Online user-driven task scheduling for FemtoClouds

Please, cite this paper as:
Cosimo Anglano, Massimo Canonico, Marco Guazzone
"Online user-driven task scheduling for FemtoClouds,"
Proc. of the 4th International Conference on Fog and Mobile Edge Computing (FMEC), Rome, Italy, Jun 10 - 13, 2019.
DOI:10.1109/FMEC.2019.8795304
Publisher: https://ieeexplore.ieee.org/document/8795304

Online user-driven task scheduling for FemtoClouds

Cosimo Anglano, Massimo Canonico and Marco Guazzone
{cosimo.anglano,massimo.canonico,marco.guazzone}@uniupo.it
Computer Science Institute, DiSIT, University of Piemonte Orientale, Italy

***Abstract*—In Fog Computing, FemtoClouds are emerging computing systems consisting of a set of heterogeneous mobile devices whose users allow to run tasks offloaded by other users. FemtoClouds are well suited to run Bag-of-Tasks (BoTs) applications, but they need effective scheduling algorithms that are able to deal with collections of independently-owned, heterogeneous devices that can suddenly leave the system. In this paper, we present UDFS, an online scheduling algorithm that, by combining knowledge-free task and device selection policies with suitable heterogeneity and volatility tolerance mechanisms, can effectively schedule a stream of BoT applications on FemtoClouds. We evaluate the ability of UDFS to achieve its design goals and to perform better than existing scheduling alternatives, by carrying out a thorough simulation study for a large set of realistic scenarios. Our results indeed show that UDFS can effectively schedule a stream of BoT applications on FemtoClouds, and it can do so more effectively than existing scheduling alternatives.**

***Index Terms*—FemtoCloud, Scheduling policy, Fog computing.**

I. Introduction

With the emerging *Internet of Things* (IoTs) and the proliferation of mobile devices, the number of connected devices will soon reach 50 billion [1]. Data generation is rapidly expanding as well. In 2020, it is expected that such devices will generate 40 trillion gigabytes of data.

The traditional approach to deal with the huge computing and storage capacity demand, needed to process these data, is to resort to *Cloud Computing*. However, these data often require (near) real-time processing (e.g., for augmented reality services or smart traffic light systems) [2]–[4]. Therefore, the inherent high latency of the core network makes Cloud Computing unsuitable to meet these stringent requirements. *Fog Computing* [5] has recently emerged as a new paradigm to mitigate the escalation in resource congestion and to support low-latency services. By migrating data computation or storage to the network "edge" (near the end users), Fog Computing can offload the computational requests from the Cloud Computing infrastructure, and can significantly reduce the latency with respect to centralized Clouds.

Recently, it has been argued that the processing capability in Fog Computing can be provided by underutilized mobile edge devices (e.g., smartphones and tablets), that are aggregated into a so-called *FemtoCloud*, whose users agree to run tasks offloaded by other users (this can be achieved by resorting to suitable incentive mechanisms) [6]. FemtoClouds are considered to be very promising since the computing capacity provided by mobile devices is becoming more and more ubiquitous, making mobile devices the most used platform in recent years [7], [8].

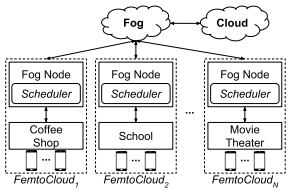

Fig. 1. FemtoCloud system architecture.

Figure 1 shows the architecture of a typical FemtoCloud system, consisting of a set of distinct FemtoClouds where each FemtoCloud is coordinated by a *Fog Node*, an always-present/always-on machine which is in charge of receiving offloaded tasks from a population of users, of dispatching them on the devices it coordinates, of receiving the results they generate, and of forwarding these results to the corresponding users. Specifically, the Fog Node runs a *Scheduler* component, whose main goal is to schedule offloaded tasks on available devices so as to minimize their completion time. In the following, without loss of generality, we use the terms "scheduler" and "Fog Node" interchangeably.

To successfully carry out scheduling, the Fog Node has to properly take into account two crucial issues, namely device *heterogeneity* (edge devices are very different both in terms of hardware and software characteristics) and *availability* (an edge device could become unreachable at any moment due either to network connection issues, to device owner mobility, or to battery depletion).

These problems are not new, and have already been addressed in the past by resorting to combinations of *replication* (i.e., several instances of the same task are created and scheduled on different devices) and *periodic checkpointing* (i.e., the state of a task is periodically saved, so that its execution can be restarted from the last saved state), first in the context of clusters of non-dedicated PCs [9] and, later, of desktop grids [10]. Hence, a natural question arises about the applicability of these solutions to FemtoCloud systems.

The analysis of the above solutions reveals however that their efficacy was somehow limited by the lack of reliable information about resource availability. In fact, while replication does not require any information concerning the characteristics of tasks and of devices, checkpointing requires the choice of a suitable frequency with which state is saved. It is known [11] that the optimal checkpoint frequency can be computed only under specific assumptions about the behavior of machines and applications so, in real cases, it can only be approximated,

and this approximation usually results in the creation of many more checkpoints than strictly needed.

