierarchical load balancing process using clustering according to algorithm 1. Thus, on the arrival of a request in our clustered architecture of HEC, as show in Fig. 7, the HEC server's scheduler, close to the user, receives the task. If the resources to execute the task are available on this node (HEC server), the scheduler allow the execution of this request on this HEC server (Fig. 9.a)). Therefore, we call algorithm 1 to update the available resources in the cluster node. Besides, if the necessary resources are not available to execute the task on the local server (HEC Server), the HEC server's scheduler checks in its local database the availability of resources on the nodes of the cluster. In case resources are available on this cluster, the request will be sent to the servers available to process the request (See Fig. 9.b)). However, if the resources are not available on the cluster, then the task will be forwarded to the cluster head (MEC Server) as show in Fig. 9.c). On this server, the MEC's scheduler performs the same procedure, by checking whether the neighboring cluster heads have sufficient resources to execute the task by using MEC. Thus, if the resources are available on one of the neighboring clusters, the request will be sent to the neighboring cluster via its cluster head. If all the neighboring clusters cannot process the request, it will be directly transferred at higher level like the central cloud where it will be processed. We Note that the status of a node is either a *Cluster Head*, a *Passing Node* or a *Member Node*. Thus, the *Cluster Head* is the main node of the cluster and its role is to organize and coordinate communications within the cluster. Then, a *Passing Node* is a node of the cluster that has the function of ensuring the interconnection between two or more clusters. Finally, a node within a cluster that is neither cluster-head nor passing node is qualified as a *Member Node*, for example, HEC cluster nodes. For the HEC Clustering architecture (Fig. 6), the MEC server has both proprieties like *Cluster Head* and *Passage Node*. Thus, sending the HELLO message for the presence of a new node automatically triggers the clustering process. Besides, upon arrival of each request on the HEC server, the scheduler checks whether the resources necessary to execute the request are available on this server. Thus, HEC scheduler execute the task on its server as show in Fig. 7. Hence, the cost of execution T_{proc} being the processing/computing time necessary for the servers to satisfy the requests sent by the users, and this time is defined as follows:

$$T_{proc} = \frac{1}{C_{cpu}} \sum_{u \in U} Q_u \times P_u \tag{6}$$

where Q_u represents the amount of data sent by the user u, C_{CPU} the current processor capacity where the request will be processed. $P_u \in \{0,1\}$ is the binary associated with the user for the need or not to send a task to the T_{prodc} and defined as follows:

$$P_u = \begin{cases} P = 1, & \text{if user } u \text{ receive task} \\ P = 0, & \text{otherwise} \end{cases} \tag{7}$$

The server can be the HEC, the MEC or the central cloud.

If the task can be executed on the nodes of the cluster, it will be distributed on the available nodes, i.e. which verify the equation (4). Therefore, the network execution cost is defined as follows:

$$T_{cl} = 2 \times \sum_{a=1}^{n} \tau_a \tag{8}$$

where n is the number of available nodes. In actually, T_{cl} is the round trip time of each link $a \in (x, y)$ to send the task from node x to node y and vice versa.

In this case, the execution delay D for the task i should be:

$$D_i = n \times T_{proc} + T_{cl} \tag{9}$$

The scheduler of MEC checks whether this request can be processed on other HEC-cluster via these neighbors Cluster-Head. In this case, the network execution cost T_{MEC} is:

$$T_{MEC} = 2 \times (\tau_{MEC} + \sum_{a=1}^{n+1} \tau_a) \tag{10}$$

where τ_{MEC} is the network latency between cluster heads.

Then, the execution time there is:

$$D_i = T_{proc} + T_{MEC} \tag{11}$$

In other words, τ_{MEC} defines the cost of the round-trip communication from the cluster network where the client is located to the neighboring cluster having the resources necessary to process the request (Fig. 6). Note that each

time a request leaves a cluster to another it goes through the passage node, i.e., the Head cluster. Actually, the passage node constitutes the gateway between the cluster where it is located towards other clusters on the one hand and towards the higher level, i.e., the central cloud on the other hand.

