# Real-Time Scheduling in Drop Computing

Silvia-Elena Nistor[1], George-Mircea Grosu[1], Raluca-Maria Hampau[1], Radu-Ioan Ciobanu[1], Florin Pop[1,2], Ciprian-Mihai Dobre[1,2], Paweł Szynkiewicz[3]
[1]*University Politehnica of Bucharest, Romania*
[2]*National Institute for Research and Development in Informatics (ICI) Bucharest, Romania*
[3]*Research and Academic Computer Network (NASK), Kolska 12, 01-045 Warsaw, Poland*
silvia_elena.nistor@stud.acs.upb.ro, george_mircea.grosu@stud.acs.upb.ro, raluca_maria.hampau@stud.acs.upb.ro, radu.ciobanu@cs.pub.ro, florin.pop@cs.pub.ro, ciprian.dobre@cs.pub.ro, florin.pop@ici.ro, ciprian.dobre@ici.ro, pawel.szynkiewicz@nask.pl

***Abstract*—The IoT world evolves at a tremendously fast rate, leaving some of the currently employed technologies behind. The modern day user has more and more expectations: high availability and performance, low energy consumption and high privacy standards. If all of these things were achievable a decade ago, the world is now witnessing an explosion of interconnected, mobile devices, which consequently generate big volumes of data. In addition to this, an exponential increase in real-time based applications fuels even more the end users' demands. Edge and Fog computing paved the way to a new, different approach in the light of delegating not only the Cloud for computational and storage resources. By bringing processing resources closer to the source, a completely different perspective was offered to the research world. The emergence of these new models revealed the need for different approaches and marked the beginning of a major paradigm shift. Drop Computing expands the horizon of the previous technologies to another level, by introducing a decentralized model, based on opportunistic networks and geographical co-locations. In this paper, we aim to optimize the solution by introducing new scheduling strategies. We addressed the proliferation of real-time applications by shaping deadline focused algorithms and we analysed the results to evaluate the system's behaviour under various constraints.**

***Index Terms*—Real-time Scheduling, Drop Computing, Dynamic ad-hoc Opportunistic Networks, Edge Devices, Mobile Devices.**

## I. Introduction

For the past decade we have been witnessing a tremendous expansion of the Internet of Things (IoT), which inevitably triggered greater demands and raised expectations amongst users. Nowadays, devices are expected to present advanced functionalities that can exceed local computational capabilities, in order to deliver a pleasant user experience. An ordinary user has become accustomed to augmented reality, facial recognition, mobile payments, advanced navigation capabilities and synchronization with home environments. Aspects such as latency sensitivity, reliability and power consumption change the perspective from: "We want things done" to: "We want things done *fast* and *efficient*" [16] [19].

To a certain degree, the cloud has managed to meet the requirements for a higher quality of service (QoS) and helped devices when their local resources were insufficient. The cloud vision is based on shifting a part of the storage and tasks processing away from local devices towards remote, over the Internet premises. Without a doubt, this paradigm shift has revolutionised the IT world and expanded the previous possibilities. However, the fact that mobile devices rely heavily on the cloud, marked a substantial increase in network overhead and revealed a concerning issue regarding the sustainability of the traditional approach. With so many devices accessing the cloud data centers, one must wonder if cloud will be enough in the foreseeable future. Technologies such as Edge Computing and its related Mobile Edge Computing, Fog Computing or CloudLets are viable alternatives with the central purpose of bringing the cloud closer to the user. However, these solutions introduce new challenges, such as privacy or reliability concerns.

This paper proposes a scheduling method for Drop Computing - a paradigm shift towards a new model of decentralized computing over multilayered networks, based on dynamic ad-hoc opportunistic networks. Drop Computing combines cloud and wireless technologies over a social crowd, formed between mobile and edge devices and leverages the extended resources for improved performance and efficient access. Therefore, before accessing the Cloud, geographically co-located devices are designated to perform the required tasks and the cloud becomes a back-up plan. In this manner, Drop Computing saves energy by reducing the number of unnecessary accesses to cloud and compensates for the high cost of off-the-premise rented resources [4]. We identified one vital aspect that must be addressed in drop computing systems: scheduling. Resource allocation is a sensitive and crucial component in designing distributed systems and in this paper we will discuss various models and strategies of scheduling tasks, both locally, on the device, and globally, between devices [18].

For this purpose, we have extended the MobEmu simulator (see Section IV) and extracted the results. The analysis conducted in this paper adds a real-time component, relevant for the use case detailed in Section V-A. We are particularly interested in the real-time dimension added to the opportunistic network, as such applications are proliferating in the IoT world. The objective is to evaluate the behaviour and performance of the system, depending on each scheduling strategy when real-time tasks are introduced in the data flow.

In terms of privacy and security, an interesting discussion can be had here. Since there is no entity that can act as a central authority, trust and reputation mechanisms (such as

SAROS [3]) should be employed, so that only trustworthy non-malicious nodes are used as relays or for computing tasks. The advantage of a paradigm such as Drop Computing, which is based on social connections and human interaction, is that some trust information is implicitly know through the online social networks (i.e., I can trust a node that I am friends on Facebook with, because I know that person).