In case of traditional computing systems, an excessive checkpointing activity would of course results in suboptimal performance, but would have no other consequences. In FemtoCloud systems, however, it would severely hurt the stability of the system, given that checkpointing is a resource-intensive activity not only in terms of storage space, but also in terms of computing capacity. On edge devices the creation of a checkpoint would indeed not only slow down the device much more than a traditional machine, but also deplete its battery more quickly, hence either forcing the device owner to leave the system or causing its switch off. Therefore, existing scheduling algorithms based on replication and periodic checkpointing would exhibit potentially unsatisfactory performance also on FemtoCloud systems.

However, we observe that – unlike clusters of PCs and desktop grids – in FemtoClouds the computational capacity is narrowed in restricted area such as a public transit, a classroom, a movie theater, a coffee shop and so on, as illustrated in Figure 1. Hence, we can exploit this fact to collect more precise information about when the device will leave the system, and carry out only one checkpoint about when the device is going to leave. In this way, the number of checkpoints is minimized (only 1 checkpoint is created for each device), and only useful checkpoints are created (i.e., only for those devices that will leave while the task allocated them is still running).

This works very well in situations where users have to stay in the FemtoCloud for an amount of time that can be determined in advance from the context (e.g., the duration of a movie or of a lecture). In other scenarios where presence times cannot be determined by the context (e.g., in a coffee shop), we can ask the device owners to declare their expected presence time in advance, as soon as they join the FemtoCloud (for instance, a reward mechanisms can be easily conceived to incentivize users to do so in a precise manner), or just before leaving the system (e.g., by pressing a specific button of the FemtoCloud client application installed on the user device).

In this paper we argue that, by exploiting the information about device presence gathered as discussed before, it is possible to build a scheduling algorithm for FemtoCloud systems that – by combining replication and checkpointing – can provide significant performance benefits with respect to state-of-the-art alternative solutions that do not use such a combination. We prove our claim by presenting a novel online scheduling algorithm for FemtoCloud systems based on the above principle, that we call *User-Driven FemtoCloud Scheduler* (UDFS), and we experimentally compare its performance against alternative solutions by carrying out an extensive simulation study for a variety of real-world operational conditions. The results we obtain clearly show that UDFS outperforms alternative solutions in all the scenarios we considered.

The rest of the paper is organized as follows: Section II summarizes related works, while Section III presents the details of our algorithm. Section IV shows the performance evaluation used in our experiments, and in Section V we discuss the results obtained. Finally, Section VI presents a summary of our findings and a discussion of future directions.

II. Related Works

The literature focusing on computation offloading for mobile devices offers several contributions that can be applied also to FemtoCloud systems [12]–[16].

In [12], authors present an offloading scheme that is based on node mobility model in mobile social network and can support heterogeneous tasks. In [13], by assuming the knowledge of transmission time, processing time and queueing time of tasks, an integer linear programming problem and an offline centralized algorithm to solve it are proposed. In [14], authors present the Optimal Fair Multi-criteria Resource Allocation (OFMRA) algorithm to select offloading nodes and to allocate resources so as to minimize task completion time and, simultaneously, to maximize device lifetime. This work assumes homogeneous tasks in terms of computational requirements, which in practice is not always true. In [15], authors design and implement a context-aware Offloading Middleware for Mobile Cloud (OMMC), which aims to simultaneously minimize the task completion time, minimize the overall energy consumption of mobile devices, and meet the task deadlines.

Finally, in [16], authors propose a scheduling algorithm for FemtoClouds, which is able to estimate the availability and the computing capacity of devices, and to exploit this information to prioritize the tasks with higher computational requirements, as well as to maximize the useful computation and increase processor utilization by selecting mobile devices able to compute tasks earlier. A static checkpointing mechanism, which runs periodically every 15 seconds regardless the device failure rate, is also used.

The above scheduling algorithms need the knowledge of information about submitted tasks (i.e., their arrival rate, the amount of work they require, or both) and device characteristics (i.e., their computation capacity). These information, however, may be very difficult to collect reliably in a FemtoCloud system, because of the lack of control on the user population providing devices, and on the possible difficulty in convincing them to install suitable software on their devices. Also, in many cases these algorithms work offline (i.e., they compute a scheduling plan ahead of time before tasks are actually submitted), which, however, implies that precise information about tasks and devices are known in advance.

In contrast, our UDFS algorithm works online (i.e., tasks are scheduled as soon as they arrive to the system), and does not require any information about the characteristics of tasks and devices. The only information that it requires is, as already mentioned, the presence time of each device in the system, which in many cases can be easily inferred from the context or can be requested to the user.

III. Scheduling algorithm

A scheduling algorithm has to select a task and submit it to a selected device where will be executed. This decision has to be

made repeatedly until all tasks have been assigned to devices or there is no idle device. The way this double selection is made characterizes the scheduling algorithm. In this section, we first describe the workload model (Section III-A) and the system model (Section III-B), and finally we present the UDFS algorithm (Section III-C).

A. Workload model

The inherent wide distribution, heterogeneity, and dynamism of FemtoCloud systems makes them better suited to the execution of loosely-coupled parallel applications rather than tightly-coupled ones. *Bag-of-Tasks applications* (BoTs) [17] (parallel applications whose tasks are completely independent from one another) can be used in a variety of IoT data analytics domains, including smart video surveillance in transportation systems [18] (to process videos in real-time coming from many different cameras so as to automatically identify potential security threats), and city-wide traffic forecasting [19] (to train learning models from data collected by smart traffic IoT devices so as to learn traffic patterns). Each BoT is characterized by two parameters, namely the number of tasks and the computation requirements of each task.