So, if the resources to process the task from the users is not available in all clusters, the request will be directly transferred to the central cloud. And there, we assume that the resources are always available to satisfy the user's request (Fig. 7). Hence the cost of network communication T_{CLOUD} to the central cloud is defined by:

$$T_{CLOUD} = 2 \times (\tau_a + \tau_{MEC-C}) \tag{12}$$

where τ_{MEC-C} is the network latency between MEC (cluster head) and central cloud.

T_{CLOUD} defines the round-trip communication delay between the cluster node where the customer sent the request and the central cloud (see Fig. 6).

If the request perform in the cloud central, the total execution delay is defined as follow:

$$D_i = T_{proc} + T_{Cloud} \tag{13}$$

Note that each time the request is transferred to a higher level, there is a network communication that will add to the cost of executing the request. For each case where the task to be processed, namely on the cluster, on the cluster head level or the central cloud, the network time is defined respectively by T_{cl}, T_{MEC} and T_{CLOUD}.

TABLE 1. Abbreviations list.

Notation dictionary	
G	Graph
X	Vertex
A	Edges
C_a	Capacity of Edge $a \in A$
D_i	execution Time required a task T_i
id_i	identity of node where $i \in (1, 2, ..., n)$
id_{cl}	Identity of Cluster
CPU_i	remaining Processing capacity of each node i
RAM_i	Random Access Memory of node i
$MIPS_i$	Millions Instructions Per Second of node i
$Status_i$	Status of nodes i
$Storage_i$	Storage capacity of the node i
T_{proc}	Processing time in a server (HEC, MEC, or Cloud)
T_{cl}	Network delay to offloading to cluster nodes
T_{MEC}	Network delay to offloading between clusters
T_{CLOUD}	Network delay to offloading from node to cloud

V. SIMULATION RESULTS

In this paper, the proposed method is a combination of the clustering technique and that of load balancing to respectively reduce the limitation of resources and ensure high availability in the architecture of the HEC. Thus, upon receipt of a request by a node in the cluster, the system verifies whether the resources on this node are sufficient to process the request. If these resources are not available, the clustering method shares the request on all the nodes where the resources are available. If resources are not available on all the nodes of the cluster, the request will be transferred to the other clusters on the network or to the central cloud by the Load balancing method. This method significantly reduces the delay on nodes (servers). Thus, these two techniques combined solve the latency issues in edge computing systems.

TABLE 2. Devices configuration.

	Parameters	Characteristics	Values
Datacenter	Architecture	"x86"	
	OS	"Linux"	N/A
	VMM	"Xen"	
Host	VMM	"Xen"	N/A
	RAM (MB)		2048
	MIPS	N/A	2000
	BW (MB)		10000
	Storage (MB)		10000
VM	VMM	"Xen"	N/A
	RAM (MB)		512
	MIPS	N/A	1000
	Num. of CPU		1
	Storage (MB)		10000

Thus, in this section, we put into practice the new Clustering Balance solution proposal on the HEC architecture. To show that the proposed clustering balance solution namely HEC-Clustering is better than among some existing methods used in this part, we use a simulator called WorkflowSim [34]. WorkflowSim has modules for experimenting with cluster environments and also for comparing them with some of the latest clustering techniques.

The main aim of this paper is to improve the latency for processing requests from users in the architecture of the HEC-Clustering and to compare it with processing time in the traditional architecture of the HEC [3] defined in Fig. 3 and Fig. 4.

For the validation of our proposed HEC-Clustering, we will compare with Heterogeneous Earliest Finish Time (HEFT) [35], Horizontal Clustering and Vertical Clustering [37].

Thus, the reason why we chose these reference algorithms, namely Vertical Clustering (VC), Horizontal Clustering (HC) and Heterogeneous Earliest Finish Time (HEFT) for our simulation is based on the functioning of our HEC-Clustering technique. In other words, the HEC-Clustering algorithm combines the methods of VC and HC, respectively according to the hierarchical choice of processing requests from the HEC-Server to the central cloud and from HEC-cluster to another HEC-cluster via the cluster-heads. This technique allows us to have the functionality of load balancing towards clusters where resources are available. Besides, the HC technique makes it possible to merge several tasks, like a job, at the same level (HEC, MEC or Cloud) and to be able to process them within the same available data center [37]. Thus, thanks to the HC, our HEC-Clustering Balance technique will search among the head clusters, those which are likely to process the tasks (job) coming from the users in the case where the resources of these users' HEC-cluster are insufficient. For the VC, the task's processing is done within a pipeline [37], from the server with the smallest capacity (HEC-Server)