The rest of this paper is structured as it follows. Section II of this paper presents other related technologies in more detail - Fog, Edge, Mobile Edge Computing and how Drop Computing complements them. A proposed use case will be outlined for a better understanding of what will be addressed afterwards. Section III zooms in the attention span and centers our view on scheduling algorithms, firstly from a general perspective and then in Drop Computing context. Different strategies will be addressed [9] [17], but in order to support the use case, scheduling real-time tasks will be the main concern. Section IV aims to support the theoretical background presented in the previous chapter with a practical approach, by implementing the aforementioned general algorithms in MobEmu simulator. After introducing the reader with a brief description of the simulator, our chosen path of implementation will be addressed. The experimental results will also be presented in this section. Section V-A puts everything that we discussed in the light of the chosen use case. This is where the scenario will be detailed and the authors will have a clear view of how the concept translates in real life. Section V-B summarizes all of our results in order to establish if the objectives were met. We also present conclusions and our vision of the future work in this project in Section VI.

## II. Background

Internet has become more than just a communication medium, reaching the proportions of a large computation platform, connecting billions of devices worldwide - an estimated number of 23-24 billion connected devices. Because of this technological explosion, the Quality of Experience (QoE) and implicitly Quality of Service (QoS) have become key components in the service providing industry. The number of real-time applications is on the rise and metrics such as latency, throughput and jitter are closely monitored for future improvement.

Without the arrival of Cloud computing, applications that now an ordinary user is accustomed to, could not have materialized: social networking, gaming, real-time video streaming (Skype, Netflix), IoT data processing. Cloud made a breakthrough in processing Big Data, becoming one of the most relevant means of offloading data computation. The National Institute of Standards and Technology extracted the most notable characteristics of Cloud: on-demand self-service, broad network access, resource pooling, rapid elasticity, measured service.

However, this centralized computational system faces big challenges: more and more heterogeneous devices demand access to it, leading to a serious negative impact not only on latency and integrity of data, but also on maintenance costs. Data-intensive applications that have QoS requirements such as time constraints or energy consumption and the rising number of users sending data, represent a major threat to the Cloud model, which justifies the need for a paradigm shift [5] [2].

In the light of this abrupt road towards higher maintenance costs, scalability disparity, power consumption and high latency, the alternatives did not cease to appear. Edge, Mobile Edge and Fog Computing were born with the purpose of expanding the horizon of cloud services.

Fog Computing caters the needs of the users by enabling data processing closer to the devices, but not exclusively on edge devices, as it happens in Edge Computing. The term ”fog” originates from the idea that fog is a cloud closer to the ground [1] and reiterates the act of leveraging geographical proximity. The main focus points for Fog Computing consist of mobility, location awareness, low latency, heterogeneity and support for large number of devices. In terms of location for data processing, fog is more relaxed than edge computing and data can be processed on any device connected to the same LAN, not necessarily on the device or sensor itself [1] [15]. Fog Computing represents the big picture, which includes Edge Computing, the next discussed platform.

Edge Computing suggests processing data at the edge of the network, closer to where the data originated from. The term edge refers to any resource with computational capabilities that happens to be on the path between the source and the cloud servers, usually closer to sensors or actuators [13]. Therefore, edge devices have a double role: they can both produce and process data. The responsibilities range from storage to task processing or even acting as middleman between the source and cloud. The key is to leverage the computational capabilities within the local network, without flooding the cloud with requests. Of course, cloud will still be available to do the ”heavy lifting”, but it will not be so aggressively solicited. Hence, we can identify a major advantage of this approach: the number of accesses to cloud is reduced and data processing is significantly faster. In terms of real-time restrictions, edge computing is an excellent candidate. Another notable benefit of EC involves the security aspect - data is contained closer to the source, without being transferred over the internet [9], [10] [20].

Multi-access Edge Computing (MEC) is a subset of Edge Computing, advocating the same principles, applied to mobile networks. MEC is a relatively new technology, standardized by ETSI, that provides computing capabilities at the edge of the cellular network. In this case, location awareness techniques occur inside Radio Access Networks (RAN) and closer to mobile subscribers. Article [14] depicts various scenarios, such as augmented reality, intelligent video acceleration, connected cars or IoT gateway where MEC proves to be applicable. Once more, a tighter collaboration between co-located nodes seems to be the right answer to cellular congestion.

Fog, Edge and Mobile Computing have less processing and storage capabilities, making them more suitable for latency-sensitive applications. However, the downtime is reduced, since data is distributed to proximity nodes and continuous Internet access is not required, making these platforms much more secure than the cloud. An interesting point of view was made in [15], regarding the impact of man-in-the-middle attack, which can become a frequent threat in fog infrastructure. The results were satisfactory and the attack showed minimal impact on CPU and memory usage.

With the emergence of these new platforms, it is clear that we are witnessing a migration from the traditional cloud computing towards collaborative, device-centered computing.