B. System model

We consider a set of heterogeneous mobile devices connected to a Fog Node. Each device d is characterized by its *nominal computing power* $P(d)$, a real number whose value is directly proportional to its speed (i.e., a device d_1 with $P(d_1) = 2$ is twice faster than a device d_2 with $P(d_2) = 1$), and by the *communication bandwidth* $B(d)$ (in bits/sec.) towards the Fog Node.

Each device may join and leave the FemtoCloud at any time, but we assume that the corresponding user declares how long (s)he will stay in that location when (s)he joins. We believe this assumption is realistic for the following two reasons: (1) the user should be able to predict how long (s)he will stay in the location (i.e., in a movie theater, until the end of the movie; in a school, the duration of the lessons; in a theater, the duration of the show; in a bus, the duration of the journey; and so on), and (2) we postulate the user receives incentives if (s)he communicate its *presence time* (i.e., the time elapsed between the arrival and the departure of the device from the system) to the local Fog Node. Finally, we assume that FemtoCloud applications run encapsulated in virtual machines on a mobile virtualization platform [20], [21], so that their execution state can be easily saved and restored later on any device.

C. The UDFS scheduling policy

UDFS combines very simple task and device selection policies with several mechanisms specifically tailored to cope with device departures from the FemtoCloud. In particular, it is derived from the WQR-FT scheduling algorithm [10], that targets desktop grids, by replacing its checkpointing mechanism (based on the Young's formula [22]) as follows.

UDFS selects in an arbitrary order the tasks of the oldest BoT (i.e., the BoT who arrived first in the system with respect to the others) and submits them to devices as soon as they become available. No information concerning task or device characteristics is taken into consideration for these selections. In order to tolerate device departures, UDFS exploits the following three mechanisms:

1) *task replication*: replication is used to cope with device heterogeneity and departures. As a matter of fact, by submitting the same task to different devices, we increase the chances of task completion, since to complete the task it suffices that at least one of these devices stays in the system. At the same time, we increase the chance of reducing its execution time, since in general both faster and slower devices will be chosen for its replicas, and the faster device will complete the tasks (at that point, the slower replicas that are still running will be aborted, as they become useless). The maximum number of replicas for each task is a scheduler parameter called *replication threshold* and denoted by τ_r;
2) *task selection*: anytime a device becomes available (because it has completed the task assigned or it has just joined the FemtoCloud system) UDFS selects the oldest task with the lowest number of replicas and submits it to the idle device. If all tasks have already a number of replicas that is equal to the replication threshold, UDFS waits for a task to submit (that is, a new task from a new BoT just arrived in the system or a task died due to a device failure). A task remains in the system until it has been completed;
3) *checkpointing*: before a device leaves the FemtoCloud system as claimed by its owner, a snapshot of the virtual machine running its task t is saved by the Fog Node, so that when the task has been selected by the scheduler, this snapshot will be used to restart task t.

The UDFS algorithm (shown in Algorithm 1) maintains a *task set* T (which stores the tasks to complete) and a *device set* D (which stores the devices currently in the Femtocloud), and is invoked upon five possible events, namely *NewTask* (fired when a new task is to be executed), *TaskDone* (fired when the execution of a task is done), *DeviceIdle* (fired when a device becomes idle or when a new device joins the FemtoCloud), *DeviceGone* (fired when a device leaves the FemtoCloud), and *NewCheckpoint* (fired when a device is going to leave the FemtoCloud so as to create a new checkpoint for the task running on it).

As shown in Algorithm 1, whatever event is fired, UDFS first performs event-specific actions (lines 15–34) and then invokes the procedure *Schedule* (line 35), which performs the following actions. Firstly, it selects a task t from T according to the above task selection policy and with a number of running replicas below the threshold τ_r (line 2). Then, it picks an idle device d from D (line 3). Next, it runs a new t's replica on d, either restoring its execution from an existing checkpoint (line 6) or, if not available, starting it from scratch (line 8). Finally, it increments the number of t's running replicas (line 10) and schedules the creation of a

new checkpoint for t from the replica on d before d's presence time elapses (line 11).

For the event-specific actions, when a *NewTask* is fired, UDFS adds the new task to T, so that it can be scheduled for execution later by the *Schedule* procedure (line 16). [1] Conversely, when a *TaskDone* event is fired, UDFS aborts all running replicas of the just terminated task (so that devices where these replicas were run become idle), deletes its checkpoint, cancels any pending *NewCheckpoint* event (if any), and removes it from T (lines 18–21). Instead, when a *DeviceIdle* event is fired, UDFS adds the idle device to D, so that it can be chosen by the *Schedule* procedure for executing a task (line 23). When a *DeviceGone* event is fired, UDFS removes the departed devices from D, cancels the pending *NewCheckpoint* event related to the replica running on it (if any) and decrements the number of running replicas of the task running on it, if any (lines 25–31). Finally, when a *NewCheckpoint* event is fired, UDFS creates a new checkpoint for a task running on a device that is leaving the FemtoCloud system (line 33).