to the one whose capacity is undetermined (central cloud), i.e., where we assume that these resources are always available. Thus, the choice of VC and HC makes it possible to verify that their combination proves more effective and allows to solve certain problems in the Edge Computing system according to the latency. Regarding the HEFT technique, we compared it with our solution to show that we can manage to reduce the latency of all types of tasks without applying prioritization rules. Actually, the HEFT technique is based on two fundamental phases, prioritization of tasks and selection of processor (server) for the task. However, for our HEC-Cluster technique, we do not need to prioritize or check if there are available nodes for processing tasks according to these requirements.

Thus, in this simulation, we prove that in addition to being more efficient in terms of clustering, the HEC-Clustering Balance also allows doing the hierarchical load balancing technique and has a perfect efficiency according to the successful processing of requests sent by the terminals.

To evaluate our proposed solution, we used two types of traffic, namely the Montage data flow (Fig. 10 and Fig. 12) and the CyberShake data flow (Fig. 11 and Fig. 13). The Montage data flow is an astronomy application used to build large mosaics of sky images. In other words, it uses image data streams. The CyberShake data flow is a seismology application that calculates Probabilistic Seismic Hazard curves for geographic sites in the Southern California region.

However, as our solution considers two techniques (i.e. clustering and hierarchical load balancing methods) that were offered separately, the simulation will be done in two parts. Thus, we evaluate our clustering balance algorithm by comparing it among the existing clustering techniques namely Vertical Clustering, Horizontal Clustering, and HEFT. On the other hand, we evaluate our proposal with the basic load balancing techniques namely First Come First Serve (FCFS), MaxMin, MinMin, Round Robin (RR) and with the latest load balancing techniques such as ALBOA, CooLoad.

For the Montage workflow, the system sent image-type queries and the simulation repeated for 25 tasks, 50 tasks, 100 tasks and finally 1000 tasks to see the behavior of each algorithm. Also for the CyberShake workflow, the simulation was performed for four times for 30, 50, 100 and 1000 requests.

Thus, Tables 2, define the configurations of data centers, Hosts and possible virtual machines (VM) in the WorkFlowSim environment.

The results obtained in Fig. 10 and Fig. 11, show that the HEC-Clustering Balance (Algorithm 2) offers better performance in terms of latency for the clustering the reception of a user task.

On these results, the proposed clustering method is not only more effective in terms of delay but also remains relatively ascertained as the number of tasks increases. This is due to the hierarchical load balancing technique that is considered by HEC-Clustering Balance method in case of an overload of the cluster nodes. Unlike other techniques, like Vertical Clustering (in the yellow band in Fig. 10), with 25 tasks, it gives a better response time, but as the tasks increase, we see that the delay increases considerably. Finally, the proposed method, which enables clustering and load balancing, reduces the execution time of tasks. It also allows for a relatively constant time regardless of the number of tasks that HEC architecture must handle.

In Fig. 11, we can see the evolution of our HEC-Clustering algorithm compared to other techniques using the CyberShake data flow. We find that our method offers much more performance in terms of calculation time with the increase in the number of tasks compared to other methods. Besides, beyond 50 tasks, the computation time tends to decrease with the evolution of the number of requests. Finally, for the functionality of clustering, we can see that the efficiency of our HEC-Clustering algorithm is independent of the type of traffic used.

Moreover, in Table 3, we can see the effectiveness of each clustering technique according to the number of requests received by the system (in percentage). For example, about 100 requests sent for each technique, Horizontal Clustering processed 72 (with a loss rate of 28%), Vertical Clustering 88 (with a loss rate of 12%). Additionally, HEFT and HEC-Clustering offer perfect efficiency (0% loss). It can be assumed that both techniques have the same efficiency (100%) but the difference lies in the calculation time (Fig. 10).