## III. Real-time Scheduling in Drop Computing

### A. A general view of scheduling problem

We consider scheduling in Drop Computing, because we aim to have a better control over how the resources are allocated between the nodes. This decision highly impacts the parameters of the system and after repeatedly stating the importance of efficiency and low latency in the future of IoT, optimizing and calibrating system parameters quantify as a first priority. In a large scale decentralized architecture, where big data is transferred at an unprecedented rate, we meet heterogeneous nodes and computational demanding workloads, often accompanied by various constraints. The scheduler is vital for the reliability, availability and performance of the system. In a decentralized architecture, the probability of disproportion between nodes is relatively high. Some nodes can be almost inactive, or really ”selfish”, while other nodes are extremely loaded. The term ”selfish” is relevant in a socially connected crowd and it reveals the social status of the device - in this case, the node is not willing to participate and help other neighbours with their tasks. On the opposite side, an ”altruistic” node accepts many tasks offloaded by other devices, contributing to the welfare of the system. This is where load balancing steps in, or as we will call it: *global scheduling*. On the other hand, *local scheduling* refers to the local ”administrative issues”, on the premises of a single node [6]. Our work evolves in a dynamic, collaborative architecture, with little to no a priori knowledge of the environment. For the case of local scheduling this aspect is not problematic, since each node has a full understanding of its own resources. However, on the global scheduling, as far as a device is concerned, it can meet almost any type of device in its proximity, in a variable length social crowd. Therefore, the device cannot make well informed decisions and it may sometimes need to adapt its choices.

### B. Scheduling Real - Time tasks in Drop Computing

There is an undeniable correlation between QoE and real-time features in devices these days. Mobile app developers invest in real-time applications and they have good reasons to do so. In just 24 hours: 24 billion Whatsapp messages are sent, 2 billion photos and 250 millions videos, 3 billion Facebook videos are watched, 760 million Snapchat pictures are shared, 99 million hours of Youtube are watched, 80 million Instagram photos are uploaded.

Real-time integration gives users the sense of live participation and interaction with multiple instances. Chatting with your relatives and friends while booking a flight and listening to your favourite music has never been easier. This technology allows us to engage in multiple activities without being aggressively extracted from the real world. It blends so well with the world around us, that the users have become almost addicted to it, which is why real-time integration is indeed a basic requirement. Tasks with deadlines can be divided into three categories [8] [11]:

- *critical tasks* or applications with hard deadlines - these are tasks that must be completed before the deadline under any circumstances, otherwise the results will undoubtedly be catastrophic. In aviation or military applications, critical tasks are often met and their negative outcome is a potential human-life threat;
- *essential tasks* or applications with firm deadlines - these tasks can exceed the deadline without such disastrous implication. Nevertheless, the performance of the system will seriously be damaged;
- *non essential tasks* or applications with soft deadlines - these tasks can exceed their deadlines without any significant impact on the overall performance.

When referring to real-time tasks in our research, we mainly focus on critical and essential tasks. However, at the current state of development the essential tasks will be more prevalent, especially in the context of our use cases.

### C. Scheduling Model

Our approach of real-time dimension in the simulator will be similar to the concept of streaming [7]. One large task is split into smaller, sorted tasks that are sent from device to device (multi-hop transmission) until they reach their destination. There, tasks are once again be sorted, to ensure their continuity and later executed as other tasks are being received. The main concern is to keep the sequence of tasks in the original order, even if some packet loss occurs. File downloading for example, is not a viable option when discussing real-time scenarios because not only it forces the receiver to wait for the full download of the file, but also consumes valuable storage resources.

What distinguishes real-time tasks from ordinary tasks is a certain set of constraints that are enforced [7]:

- they have priority in execution, because they need to arrive faster at the destination
- they must be received in the order of transmission
- once lost, they are lost forever - there is no retransmission
- once late, they are useless

This strategy is a best effort strategy, which means that the service does not provide any guarantees regarding transmission rate. Packet loss can occur at any time even if we do not send packets over the internet - in wireless channels for example, radio frequency interference can affect data transmission. We envisioned this model as a User Datagram Protocol-like model:

- connection-less communication - devices source and destination do not have to establish a connection previous to starting the actual data transmission;
- delivery is not guaranteed;
- no flow control and re-transmission mechanism.

This means of data dissemination exhibits a high level of flexibility in choosing the type of communication. For the purpose of this paper, only unicast requests were tested, but, support for multicast and broadcast is remarkably easy to add. Since packets are moved from one node to another, each node can decide if it should accept the request or not. In unicast, only one node will accept it, but in multicast or broadcast, this constraint can be relaxed.

In order to understand the problem of global scheduling, we first need to assess the initial situation of our resources as it follows. We have a set of $n$ nodes, $N = \{N_0, N_1, ..., N_{n-1}\},\ n = |N|$ and each node generates a set of $t$ tasks, $T_{node} = \{T_0, T_1, ..., T_{t-1}\},\ t = |T_{node}|$. Tasks can be divided into three main types of tasks: $\{SMALL, MEDIUM, LARGE\}$, but the medium and the large tasks can also be regarded as real-time tasks.