Algorithm 1: The UDFS scheduling algorithm.

```
1  procedure Schedule(T,D,τ_r)
       Input: task set T, device set D, replication threshold τ_r.
2      t ← GetOldestTaskWithLowestNumReplicas(T,τ_r)
3      d ← GetIdleDevice(D)
4      if t ≠ nil and d ≠ nil then
5          if CheckpointExist(t) then
6              RunTaskReplicaFromCheckpoint(t, d)
7          else
8              RunTaskReplica(t, d)
9          end
10         IncrNumTaskReplicas(t)
11         ScheduleNewCheckpointEvent(t, d)
12     end
13 end
14 procedure Main(T,D,e,τ_r)
       Input: task set T, device set D, event e, replication threshold τ_r.
15     if EventType(e) = NewTask then
16         InsertTask(T, GetTask(e))
17     else if EventType(e) = TaskDone then
18         t ← GetTask(e)
19         RemoveCheckpoint(t)
20         CancelNewCheckpointEvents(t,
             GetDevicesForTask(t))
21         RemoveTaskReplicas(t, T)
22     else if EventType(e) = DeviceIdle then
23         InsertDevice(D, GetDevice(e))
24     else if EventType(e) = DeviceGone then
25         d ← GetDevice(e)
26         RemoveDevice(D, d)
27         t ← GetTask(e)
28         if t ≠ nil then
29             CancelNewCheckpointEvent(t, d)
30             DecrNumTaskReplicas(t)
31         end
32     else if EventType(e) = NewCheckpoint then
33         CreateCheckpoint(GetTask(e), GetDevice(e))
34     end
35     Schedule(T,D,τ_r)
36 end
```

[1] In T there can be other tasks older than the new one that are waiting for an idle device where to be executed.

IV. Performance evaluation

In order to assess the ability of UDFS to successfully schedule BoT applications on FemtoClouds, we perform an exhaustive simulation study in which we compare the performance it attains against those attained by the WQR-FT [10], the Habak's [16], and the First Come, First Served (FCFS) scheduling algorithms.

Habak's algorithm has been included in the comparison since, at the best of our knowledge, it is the only existing scheduling algorithm for FemtoClouds. WQR-FT has been instead included since it is the progenitor of UDFS, so its inclusion allows us to assess the efficacy of user-driven checkpointing with respect to periodic checkpointing. Finally, FCFS is a scheduling algorithm that uses no information and no device departure tolerance mechanisms, so its inclusion allows us to assess the impact of the mechanisms used by UDFS.

To carry out our study, we developed a discrete-event simulator, written in Python and C++, and based on the *Salabim library* [23].

In order to obtain realistic results, in our simulations we considered a set of realistic scenarios and workloads, obtained from the experimental setup used in [16]. Also, in order to assess the sensitivity of UDFS to errors in the presence times communicated by device owners, we consider scenarios in which the presence times are affected by errors of different magnitudes and sign.

Our study has been carried out by using as metric the *Average BoT Completion Time* (*ABCT*), that is the time elapsed between the submission of a BoT and the termination of all its tasks. This average value has been computed by using the independent replication method [24], where each independent replica corresponds to the time to complete 2000 BoTs and where the whole simulation stops when the relative precision of the 95% confidence interval is $\leq 4\%$. [2]

A. Simulated scenarios

For our study, we consider 9 distinct configurations of FemtoCloud systems (described in Section IV-A1), that have to process a workload consisting in a stream of BoTs (described in Section IV-A2) (the rationale behind the choice of the parameters characterizing the FemtoCloud system and the workload is discussed in Section IV-A3). Each experiment is repeated for each one of the scheduling algorithms included in our comparison (whose parameters are discussed in Section IV-A4).

1) FemtoCloud configurations: The FemtoCloud system considered in our study is composed by a cluster of heterogeneous mobile devices whose types are "Galaxy S5", "Nexus 7 (2012)", "Nexus 7 (2013)" and "Nexus 10 (2013)", and whose associated nominal computing powers are 3.3, 7.1, 8.5, 10.7, respectively. The nominal computing powers have

[2] The choice of 95% confidence level is most common because it provides a good balance between *precision* (as reflected in the width of the confidence interval) and *reliability* (as expressed by the confidence level) [25].

TABLE I
EXPERIMENTAL TASK CHARACTERISTICS.

Task type	Capacity (MFLOPS)	Output (MBytes)
Lightweight tasks	10	0.2
Medium tasks	30	2.0
Compute intensive tasks	100	0.5
Data generating tasks	20	20.0

been obtained by a measurement study with real devices. We consider that the types of devices are uniformly distributed between the resources in our FemtoCloud system.

For the *device arrival rate*, we consider 3 different scenarios, called *FewDev*, *MedDev* and *ManyDev*, corresponding to FemtoClouds with a small, moderate and large number of mobile devices, respectively, and we characterize them according to 3 different Exponential probability distributions whose rate parameters are set to 7.5, 15 and 22.5, respectively. Finally, for the *device presence time*, we also consider 3 different scenarios, namely *LowAvail*, *MedAvail* and *HighAvail*, corresponding to the case of low, medium and high presence time, respectively, and we characterize them according to 3 different Normal probability distributions whose mean parameters are set to 10.38, 13.84, 17.30 and whose standard deviation parameters are set to 2.08, 2.77, 3.46, respectively.

