Journal of Computer and Knowledge Engineering, Vol. 3, No. 1, 2020.
DOI: 10.22067/cke.2020.63251.0

A Multi-objective Dynamic Scheduling Approach for IoT Task Offloading on Amazon EC2 Spot Instances

Arash Deldari *1 Ghasem Akbarian 2 and Mostafa Sabzekar 3

Abstract-- In the recent years, the internet of things (IoT) applications have affected many aspects of human life due to the continuous innovation in hardware, software, and communication technologies along with the growth of the connected devices by the day. The enormous amounts of data generated by these devices must be stored and then processed for analysis and decision-making. One promising solution for this purpose is cloud computing. However, offloading tasks to the cloud imposes costs that must be reduced by intelligent techniques and optimization algorithms. Fortunately, considering cloud computing instances with dynamic pricing, referred to as spot instances, can significantly reduce the processing costs. Although, these instances offer a considerable price saving compared to on-demand instances, they can be evicted by the cloud providers and require special scheduling techniques. In this paper, we propose a dynamic scheduling method for IoT task offloading on Amazon EC2 spot instances. The proposed method considers both task's predicted execution time and deadline to specify tasks that can be mapped on spot instances. The experimental results denote that the proposed method leads to a considerable reduction in the execution costs. Simultaneously, it increases the number of successful tasks executed before the deadline and decreases task turnaround time.

Keywords. Index Terms - Cloud computing. Internet of Things. Spot instances. Task offloading

1. Introduction

Nowadays, significant and rapid advances in information and communication technology (ICT) have changed human lives in different aspects [1]. Internet of things (IoT) plays a fundamental role in the ICT sector. It is estimated that the IoT industry indicates almost $949.42 billion by the start of the year 2025 [2]. The IoT paradigm refers to a pattern of unique addressable chain objects that can include real and physical tools, including sensors, tags, cars, smartphones, and so on [3]. The data collected by IoT tools can be transferred over the internet and processed. Therefore, data processing is considered as one of the main requirements of IoT. One of the main challenges in IoT is cost reduction, which is guaranteed by optimizing operating costs [4]. In some cases, IoT devices face challenges such as limited processing power, storage, and battery constraints. In this situation, transferring tasks to third-party computing platforms such as the cloud is inevitable, which is referred to as task offloading [5].

Another issue in IoT is to consider time constraints because each task should be executed before the deadline. Deadline constrained tasks are time-limited tasks that must be completed, and the results must be returned to the user before the deadline expires. At the core of this technology, cloud computing can be used to provide the required processing and storage services. Cloud computing is a model for universal, convenient, and on-demand access to a network of computing resources that changes the task execution model from traditional models to a pay-per-use approach [7]. It offers storage and processing requirements for a wide range of applications such as bag of tasks (BoT), workflows, web applications, real-time tasks, and so on [9]–[11]. However, the successful execution of the IoT tasks without deadline violations must be taken into account. Thus, presenting an appropriate scheduling algorithm for IoT task offloading that considers the issues as mentioned earlier is an exciting research topic [5], [6].

Major service providers such as Microsoft, Google, and Amazon deploy cloud resource management systems in distributed data centers worldwide. They offer services according to their unique rules. Some cloud service providers offer a special type of auction-based pricing model for the idle resources in their data centers to maximize their profits. The price of these resources varies with time according to supply and demand rates. The most popular cloud service providers that offer this type of service include Google cloud preemptible virtual machines (VMs), Amazon EC2 spot instances, and Microsoft Azure low priority VMs [12]–[14].

Amazon offers spot instances across different geographic regions with different memory, processing, and operating system features as different VM types. The significant cost saving offered by the Amazon EC2 spot instances has increased the popularity of spot instances in the academy and industry. Therefore, many types of research have focused on the use of spot instances in task scheduling [15], [16]. Due to the dynamic pricing of spot instances, offering an appropriate bid requires a proper price prediction approach. Thus, this context has received widespread attention from researchers [17]–[19]. The price offered in these works is specific to geographic regions, meaning that each task has a geographical range, and the number of available instances is limited [11]. One of the main challenges that should be considered is the reduced availability of the spot instances due to their auctioning essence. This unguaranteed availability of spot instances can affect task execution in several ways. For example, if the execution of a leased spot instance is terminated unexpectedly, a proper solution should be proposed so that the computations of the executing job remain safe and the execution recommences so that the task's deadline is not threatened.