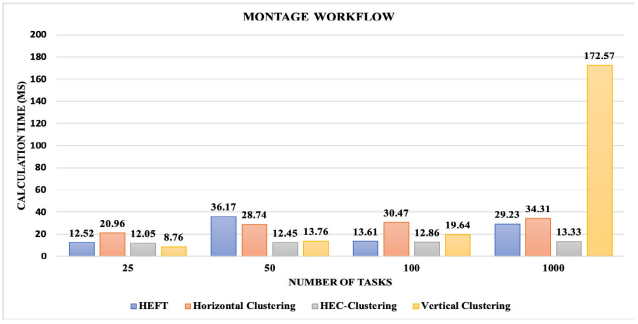


FIGURE 10. Measuring calculation time based on multiples clustering methods with Montage.

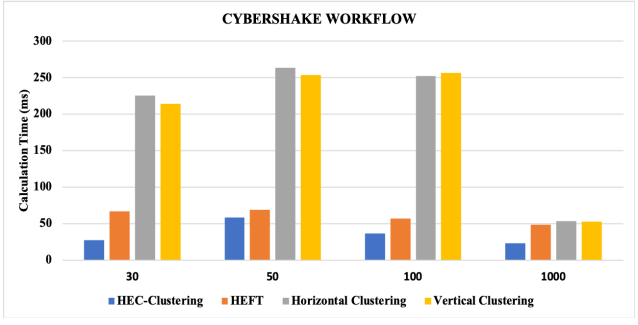


FIGURE 11. Calculation time based on multiples clustering methods with CyberShake.

The results obtained in Fig. 12 and Fig. 13 based respectively on the Montage and CyberShake data flows show that the evaluation of HEC-Clustering offers better performances compared with the other techniques load balancing in terms of execution cost according to service time. In other words, in Fig. 12, with the Montage workflow, there is a stabilization of HEC-Clustering with an increase in the number of tasks sent by users. This is explained by the fact that at a certain occupancy rate of the HEC server, the tasks will be balanced either towards the least saturated servers or else towards the higher level.

Thanks to our solution, we have reduced over 73% on the processing time of requests compared to the standard architecture of HEC [3]. In Fig. 13, with the CyberShake workflow, we have better performances than the other techniques. In addition, in this figure, we have seen an increase in treatment time for 30 and 50 tasks. From 100 tasks, the cost of execution has relatively decreased. This is due to the presence of the hierarchical load balancing functionality in HEC-Clustering. Thus for this type of traffic, the response time has also been reduced to 19%.

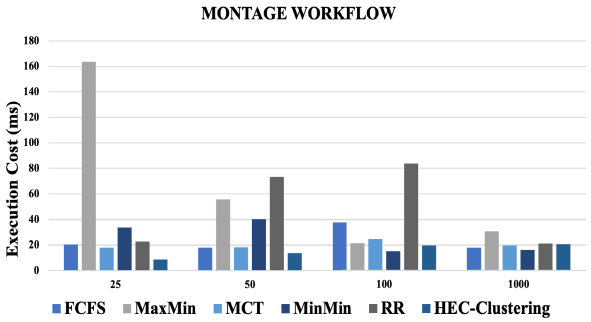


FIGURE 12. Execution cost based on load balancing techniques with Montage workflow.

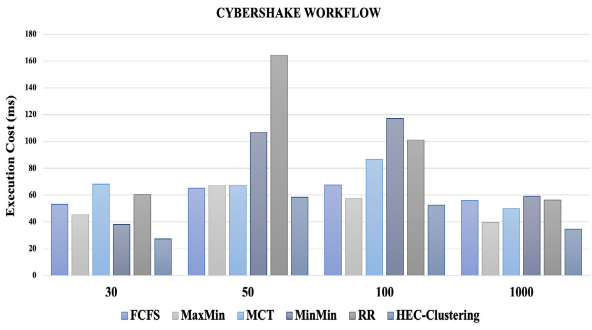


FIGURE 13. Execution cost based on load balancing methods with CyberShake workFlow.

Fig. 12 and Fig. 13 allow us to compare the basic load balancing methods with our technique (HEC-Clustering Balance). Even, if HEC-Clustering shows its efficiency on these basic algorithms, it would be more judicious to compare them with the most recent load balancing techniques. Thus, in Fig.18, we compare the execution time on HEC-Clustering with two of the latest load balancing techniques, namely CooLoad [6] and ALBOA [37]. These results show that the HEC-Clustering Balance remains effective in processing time compared with CooLoad and ALBOA.