**Global scheduling** aims to distribute as fairly as possible all the tasks from node $A$ to $m$ nodes from $N$, where $m \leq n$, $m = |F|$: $F = \{node \in N \mid node\ isCloseTo\ A\}$, where *isCloseTo* is a function that determines whether a node is in the proximity of another node and if the neighbourhood requirement is met and *F* is the set of neighbours for node A.

Due to the mobility of the nodes, the $m$ value is variable and depends on the exact location of each node at a certain moment of time. The closeness condition is met when two nodes met more than two times during the simulation trace.

In the case of **local scheduling**, tasks must be assigned to processors. We have a node, which has a set of tasks $T$ to execute - its own or from other nodes, it does not matter: $T = \{T_0, T_1, ..., T_{t-1}\},\ t = |T|$. These tasks must be assigned to the processor of the node $A$: $P = \{P_0, P_1, ..., P_{p-1}\},\ p = |P|$.

Regardless of the scheduling strategy, real-time tasks will always have priority in front of other tasks. When dealing with local scheduling, a frequently encountered set of metrics is usually applied, that help us assess the performance of the system:

- arrival time, $a(T_i)$ - when a task enters into ready state;
- burst time, $b(T_i)$ - represents the total time needed for the execution of the task;
- completion time, $c(T_i)$ - when a task completes its execution;
- turnaround time, $t(T_i)$ - total time spent by the task from its arrival to its completion;
- waiting time, $w(T_i)$ - total time spent by the task while waiting for CPU in ready queue.

### *D. Scheduling Algorithms*

**Global scheduling**. The global scheduling policy prevents nodes from having disproportionate workloads. Tasks are spread evenly throughout the social crowd by means of different strategies. Our global scheduler has two components: balancer and publisher.

The balancer is used when two devices make contact and exchange information. If the nodes are not balanced in terms of workload, than balancing algorithms remedy the situation. The publisher is called when the current node needs to decide where to send the generated tasks. In this case, there is no need for a specific encounter with another node. This dissemination technique is essential for keeping the data in continuous movement, otherwise the tasks might not be executed on time.

For the publisher component, we identified three algorithms suitable for our needs: Default Publisher (cf. Alg. 1), Bidding Publisher (cf. Alg. 2), Real-Time Publisher (cf. Alg. 3) - as latency is such an important factor in real-time transmission, flooding all the neighbours with tasks can slow down the communication.

The goal of these algorithms is to offer a comparison and analysis support for different scheduling strategies, in the context of Drop Computing. We aim to address well-known scheduling algorithms and investigate their behaviour in the proposed paradigm. Therefore, the first algorithm presented, Default Publisher, is the naive implementation of the global scheduling, in which all tasks are disseminated through all the neighbours. In this case, the overhead is worst-case scenario, with maximum level. Bidding Publisher aims to improve the overhead and splits the tasks between neighbours only if they are willing to participate with their resources in the computation process. Once the communication overhead has been reduced, Real-Time Publisher prioritises the real-time tasks in the Bidding Publisher process. In this algorithm we generate a random number of nodes to deliver the real-time tasks, no more than a quarter of the total number.

**Algorithm 1** Default Publisher

```
1: $currentTasks \leftarrow list\ of\ tasks\ to\ be\ disseminated$
2: for $neighbour = 1, 2, \ldots$ do
3:     $neighbour.tasks.add(currentTasks)$
4: end for
```

For the balancing component, five algorithms were implemented: Bidding Balancer, Broadcast Balancer, Initial Time Balancer and Remaining Time Balancer.

We followed the model presented by K. Ramamritham, J. A. Stankovic, and W. Zhao, in [12] and respected their taxonomy. When local scheduling fails, a collaborative alternative is looked for. In this respect, four types of algorithms were implemented by us:

- random scheduling - we send the task to a random node in our social network. The approach is easy to implement, but redundant transmission might occur.
- focused addressing - we send the task to nodes that are most rich in resources and increase the probability of solving the task before its due-date.
- bidding - two components work closely together for this strategy: the manager and the contractor [6]. Managers

**Algorithm 2** Bidding Publisher

```
1: $tasksMap \leftarrow map\ with\ all\ the\ associations$
   $between\ nodes\ and\ tasks$
2: for $neighbour : neighbours\ of\ current\ node$ do
3:     $initialize\ list\ of\ tasks\ to\ that\ neighbour\ in\ the\ map$
4:     $each\ list\ is\ ordered\ by\ task\ ID$
5: end for
6: for $task : tasks\ of\ current\ node$ do
7:     $maxBid \leftarrow findMaxBid();$
8:     $maxNode \leftarrow maxNodeBid()$
9:     for $neighbour : neighbours\ of\ current\ node$ do
10:        $currBid \leftarrow neighbour's\ bidding\ value$
11:        if $currBid > maxBid$ then
12:            $this\ node\ becomes\ the\ highest\ contractor$
13:        end if
14:    end for
15:    $add\ task\ to\ the\ selected\ contractor's\ list$
16: end for
```