In all the scenarios, the communication bandwidth between each device and the Fog Node is drawn from a Normal probability distribution with mean set to 30 Mbps and standard deviation set to 6 Mbps.

By combining the 3 presence time scenarios with the 3 device arrival scenarios, we consider 9 different scenarios, that we denote as "*X-Y*", where $X \in \{\text{LowAvail}, \text{MedAvail}, \text{HighAvail}\}$ and $Y \in \{\text{FewDev}, \text{MedDev}, \text{ManyDev}\}$.

2) BoT workloads: For our study, we consider a BoT workload where each BoT has 30 tasks, and they arrive at the system with a rate that is exponentially distributed with rate 0.5 (these parameters have been taken from [16]). We consider four task types, that differ among them in both the amount of computing capacity they require (expressed in MFLOPS) and the amount of data that they generate (expressed in MBytes), as reported in Table I. These types cover different resource usage scenarios, spanning from tasks with a very low resource demand (i.e., "Lightweight tasks") to tasks with a high computation or network bandwidth demand (i.e., "Compute intensive tasks" and "Data generating tasks", respectively). Each BoT may be composed by tasks of different types, and the composition of each BoT is determined by randomly sampling from these types according to the uniform distribution.

The *completion time* of a task is computed as the sum of its *execution time* (given by the ratio between its computing requirement and the nominal computing power of the device where it has been executed) and its *stage-out time* (given by the ratio between the task output size and the communication bandwidth between the device where the task has run and the Fog Node).

3) Rationale of the FemtoCloud parameters settings: While, as already mentioned, most of the values of the simulation parameters are taken from [16], the device presence time and the device arrival rate have been suitably set in order to study the performance of the scheduling policies in different scenarios, as follows.

For the device presence time, we consider 3 different scenarios, namely LowAvail, MedAvail, and HighAvail, where the device presence time is on average lower than, very close to, or higher than the task completion time. Thus, we consider the *Average Task Completion Time* (*ATCT*) (i.e., the mean time needed to complete a task), which depends on the *Average Task Execution Time* (*ATET*) and the *Average Stage Out Time* (*ASOT*) as follows: $ATCT = ATET + ASOT$. The *ATET* value is computed as the ratio between the *Average Nominal Task Execution Time* (*ANTET*) (i.e., the average of the computing capacity values of Table I) and the *Average Nominal Computing Power* (*ANCP*) (i.e., the average of the device nominal computing power values of Section IV-A1) as follows: $ATET = ANTET/ANCP$. The *ASOT* value is computed as the ratio between the *Average Nominal Task Output Size* (*ANTOS*) (i.e., the average of the task output size values of Table I) and the *Average Network Bandwidth* (*ANB*) value (which is equal to 30 Mbps, see Section IV-A1), as follows: $ASOT = ANTOS/ANB$.

By substituting the equations above with the actual values provided in Sections IV-A1 and IV-A2, we obtain that *ATCT* is equal to 6.92 sec. However, this value does not consider the time spent to create and store the checkpoints during the task execution. For this reason we decide to consider 3 different scenarios where the average device presence time is set to a multiple of *ATCT*. In particular, for the LowAvail scenario we set it to $1.5 \cdot ATCT = 10.38$ sec., for the MedAvail scenario we set it to $2 \cdot ATCT = 13.84$ sec. and, finally, for the HighAvail scenario we set it to $2.5 \cdot ATCT = 17.30$ sec.

For the device arrival rate, we consider 3 different scenarios, namely FewDev, MedDev, and ManyDev, where the number of devices is on average lower than, very close to, and higher than the average number of tasks in the system, and, for each one of them, we set the device arrival rate *DAR* as follows: $DAR = BAR \cdot BNT \cdot \beta$, where *BAR* is the BoT arrival rate (which is equal to 0.5, see Section IV-A2), *BNT* is the number of tasks for each BoT (which is equal to 30, see Section IV-A2), and β is the percentage of the number of devices in the system with respect to the number of tasks, that we set to 50%, 100% and 150% for the FewDev, MedDev and ManyDev scenarios, respectively.

4) Scheduler parameters: For FCFS, no parameters are needed to be set, as this scheduling algorithm does not use any of the mechanisms employed by the other scheduling algorithms we consider.

For the other 3 algorithms (Habak, WQR-FT, UDFS), instead, it is needed to specify the *checkpoint frequency* and the *checkpoint time* (i.e., the time required to create a checkpoint).

TABLE II
PRESENCE TIME MODEL ERROR.

Error Model	Distribution
No	No error
Neutral (Neu)	Normal(0, 0.1)
LowNegative (LNeg)	Normal(-0.05, 0.1)
Negative (Neg)	Normal(-0.025, 0.1)
LowPositive (LPos)	Normal(0.025, 0.1)
Positive (Pos)	Normal(0.05, 0.1)

For WQR-FT and UDFS, it is also needed to specify the *replication threshold*.