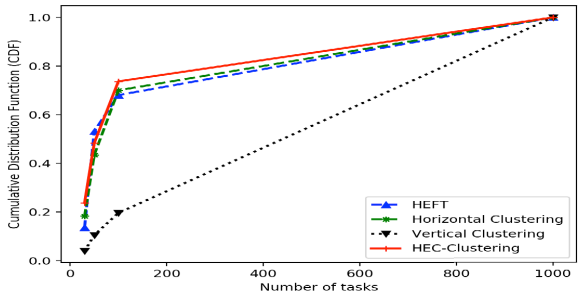


FIGURE 14. Cumulative Distribution Function with Montage for clustering methods.

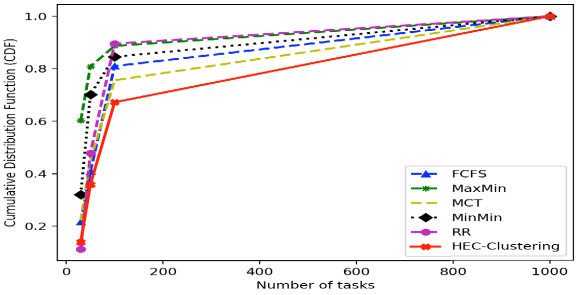


FIGURE 15. Cumulative distribution function with Montage for load balancing methods.

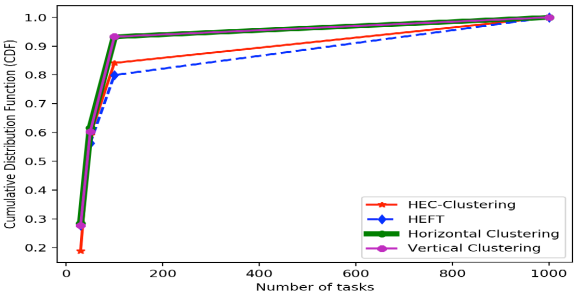


FIGURE 16. Cumulative distribution function with Cybershake for clustering methods.

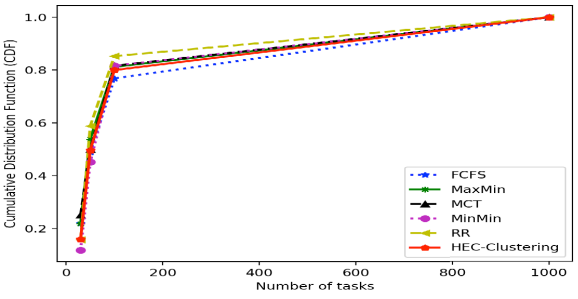


FIGURE 17. Cumulative distribution function with Cybershake for load balancing methods.

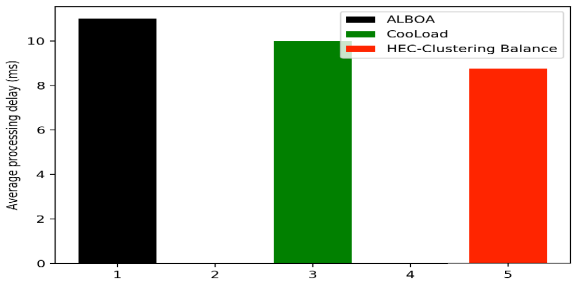


FIGURE 18. Comparison of the different load balancing techniques in distributed systems.

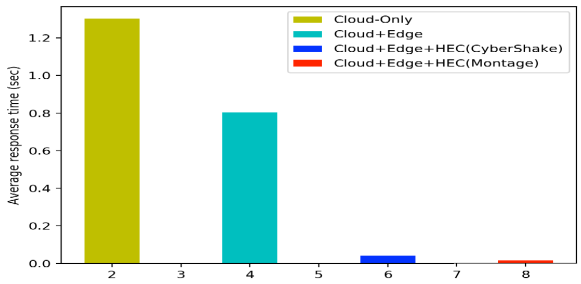


FIGURE 19. Average response time of workloads in the different types of clouds.