**Algorithm 3** Real-Time Publisher

```
1: $tasksMap \leftarrow map\ with\ all\ the\ associations$
   $between\ nodes\ and\ tasks$
2: for $neighbour : neighbours$ do
3:     $initialize\ list\ of\ tasks\ to\ that\ neighbour\ in\ the\ map$
4:     $real - time\ tasks\ will\ be\ added\ first$
5: end for
6: $this.tasks.sort(real - time\ tasks\ first)$
7: for $task : tasks$ do
8:     $width \leftarrow number\ of\ nodes\ to\ send\ the\ task\ to$
9:     while $width > 0$ and $isRealTime(task) = true$ do
10:        $maxBid \leftarrow findMaxBid();$
11:        $maxNode \leftarrow maxNodeBid()$
12:        for $neighbour : neighbours$ do
13:            $currBid \leftarrow neighbour's\ bidding\ value$
14:            if $currBid > maxBid$ then
15:                $this \leftarrow maxNode$
16:            end if
17:        end for
18:        $add\ task\ to\ the\ selected\ contractor's\ list$
19:        $width \leftarrow width - 1$
20:    end while
21: end for
```

look for offers from contractor nodes and the highest bidding node will receive the task.

- flexible - this algorithm tries to merge the benefits of focused and bidding strategies. If the first one fails, the bidding takes place.

**Local scheduling**. Local scheduling administrates the allocation of resources between processors on a single device. After the global scheduler has completed its job, the local scheduler is in charge with task execution. We implemented a series of algorithms, both for the real-time case and without. The only difference between real-time and non real-time lays in a few set of constraints [7]:

- no matter what algorithm of scheduling is enforced, real-time tasks will always be first in the queue of tasks to execute. This aspect is handled in the comparator - regardless of insertion policy, real-time tasks have priority. The comparator establishes the task importance using the base metric of algorithm in order to sort the execution queue efficiently. For example, if we have First Come First Served (FCFS) algorithm with real-time comparator, the queue will begin with real-time tasks sorted in FCFS order, continuing with the other tasks, in FCFS order.
- real-time tasks cannot be executed on a device that does not represent the destination for the certain task. Real-time tasks will be passed around and only executed on the destination node.
- real-time tasks that arrive too late at the destination will be dropped. For example, suppose device A sends a stream of tasks $\{T_0, T_1, T_2, T_3\}$ to node B. Assuming the last real-time task executed on B, coming from node A is $T_2$, if B receives task $T_1$, $T_1$ will be dropped.

A vastly used taxonomy in scheduling strategies consists of dividing them into two main categories (see Table I):

- preemptive - Round-Robin, Priority Round-Robin, Earliest Due Date (EDD)
- non-preemptive - FCFS, Last Come First Served (LCFS)

The algorithms proposed and implemented in the simulator are: FCFS, Earliest Due-Date (Earliest Deadline First), Round Robin (RR), Priority Round Robin.

The size of quanta in Round Robin algorithm is essential for the outcome of the scheduling and the performance of the system. Most often, the time quanta is chosen between 10 and 100 milliseconds. It is essential not to choose a value too small for the system, because this will result in a waste of CPU time for context switches. On the other hand, if the quanta is too big, smaller processes are forced to wait too long for their turn and the interactivity of the system will be seriously damaged - after all, Round Robin intents to be better than FCFS. However, if the quanta is too small, the number of context switches rise and dramatically impact the performance.

## IV. Modeling using MobEmu

### A. MobEmu simulator

MobEmu is the simulation framework that enabled us to carry out our research. It was especially designed to evaluate opportunistic networks (ON) and to test routing and dissemination algorithms in the context of mobility traces. A more exhaustive analysis upon synthetic mobility models and mobility traces is conducted in [8]. However, for this paper we test our solutions based on the synthetic, social-based models. One of the key aspects of Drop Computing is the human mobility, which is why social-based models are a viable option in this scenario.

The *Trace* implemented in MobEmu is a vital component that contains a list of Contacts and temporal markers such as start time, end time and sample time.

TABLE I
PREEMPTIVE VS. NON-PREEMPTIVE SCHEDULING

| **Criteria** | **Non-Preemptive** | **Preemptive** |
|---|---|---|
| **Definition** | The process acquiring the CPU cannot be interrupted. It will release the CPU after it finishes its execution. | All processes have access to CPU resources only for a limited amount of time. Context switches are enforced by the system and the processes must comply. |
| **Starvation** | A process with high burst-time can monopolize the CPU. | Starvation can occur if processes with high priorities are constantly added in the ready queue. |
| **Overhead** | No overhead | Overhead from context switches |

The *Context* of a node contains the node's id and information about *Topics*. Just like in real life, a node can manifest some particular interest in certain topics. Because we are only interested in simulating the behaviour, topics are not described in such detail - it is enough to be aware of the concept.

For identification, each *Node* has a distinct, unique ID, throughout the duration of the simulation. The storage capabilities of each node are registered in the *dataMemory*. Dissemination algorithms rely on sending data from node to node, therefore we need to know exactly how much data can be stored on a node. It goes without saying that a device with more storage space will have a higher dissemination power, but it can also cause a delay in transmission as the decision making process is more difficult. Bear in mind that this memory refers to memory allocated just for data received from other nodes; for its own data, the node has a different storage area - *ownMessages*. Both these memories are just collections of Message objects. A message that can be exchanged between two nodes has a unique ID, source, destination, the message itself, a timestamp and some tags for the Context.