The *checkpoint time* is set to 1 sec. for each algorithm. Instead, for the *checkpoint frequency*, we make the following choices:

- Habak: as in [16], a checkpoint is created periodically every 15 sec.;
- WQR-FT: checkpoints are created periodically every *ChkInt* seconds, computed by means of the Young's formula [22] as: $ChkInt = \sqrt{2 \cdot C \cdot EPT}$, where the *EPT* is the estimated presence time of the device, and C is the checkpoint time. In our simulator, *EPT* is computed on-line by the Fog Node by means of an *exponential moving average* [26] any time a device leaves the system as follows: $EPT = \alpha \cdot PT + (1 - \alpha) \cdot EPT$, where *PT* is the presence time of the device, and α is the *smoothing parameter* that we set to 0.5 so that the contribution of the last observed presence time on the estimated presence time is 50%.
- UDFS: since the device presence time is stated by the device owner when it joins the FemtoCloud, only one single checkpoint is created when the device is about to leave the system. However, this information may not be always accurate as the device owner may decide to leave the system earlier or later than what claimed. To study the sensitivity of UDFS to the occurrence of these errors, we run experiments for different error models, each of which is characterized by a Normal probability distribution whose parameters are reported in Table II. Specifically, in our experiments, UDFS performs a checkpoint according to the presence time stated by the users (*SPT*), but we simulate the *real presence time* (*RPT*) of devices by considering the error model as follows: $RPT = SPT \cdot (1 + error)$, where *error* is a random number drawn from the Normal distribution depending on one of the error models of Table II. The considered error models include the cases where the device owner decide to leave the system both before and after the claimed time. For instance, the *Pos* error model represents the scenarios where, on average, the device owner leaves the system later than what (s)he claimed (i.e., on average the device presence time is 5% greater than what claimed by the device owner); instead, the *LNeg* error model represents the cases where, on average, the device owner leaves the system earlier than what claimed (i.e., on average the device presence time is 5% smaller that what claimed by the device owner).

Finally, for WQR-FT and UDFS, we performed experiments for various values of the *replication threshold*, but we observed that using more than 2 replicas does not result in significant performance improvements. Therefore, we set the replication threshold to 2 for all the experimental scenarios we consider.

V. RESULTS

In this section, we present the results obtained in our study as bar charts with error bars, where each bar denotes the *Average BoT Completion Time* (*ABCT*) achieved by a specific scheduling policy and the associated error bar represents its 95% confidence interval. For the sake of readability, we do not report the graphs of all 9 scenarios but only the most significant ones.

Figure 2 reports the results for the LowAvail-FewDev scenario. As can be seen by these results, only WQR-FT and UDFS are able to complete all the submitted BoTs, while the *ABCT* for FCFS and Habak grows beyond any reasonable limit (in the figure, we denote this with a bar approaching to infinity), meaning that, under these policies, the system is operating under extremely high resource saturation levels. This is due to the fact that when the device presence time is low, it is necessary to consider dynamic fault tolerant mechanisms in order to overcome the failures. In fact, FCFS does not implement any fault tolerant mechanisms and Habak uses a static checkpoint mechanisms (a checkpoint is saved every 15 sec.).

By comparing the performance of WQR-FT with respect to UDFS, we note that UDFS outperforms WQR-FT for all error models considered except for the LNeg and Neg cases. For the LNeg case, since UDFS cannot save a checkpoint for a task running on a device that leaves the system sooner than claimed, a premature device departure will result in restarting the task from scratch if no other checkpoint was saved before for it. For the Neg case, we note that the performance of WQR-FT and UDFS are comparable. As expected, the best performance of UDFS is when there is no error which means that our approach is able to save the checkpoint of the state of the execution just before the device leaves the system (this is the optimal scenario for UDFS). For all the other error models, UDFS is always able to achieve better performance with respect to WQR-FT even if a positive error in our approach could result in a too early checkpoint creation.

Figure 3 reports the results for the LowAvail-ManyDev scenario. Firstly, we note that all scheduling policies are able to complete all the submitted BoTs thanks to the high number of devices in the system that compensate poor scheduling decisions. In particular, we note the significant performance difference between FCFS and Habak scheduling policies with respect to WQR-FT and UDFS policies; for example, in the scenario without error, the *ABCT* for FCFS and Habak is around 1700 sec., while the one for WQR-FT and UDFS policies is less than 30 sec.

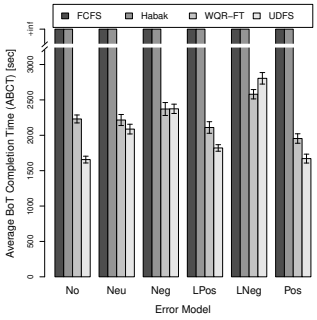

Fig. 2. Results for LowAvail-FewDev scenario.

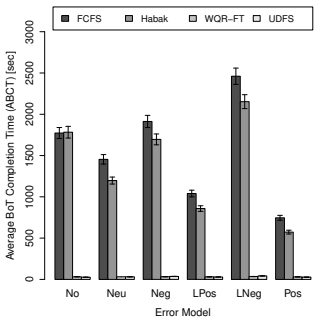

Fig. 3. Results for LowAvail-ManyDev scenario.

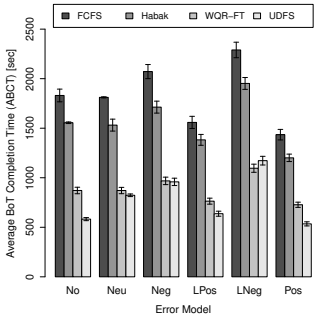

Fig. 4. Results for MedAvail-FewDev scenario.