Also, Fig.19, we show the efficiency of using a three-tier distributed cloud (HEC) architecture regardless of the type of traffic used. In other words, our system (Cloud+Edge+HEC), whatever the type of traffic used, reduces the delay compared to existing Edge systems (Cloud+Edge) and centralized cloud (Cloud-Only).

However, in our simulation tool, i.e., the WorkflowSim, we use Weibull distribution for the Cumulative Distribution Function (CDF). Weibull's distribution is used to analyze the reliability of the simulation. In our simulation, using the Weibull distribution technique, we can know, among the numbers of requests sent (between 0 and 1,000 requests), those that are processed most often. This also allows us to conclude over which time interval most requests are processed. According to Weibull's distribution on WorkflowSim, we calculate the CDF with the techniques used for our simulation. With CyberShake and Montage data flows, simulation is performed for clustering (Fig. 14, Fig. 16) And Balancing (Fig. 15, Fig. 17) techniques. In these figures, with the use of Cybershake and Montage as data flow, the Weibull distribution function allows us to know, in our simulation with WorkflowSim, that the frequency of requests sent is greater between 0 and 200 tasks out of the 1,000 tasks. In other words, the distribution is greater, in this interval out of a total capacity for sending 1,000 tasks users. After an analysis done on Fig. 14, Fig. 15, Fig. 16 and Fig. 17, we can see that the Weibull distribution follows a rule which does not depend on the data used but on the basic parameters of this function. Besides, through the results of our simulation, with the Weibull distribution, the average execution time for the different techniques is between 0 and 40 ms.

In summary, thanks to our simulation, we see that our solution showed its effectiveness compared with the other techniques of clustering and also load balancing techniques.

TABLE 3. Number of requests successfully processed according to the clustering techniques used.

Clustering Methods	Efficiency about response requests (%)
HEC-Clustering	100
Horizontal Clustering	72
Vertical Clustering	88
HEFT	100

VI. CONCLUSION AND FUTURE WORK

In this paper, the main idea was to propose a clustering system called HEC-Clustering Balance based on the new Edge Computing architecture HEC to solve the latency problems related to the limitation of HEC server resources [3]. Moreover, taking into account the overloading of these servers, a hierarchical load balancing system has also been integrated with this HEC-Clustering method, which allowed in this simulation to have, whatever the number of tasks, a delay that is approximately constant compared to integrated existing techniques.

For the next step, we will ask the following question: how to centralize and automate this resource management? To answer this question, we will focus on the integration of Software Defined Network (SDN) into the Home Edge Computing (HEC) architecture.

REFERENCES

[1] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, "A survey on low latency towards 5G: RAN, core network and caching solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3098–3130, 4th Quart., 2018.

[2] N. Akkari and N. Dimitriou, "Mobility management solutions for 5G networks: Architecture and services," *Comput. Netw.*, vol. 169, Mar. 2020, Art. no. 107082.

[3] C. S. M. Babou, D. Fall, S. Kashihara, I. Niang, and Y. Kadobayashi, "Home edge computing (HEC): Design of a new edge computing technology for achieving ultra-low latency," in *Proc. Int. Conf. Edge Comput.* Cham, Switzerland: Springer, 2018, pp. 3–17.

[4] C. S. M. Babou, B. O. Sane, and I. Niang, "A hierarchical method for dynamic job execution in NREN-based cloud systems," in *Proc. 2nd Int. Conf. Netw., Inf. Syst. Secur. (NISS)*, 2019, pp. 1–6.

[5] C. S. M. Babou, B. O. Sane, I. Diane, and I. Niang, "Home edge computing architecture for smart and sustainable agriculture and breeding," in *Proc. 2nd Int. Conf. Netw., Inf. Syst. Secur. (NISS)*, 2019, pp. 1–7.

[6] R. Beraldi, A. Mtibaa, and H. Alnuweiri, "Cooperative load balancing scheme for edge computing resources," in *Proc. 2nd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, May 2017, pp. 94–100.

[7] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–37, Sep. 2019.

[8] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proc. 10th ACM Int. Conf. Distrib. Event-Based Syst. (DEBS)*, 2016, pp. 258–269.