### B. Extension

The first step in implementing the scheduling algorithms was to define a class for system resources emulation, *SystemData*. *SystemData* objects keep track of CPU, RAM and disk usage with helper methods such as *useRAM*, *useStorage*, *getLoad* or *updateCPUUsage*.

In our simulation, we modified the types of tasks to three categories: SMALL, MEDIUM and LARGE. This taxonomy must be reflected in statistics about the hardware resources. For that, in the *Task* class we added methods that return cycles per instruction, RAM usage and transfer size adjusted to each of those three types. We view small tasks as CPU intensive, medium tasks as a mixture of CPU bound and I/O bound and large tasks as I/O intensive. Thus, a device has a more detailed view upon RAM, ROM, CPU and battery depletion rate.

MobEmu already supported drop computing algorithm, we just added a few features to prepare the infrastructure for the scheduling step. Two key components in MobEmu's initial class *Device* are worth mentioning [4]:

- *onTick* (inherited from parent class *Node*) - describes what a device does at each clock tick. This is where the device generates new tasks if it doesn't have any tasks left for execution and adds them to its task set. Moreover, if the Cloud is used, this method sends them to the Cloud.
- *onDataExchange* (called from *exchangeData* in parent class *Node*) - this method is called when two devices meet. They exchange data about the completed tasks, they balance their workloads or even completely offload tasks if the nodes are familiar.

As stated previously, our approach towards scheduling is based on a bivalent taxonomy: global and local scheduling. To achieve this, we implemented three crucial components in form of interfaces:

- *Scheduler* - assigns tasks to processors;
- *Balancer* - balances the tasks of two encountered nodes;
- *Publisher* - is a more general global scheduler, used to propagate tasks to neighbour nodes. This is the first step in global scheduling for a device and is meant to offer a broader perspective upon the network.

This blueprint allowed us to add and interchange strategies with minimum impact on the skeleton of the code. We chose to work with sorted sets, more specifically *TreeSet* implementation because it allows us a relatively straightforward addition of different Comparators. Each local scheduling strategy has its own comparator. For the real time case, the comparators were modified - regardless of the order imposed by a certain scheduling algorithms, the first addressed tasks will always be the real-time tasks. Hopefully, this optimization will increase the hit rate for real-time tasks.

## V. RESULTS

### A. Use Case: Drop Computing for Care Centers

Drop Computing works best when the nodes density is rather high. If devices are located in sparse networks, than the possibility of offloading data are extremely low. However, in places such as schools, colleges, shopping malls or healthcare centers, devices are numerous and in the same geographical area. Possibilities are abundant in nowadays world, simply because crowds of people are often met in daily activities. For this paper, we thought of applying this paradigm in healthcare centers. Our motivation regarding this choice consists of two elements: (1) healthcare centers are never empty, therefore we have the certainty that a device will find enough nodes to exchange data and offload tasks, (2) this application also has a noble purpose - there is always a highest sense of reward when pursuing a greater good.

### B. Experimental Results

The foundations of our testing platforms were already set and described in [4] - a big part of the configuration presented in the article remained unchanged, simply because alterations were not demanded. We continued our evaluations on the same initiated path, using the HCMM mobility model. The parameters used to configure the mobility model are: 20 nodes, a trace duration of 9 hours, 200x200 grid, 4 communities and 4 travelers. As previously stated, we used devices that tend to follow the patterns of iPhone 7, with 2GB RAM and 128GB of storage and connections with other nodes are carried out via Bluetooth. Wireless technology is used for reaching other communities. The motivation behind these parameters is as it follows: 9 hours for a healthcare shift, 4 communities for different types of users: patient, doctor, nurse, relatives and 4 travelers, one for each community. As for the device used, we chose iPhone 7 because it was the most popular Apple mobile phone, both in 2019 and in 2020[1].

To test our implementation, we generated 240 combinations between schedulers, balancers and publishers. Tested schedulers [alongside with the configuration order numbers]: FCFS - [0 - 59], Round Robin - [60 - 119], LCFS - [120 - 179] and EDD - [240 - 299]. The tested balancers [11] are: Bidding Balance, Broadcast Balance, Initial Time Balance, and Remaining Time Balance. The tested publishers: Default Publisher, Bidding Publisher and Real-Time Publisher.

Moreover, for each scheduling algorithm, we tested the real-time case too, which prioritises real-time tasks before any other strategy applied. Regarding the task groups limitation on each device, the tests that we run showed that in the interval [2-256] (exponents of two), the number of real-time tasks generated was not so relevant, even zero in certain cases. Because the set of data generated by each version of configuration is tremendously high, we decided to reduce the evaluation to a set of three values for the maximum groups limitation: $tasksGroupsLimit \in \{512, 1024, 2048\}$.