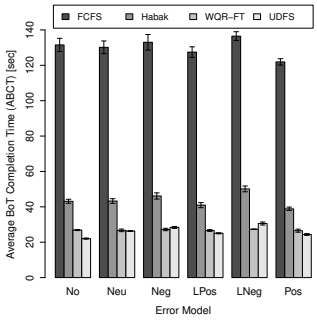

Fig. 5. Results for MedAvail-ManyDev scenario.

As expected, in most of the error models considered the worst scheduling policy is FCFS while the difference between WQR-FT and UDFS is negligible. The latter is due to the fact that with plenty of available devices, the replication mechanisms can be exploited to overcome the failure and this make the checkpoint mechanism less useful. As a matter of fact, the higher the number of idle resources, the higher the probability to submit a task to a fast device able to complete the task before it leaves the system. In general, the scheduling policies are able to achieve their best performance when the error model is positive. This can be explained by considering that a positive error means that the device will stay in the system longer than claimed so they could be able to complete the task or checkpointing-based policies can save more checkpoints.

Figure 4 reports the results for the MedAvail-FewDev scenario. If we compare these results with those obtained for the LowAvail-FewDev scenario (see Figure 2) we note that having a higher device presence time makes FCFS and Habak able to complete all the submitted BoTs and, in general, all scheduling policies achieve better performance with respect to the LowAvail-FewDev scenario. For example, UDFS in LowAvail-FewDev scenario was able to complete the BoTs in between 1500 and 3000 sec., while in MedAvail-FewDev scenario it is able to complete the BoT between 500 and 1250 sec. In general, FCFS is again the worst scheduling policy and WQR-FT and UDFS are the best ones, where WQR-FT is the best policy when the error model is negative, while UDFS outperforms WQR-FT when the error model is positive for the same reasons mentioned before.

By increasing the number of devices in the system, the performance gap between the various scheduling policies decreases, as can be observed in Figure 5, where results for the MedAvail- ManyDev scenario are reported. In particular, we note that WQR-FT and UDFS have comparable performances and this confirms that when there are many devices the replication mechanism is more important than the checkpoint mechanism.

Finally, Figure 6 shows the results for the HighAvail-ManyDev scenario, which represents the best possible working condition for a scheduling policy because there are many devices with a high presence time. From the figure, we note that the performance of FCFS is still the worst but, conversely to the previous scenarios, we note that Habak is able to achieve performance close to WQR-FT and UDFS policies. This is due to the fact that in a scenario with plenty of devices with high availability, the chance that submitted tasks get completed in the first replica is much more higher than the other scenarios, thus making the use of dynamic fault-tolerant mechanisms less effective.

VI. Conclusions

In this paper, we have considered the problem of scheduling a stream of BoT applications on a FemtoCloud system, composed of an ensemble of heterogeneous mobile devices that join and leave the system anytime without notice. We addressed this problem by proposing an online scheduling

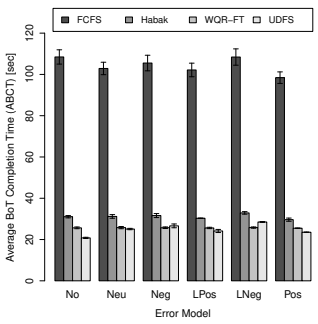

Fig. 6. Results for HighAvail-ManyDev scenario.

algorithm, named UDFS, able to effectively schedule a stream of BoT applications on a FemtoCloud system, thanks to the combination of simple task and machine selection policies (that do not require any information concerning the applications and the devices) with mechanisms specifically conceived to tolerate device heterogeneity and volatility.

We assessed the ability of UDFS to meet its design goals, and to perform better than existing alternatives, by performing an extensive simulation study for a large set of realistic operational scenarios. Our results clearly indicate that UDFS is able to effectively schedule a stream of BoT applications on FemtoCloud systems, and to do so more effectively than existing scheduling alternatives.

As future work, we plan to study the effects of different error models as well as of other scheduling mechanisms, such as the use of a dynamic replication threshold and of task-dependent replication thresholds, on the performance of UDFS. We are also interested in implementing a fuzzy controller inside the Fog Node as proposed in [27]–[29]. Furthermore, we consider to integrate in UDFS a prediction algorithm (e.g., [30]) to estimate device presence time. Finally, we plan to implement a prototype of UDFS using existing platforms like *OpenStack++* [31], *Edgent* [32], *CirrusCloud* [16], *Cloud-TUI* [33], or the *Prometheus* toolkit [34].

ACKNOWLEDGMENTS

This research is original and has a financial support of the Università del Piemonte Orientale.

REFERENCES

[1] N. Zhang et al., "Synergy of big data and 5g wireless networks: opportunities, approaches, and challenges," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 12–18, 2018.

[2] F. Bonomi et al., "Fog computing and its role in the internet of things," in *Proc. of the MCC'12*, 2012, pp. 13–16.

[3] C. Anglano, M. Canonico, P. Castagno, M. Guazzone, and M. Sereno, "A game-theoretic approach to coalition formation in fog provider federations," in *2018 Third International Conference on Fog and Mobile Edge Computing*, ser. FMEC'18, 2018, pp. 123–130.