[9] B. Varghese, N. Wang, J. Li, and D. S. Nikolopoulos, "Edge-as-a-service: Towards distributed cloud architectures," 2017, *arXiv:1710.10090*. [Online]. Available: http://arxiv.org/abs/1710.10090
[10] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 416–464, 1st Quart., 2018.
[11] R. Kolcun, D. Boyle, and J. A. McCann, "Optimal processing node discovery algorithm for distributed computing in IoT," in *Proc. 5th Int. Conf. Internet Things (IoT)*, Oct. 2015, pp. 72–79.
[12] C. T. Do, N. H. Tran, C. Pham, M. G. R. Alam, J. H. Son, and C. S. Hong, "A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2015, pp. 324–329.
[13] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, 2014.
[14] M. A. Hassan, M. Xiao, Q. Wei, and S. Chen, "Help your mobile applications with fog computing," in *Proc. 12th Annu. IEEE Int. Conf. Sens., Commun., Netw.-Workshops (SECON Workshops)*, Jun. 2015, pp. 1–6.
[15] R.-A. Cherrueau, D. Pertin, A. Simonet, A. Lebre, and M. Simonin, "Toward a holistic framework for conducting scientific evaluations of OpenStack," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2017, pp. 544–548.
[16] R. Morabito, "Virtualization on Internet of Things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.
[17] I. Farris, T. Taleb, H. Flinck, and A. Iera, "Providing ultra-short latency to user-centric 5G applications at the mobile network edge," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 4, p. e3169, Apr. 2018.
[18] S. Ningning, G. Chao, A. Xingshuo, and Z. Qiang, "Fog computing dynamic load balancing mechanism based on graph repartitioning," *China Commun.*, vol. 13, no. 3, pp. 156–164, Mar. 2016.
[19] K. E. Parsopoulos and N. M. Vrahatis, "Particle swarm optimization method for constrained optimization problems," *Intell. Technol.-Theory Appl., New Trends Intell. Technol.* vol. 76, no. 1, pp. 214–220, 2002.
[20] D. Puthal, M. S. Obaidat, P. Nanda, M. Prasad, S. P. Mohanty, and A. Y. Zomaya, "Secure and sustainable load balancing of edge data centers in fog computing," *IEEE Commun. Mag.*, vol. 56, no. 5, pp. 60–65, May 2018.
[21] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," in *Internet of Things*. San Mateo, CA, USA: Morgan Kaufmann, 2016, pp. 61–75.
[22] H. Liu, "Self-duality in quantum K-theory," 2019, *arXiv:1906.10824*. [Online]. Available: http://arxiv.org/abs/1906.10824
[23] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.
[24] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in fog-cloud computing paradigm," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 1222–1228.
[25] S. Li, N. Zhang, S. Lin, L. Kong, A. Katangur, M. K. Khan, M. Ni, and G. Zhu, "Joint admission control and resource allocation in edge computing for Internet of Things," *IEEE Netw.*, vol. 32, no. 1, pp. 72–79, Jan. 2018.
[26] A. A. Alsaffar, H. P. Pham, C.-S. Hong, E.-N. Huh, and M. Aazam, "An architecture of IoT service delegation and resource allocation based on collaboration between fog and cloud computing," *Mobile Inf. Syst.*, vol. 2016, pp. 1–15, Aug. 2016.
[27] D. Kelaidonis, A. Rouskas, V. Stavroulaki, P. Demestichas, and P. Vlacheas, "A federated edge cloud-IoT architecture," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2016, pp. 230–234.
[28] S. Maheshwari, D. Raychaudhuri, I. Seskar, and F. Bronzino, "Scalability and performance evaluation of edge cloud systems for latency constrained applications," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 286–299.
[29] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee, "A container-based edge cloud PaaS architecture based on raspberry pi clusters," in *Proc. IEEE 4th Int. Conf. Future Internet Things Cloud Workshops (FiCloudW)*, Aug. 2016, pp. 117–124.
[30] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. Veh. Technol.*, vol. 66, no. 4, pp. 3435–3447, Apr. 2017.
[31] X. Xu, S. Fu, Q. Cai, W. Tian, W. Liu, W. Dou, X. Sun, and A. X. Liu, "Dynamic resource allocation for load balancing in fog environment," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–15, Apr. 2018.
[32] M. Bouet and V. Conan, "Mobile edge computing resources optimization: A geo-clustering approach," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 2, pp. 787–796, Jun. 2018.
[33] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2017, pp. 1–6.
[34] W. Chen and E. Deelman, "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments," in *Proc. IEEE 8th Int. Conf. E-Sci.*, Oct. 2012, pp. 1–8.
[35] N. Chopra and S. Singh, "HEFT based workflow scheduling algorithm for cost optimization within deadline in hybrid clouds," in *Proc. 4th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2013, pp. 1–6.
[36] W. Chen, R. F. da Silva, E. Deelman, and R. Sakellariou, "Using imbalance metrics to optimize task clustering in scientific workflow executions," *Future Gener. Comput. Syst.*, vol. 46, pp. 69–84, May 2015.
[37] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2092–2104, Feb. 2020.