We propose for evaluation a pair of non-preemptive local scheduling algorithms, FCFS (see Figure 1) and LCFS (see Figure 2), and a preemptive one, EDD (see Figure 3), accompanied by different global scheduling algorithms. From a first glance, some configurations are clearly not viable for real-time systems at all. From these charts we can draw multiple conclusions. First of all, it is helpful to visualize these charts as two halves: the first half doesn't get any sort of help from the cloud, whereas the second half sends some of the tasks to the cloud for computation, because their time has expired and the devices couldn't execute them.

Needless to say, the general completion rate is visibly improved when involving the cloud. However, as predicted, the whole "journey" of the task from the device to the cloud and back has a major impact on average turnaround time (TAT) and waiting time (WT). However, promising results lay on the real-time side. The completion rate for real-time tasks in the first half is much better, with a few, understandable exceptions. Moreover, when using combinations of real-time global and local scheduling, the results are favourable.

[1]https://deviceatlas.com/blog/most-popular-iphones

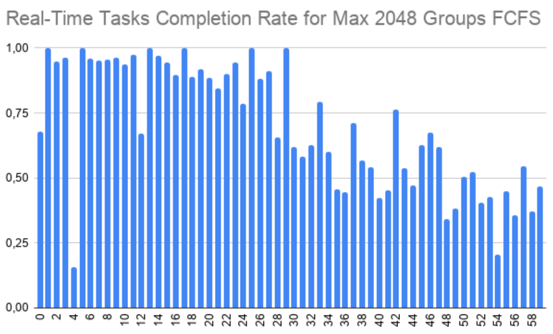

Fig. 1. Real-Time Tasks Completion Rate for Max 2048 Groups FCFS. We generated results for all possible combinations between balancer and publisher, with and without the use of cloud for the FCFS algorithm.

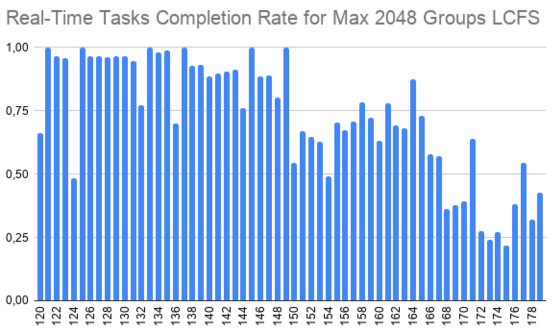

Fig. 2. Real-Time Tasks Completion Rate for Max 2048 Groups LCFS. We generated results for all possible combinations between balancer and publisher, with and without the use of cloud for the LCFS algorithm.

Another algorithm that we tested is the generic version of Round Robin. We varied the quanta allocated for each task with values within a wider range: $quanta \in [50 - 950]\ ms$. The results showed that regardless of the quanta used, when Round Robin is implemented at a local level, the results are almost unchanged - only small variations occur (see Figure 4).

Some successful FCFS configurations are (in terms of real-time completion rate, average TAT and average WT):

- Broadcast Balancer + Real-Time comparator + Real-Time Publisher;
- Initial Time Balancer + Real-Time / Non Real-Time comparator + Real-Time Publisher;
- Remaining Time Balancer + Real-Time comparator + Real-Time / Non Real-Time Publisher.

Some successful LCFS configurations are (in terms of real-time completion rate, average TAT and average WT):

- Bidding Balancer + Real-Time comparator + Bidding Publisher;
- Broadcast Balancer + Real-Time comparator + Real-Time Publisher(100 % completion rate);
- Remaining Time Balancer + Real-Time comparator + Real-Time Publisher(100 % completion rate).

Some successful EDD configurations are (in terms of real-time completion rate, average TAT and average WT):

- Bidding Balancer + Real-Time comparator + Real-Time Publisher;
- Broadcast Balancer + Real-Time comparator + Default Publisher;
- Remaining Time Balancer + Real-Time comparator + Real-Time Publisher.

We understand from these results that our objective can be successfully met when the parameters of the systems are

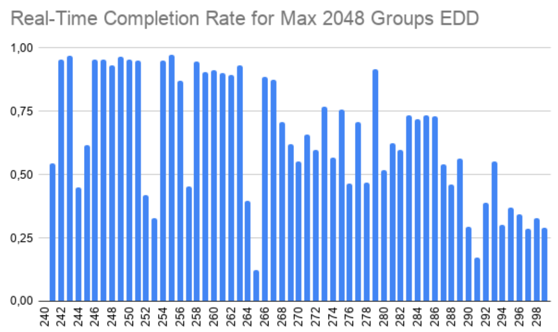

Fig. 3. Real-Time Tasks Completion Rate for Max 2048 Groups EDD. We generated results for all possible combinations between balancer and publisher, with and without the use of cloud for the EDD algorithm.

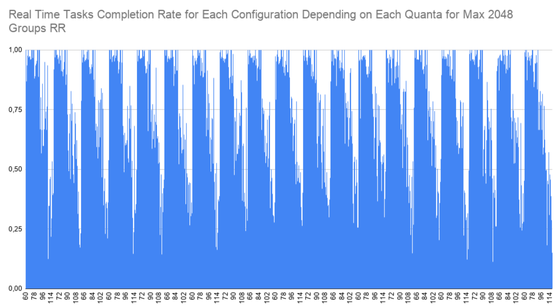

Fig. 4. Real-Time Rasks Completion Rate for Each Configuration Depending on Each Quanta for Max 2048 Groups RR. We generated results for all possible combinations between balancer and publisher, with and without the use of cloud for the RR algorithm. In this case, we changed the quanta value.

configured accordingly. For developing an app that features mainly real-time activities, the optimal strategy is to employ those algorithms that we designed especially for these cases.