[4] C. Anglano, M. Canonico, and M. Guazzone, "Profit-aware Resource Management for Edge Computing Systems," in *Proc. of the 1st International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys'18, 2018, pp. 25–30.

[5] W. Yu et al., "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017.

[6] K. Habak et al., "Femtoclouds: Leveraging mobile devices to provide cloud service at the edge," in *Proc. of the CLOUD'15*, 2015, pp. 9–16.

[7] C. Anglano, M. Guazzone, and M. Sereno, "Maximizing profit in green cellular networks through collaborative games," *Computer Networks*, vol. 75, Part A, pp. 260–275, 2014.

[8] D. Chaffey. (2018) Mobile marketing statistics compilation. [Online]. Available: https://bit.ly/1wXFQtJ

[9] C. Anglano and M. Botta, "Now g-net: learning classification programs on networks of workstations," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 463–480, Oct 2002.

[10] C. Anglano and M. Canonico, "Fault-tolerant scheduling for bag-of-tasks grid applications," in *European Grid Conference*. Springer, 2005, pp. 630–639.

[11] M. Bougeret et al., "Checkpointing strategies for parallel jobs," in *Proc. of SC'11*, 2011, pp. 33:1–33:11.

[12] M. Xiao et al., "Multi-task assignment for crowdsensing in mobile social networks," in *Proc. of the INFOCOM'15*, 2015, pp. 2227–2235.

[13] Z. Lu et al., "Task allocation for mobile cloud computing in heterogeneous wireless networks," in *Proc. of the ICCCN'15*, 2015, pp. 1–9.

[14] S. Ghasemi-Falavarjani et al., "A multi-criteria resource allocation mechanism for mobile clouds," in *Proc. of the CNDS'13*, 2013, pp. 145–154.

[15] ——, "Context-aware multi-objective resource allocation in mobile cloud," *Comput. Electr. Eng.*, vol. 44, pp. 218–240, 2015.

[16] K. Habak et al., "Workload management for dynamic mobile device clusters in edge femtoclouds," in *Proc. of the SEC'17*, 2017, p. 6.

[17] K. H. Kim et al., "Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters." in *Proc. of the CCGrid'07*, vol. 7, 2007, pp. 541–548.

[18] A. J. V. Neto et al., "Fog-based crime-assistance in smart iot transportation system," *IEEE Access*, vol. 6, pp. 11 101–11 111, 2018.

[19] J. L. Pérez et al., "A resilient and distributed near real-time traffic forecasting application for fog computing environments," *Future Generat. Comput. Syst.*, vol. 87, pp. 198–212, 2018.

[20] D. Jaramillo et al., Ed., *Virtualization Techniques for Mobile Systems*. Springer, 2014.

[21] J. Shuja et al., "A survey of mobile device virtualization: Taxonomy and state of the art," *ACM Comput. Surv.*, vol. 49, no. 1, pp. 1:1–1:36, Apr. 2016.

[22] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Commun. ACM*, vol. 17, no. 9, pp. 530–531, Sep. 1974.

[23] R. Van der Ham, "Salabim: discrete event simulation and animation in python," *J. Open Source Softw.*, vol. 3, no. 27, 2018.

[24] J. Banks et al., *Discrete-Event System Simulation*, 5th ed. Prentice Hall, 2010.

[25] M. F. Triola, *Elementary Statistics*, 11st ed. Addison-Wesley, 2010.

[26] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *Int. J. Forecast.*, vol. 20, no. 1, pp. 5–10, 2004.

[27] L. Albano, C. Anglano, M. Canonico, and M. Guazzone, "Fuzzy-Q&E: Achieving QoS guarantees and energy savings for cloud applications with fuzzy control," in *2013 International Conference on Cloud and Green Computing*, ser. CGC'13, 2013, pp. 159–166.

[28] C. Anglano, M. Canonico, and M. Guazzone, "FC2Q: exploiting fuzzy control in server consolidation for cloud applications with SLA constraints," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 4491–4514, 2015.

[29] ——, "FCMS: A fuzzy controller for CPU and memory consolidation under SLA constraints," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 5, 2017.

[30] S. Montani and C. Anglano, "Case-based reasoning for autonomous service failure diagnosis and remediation in software systems," in *Advances in Case-Based Reasoning*, T. R. Roth-Berghofer, M. H. Göker, and H. A. Güvenir, Eds. Springer-Verlag, 2006, pp. 489–503.

[31] K. Ha et al., "Openstack++ for cloudlet deployment," Carnegie Mellon University, Technical Report CMU-CS-15-123, 2015. [Online]. Available: https://bit.ly/2NFEZr2

[32] The Apache foundation. Edgent: a community for accelerating analytics at the edge. [Online]. Available: https://edgent.apache.org/

[33] M. Canonico and D. Monfrecola, "CloudTUI-FTS: A user-friendly and powerful tool to manage cloud computing platforms," *EAI Endorsed Transactions on Cloud Systems*, vol. 16, no. 6, 1 2016.

[34] C. Anglano, M. Canonico, and M. Guazzone, "Prometheus: A flexible toolkit for the experimentation with virtualized infrastructures," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 11, 2018.