CHEIKH SALIOU MBACKE BABOU received the M.E. degree in networks, systems and telecommunication from University Cheikh Anta Diop (UCAD), Senegal, in 2016, where he is currently pursuing the Ph.D. degree. He is also a special Research Student with the Nara Institute of Science and Technology (NAIST). His research interests include edge computing, cloud computing, resources management, quality of service (QoS), 5G technology, and optimization of networks and systems. He received the Best Paper Award in EDGE2018, USA.

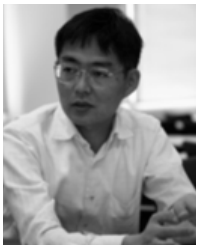
DOUDOU FALL received the M.E. degree in data transmission and information security from Cheikh Anta Diop University, Senegal, in 2009, and the M.E. and Ph.D. degrees in information science from the Nara Institute of Science and Technology (NAIST), Japan, in 2012 and 2015, respectively. He is currently an Assistant Professor with the Division of Information Science, NAIST. His research interests include cloud computing security, the IoT security, blockchain security, vulnerability, and security risk analysis.

SHIGERU KASHIHARA (Member, IEEE) received the Ph.D. degree in engineering from the Nara Institute of Science and Technology (NAIST), Japan, in 2003. He worked with the Kyushu Institute of Technology, in 2004. He has been with NAIST, since 2005. In 2010, he was a Visiting Researcher with the University of California, Los Angeles. He is currently an Associate Professor with the Faculty of Information Science and Technology, Osaka Institute of Technology–Hirakata, Japan. His current research interest includes cyber-physical-human systems based on multidisciplinary aspects.

YUZO TAENAKA (Member, IEEE) received the D.E. degree in information science from the Nara Institute of Science and Technology (NAIST), Japan, in 2010. He was an Assistant Professor with the University of Tokyo, Japan. He has been an Associate Professor with the Laboratory for Cyber Resilience, NAIST, since April 2018. His research interests include information networks, cybersecurity, distributed systems, and software defined technology.

MONOWAR H. BHUYAN (Member, IEEE) received the Ph.D. degree in computer science and engineering from Tezpur University, in 2014. He is currently an Assistant Professor with the Department of Computing Science, Umeå University, Sweden, since January 2020, and one of the research group leaders at Autonomous Distributed Systems Laboratory. Before this, he worked with the Nara Institute of Science and Technology, Japan, Umeå University, Assam Kaziranga University, India, and Tezpur University, India, from January 2009 to December 2019. His research areas include machine learning, anomaly detection, security and privacy, and distributed systems.

IBRAHIMA NIANG received the degree in computer systems engineering from the Polytechnic Institute of Kharkov, Ukraine, in 1994, and the Ph.D. degree in computer science from the University of Paris Réné Descartes, in 2002. He was a Full Professor from University Cheikh Anta Diop, Senegal, in 2018. His research interests include quality of service (QoS) management, mobility. and optimization of networks and systems. Concretely, he had to work on sensor networks and Pair to Pair systems. His current interests include cloud-edge computing systems, with the Internet of Things.



YOUKI KADOBAYASHI (Member, IEEE) received the Ph.D. degree in computer science from Osaka University, Japan, in 1997. He is currently a Professor with the Laboratory for Cyber Resilience, Nara Institute of Science and Technology, Japan. His research interests include cybersecurity, Web security, and distributed systems. He is a member of the ACM and the IEEE Communications Society.