## VI. Conclusions

In this paper, we proposed a decentralized, collaborative model, based on Opportunistic Networks and ad-hoc dynamic connections. The proliferation of the IoT dimension requested for a change in the traditional client-server, centralized model for a number of reasons: sending data over the Internet has a negative impact on latency and affects the system's performance, most particularly in time constrained applications, this unprecedented traffic to the Cloud for every single device imposes major privacy risks. Our research was based on a worthy entry-point regarding discussing drop computing paradigm: the aspect of tasks scheduling and balancing. We believe that by optimizing scheduling strategies and using different algorithms for the appropriate corresponding context, the performance of the system will be boosted significantly. Due to the proliferation of real-time based application, we also added real-time tasks in our algorithms.

We intend to develop more algorithms, especially by leveraging the human tie aspect. We started a strategy called Friends First, but there is still work to be done. Moreover, as a future goal, we want to test all the algorithms under even stricter regulations and more demanding computational tasks - stress testing would be necessary, but also time consuming. The mobile application will also need a detailed and rigorous attention, in order to prove the efficiency and usability of Cloud Computing. Real-time testing will also be required. We plan to extend the mobile app and even test it real life.

## Acknowledgment

This work is partially supported by the European Commission, under Grant Agreement no. 833456 (GUARD).

## References

[1] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proc. of the 1st MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

[2] H.J. Cha, H.K. Yang, and YJ. Song. A study on the design of fog computing architecture using sensor networks. *Sensors*, 18(3633):1–16, 2018.

[3] Radu-Ioan Ciobanu, Radu-Corneliu Marin, Ciprian Dobre, and Valentin Cristea. Trust and reputation management for opportunistic dissemination. *Pervasive and Mobile Computing*, 36:44–56, 2017.

[4] Radu-Ioan Ciobanu, Catalin Negru, Florin Pop, Ciprian Dobre, Constandinos X Mavromoustakis, and George Mastorakis. Drop computing: Ad-hoc dynamic collaborative computing, 2019.

[5] Tien Van Do, N.H. Do, H.T. Nguyen, Csaba Rotter, Attila Hegyi, and Peter Hegyi. Comparison of scheduling algorithms for multiple mobile computing edge clouds. *Simulation Modelling Practice and Theory*, 93:104–118, 2019.

[6] X. Evers and R. van Dantzig. A literature study on scheduling in distributed systems. 1992.

[7] Susie J. Wee John G. Apostolopoulos, Wai-tian Tan. Video streaming: Concepts, algorithms and systems. 2002.

[8] Joanna Kołodziej, Florin Pop, and Ciprian Dobre. *Modeling and Simulation in HPC and Cloud Systems*, volume 36. Springer, 2018.

[9] Piotr Nawrocki, Bartłomiej Śnieżyński, Joanna Kołodziej, and Pawel Szynkiewicz. Adaptive context-aware service optimization in mobile cloud computing accounting for security aspects. *Concurrency and Computation: Practice and Experience*, 11 2020.

[10] Ewa Niewiadomska-Szynkiewicz, Andrzej Sikora, Joanna Kołodziej, and Paweł Szynkiewicz. Modelling and simulation of secure energy aware fog sensing systems. *Simulation Modelling Practice and Theory*, 101:102011, 2020.

[11] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8):1110–1123, 1989.

[12] Krithi Ramamritham, John A. Stankovic, and Wei Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8):1110–1123, 1989.

[13] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE IoT J.*, 3(5):637–646, 2016.

[14] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher and Valerie Young. Mobile edge computing a key technology towards 5g. *ETSI White Paper*, (11):2–15, 2015.

[15] Ivan Stojmenovic and Sheng Wen. The fog computing paradigm: Scenarios and security issues. In *2014 federated conference on computer science and information systems*, pages 1–8. IEEE, 2014.

[16] X. Sun and N. Ansari. Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, 54(12):22–29, 2016.

[17] Dimitrios Tychalas and Helen Karatza. A scheduling algorithm for a fog computing system with bag-of-tasks jobs: Simulation and performance evaluation. *Simulation Modelling Practice and Theory*, 98:101982, 2020.

[18] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos. Challenges and opportunities in edge computing. In *Proc. of the IEEE Int. Conf. Smart Cloud*, pages 20–261, November 2016.

[19] Z. Xu, M. Cai, X. Li, T. Hu, and Q. Song. Edge-aided reliable data transmission for heterogeneous edge-iot sensor networks. *Sensors*, 19(2078):1–14, 2019.

[20] S. Yi, Z. Qin, and Q. Li. Security and privacy issues of fog computing: A survey. In *Proc. of 10th Int. Conf. Wireless Algorithms Syst. Appl. (WASA)*, pages 685–695, August 2015.