* Manuscript received April, 10, 2020; accepted. December, 14, 2020. .

1 Department of Computer Engineering, University of Torbat Heydarieh, Torbat Heydarieh, Iran. Email: adeldari@torbath.ac.ir.
2 Department of Computer Engineering, Imamreza International University, Mashhad, Iran.
3 Department of Computer Engineering, Birjand University of Technology, Birjand, Iran.

Task scheduling is generally divided into dynamic and static categories. In static scheduling methods, all the tasks are submitted to the system before execution. Therefore, the schedule map is determined before the execution begins and does not change during the execution. However, in dynamic scheduling methods, the tasks and their entry time are not specified before execution. Thus, the tasks are submitted to the system in a dynamic manner, and the schedule map might change during execution.

In this research, we propose a dynamic scheduling algorithm for IoT task offloading on Amazon EC2 spot instances. The proposed scheduling algorithm functions in a way that the scheduling procedure is conducted as soon as a new task is submitted to the system, which in some cases might change the current schedule map. Accordingly, tasks with different CPU and memory requirements are scheduled on resources, and the execution time of the tasks is predicted using machine learning techniques. The IoT tasks considered in this study are very similar to the bag of tasks (BoT) since there are no data communications between them. On the other hand, these tasks have user-defined deadlines to make them more similar to real-world IoT tasks.

To increase the successful execution of the submitted tasks, both on-demand and spot instances are considered in the scheduling process. The main goal is to minimize the execution fees by scheduling as many tasks as possible on spot instances, and the decision will be made using the task's predicted execution time and deadline. We also consider powerful on-demand instances based on the task's requirements at specific times, which increases the success rate of the scheduling algorithm. Besides, a virtual machine migration approach based on the predicted execution time is proposed, which acts in case of resource retrieval. We utilize the LSTM method proposed in [18] to predict the future price of spot instances with high accuracy. In this context, the worst-case scenario occurs when all of the acquired spot instances are terminated unexpectedly. Nevertheless, the execution cost of the proposed algorithm will be dramatically reduced due to the increased utilization of the leased on-demand instances.

The main contributions of this research are as follows:

- It proposes a novel dynamic scheduling algorithm for the execution of IoT tasks considering Amazon's on-demand and spot instances.
- It provides an IoT task classification method based on the IoT task's memory and processing requirements. The dynamic scheduling method considers the presented task's classification in the scheduling procedure.
- Finally, it presents a novel dynamic migration method in the event of spot instance eviction to improve the system's reliability using the predicted execution time.

2. Literature Review

One of the significant advantages of Amazon EC2 spot instances is the considerable price saving offered to the users compared to on-demand instances. Task scheduling and resource management on cloud processing instances are considered as an NP-complete problem [20]. However, utilizing Amazon's auction-based instances leads to challenges created due to the possibility of resource retrieval in case the service provider needs the processing capacity. Moreover, smart bidding strategies must be defined that specify the user's maximum willing price according to the importance of the task. In other words, offering higher bids for sensitive tasks decreases the possibility of resource retrieval, which in turn enhances reliability.

In this section, we briefly summarize the state-of-the-art methods that consider spot instances in scheduling and resource management in literature. A wide range of research has been conducted to predict the price of the spot instances that will result in an optimal bid [16], [17], [21]. Resource migration is another policy to increase the system's reliability that is popular among researchers [22], [23]. Other research has addressed the challenge of checkpointing to provide fault-tolerance for the application's considering spot instances [24], [25]. Most of the research seeks to calculate the checkpoint timing and investigate the resuming of the task in the case of resource revocation. CPU scheduling and task queuing, considering spot instances is another hot topic in the literature [15], [26].

The significant growth of IoT applications and the heterogeneity of IoT tasks intensifies the importance of scheduling algorithms and load balancing techniques [27]. Kabir et al. [28] presented a framework for spot price bidding and executing deadline-constrained tasks on cloud spot instances. Their main goal was to bargain for cheaper processing resources to process tasks with loose deadlines. The authors claimed that the proposed system saves an average of more than 80% in execution costs for tasks with loose deadlines. Therefore, the proposed approach is not suitable for tasks with tight deadlines, such as some IoT applications. In similar research [10], a method for scheduling workflows on Amazon's spot instances is presented, which minimizes execution fees with consideration on the deadline constraint. This approach creates a Markov decision-making process for searching the optimal policies to execute tasks according to user quality of service (QoS) requirements such as time and cost. Fabra et al. [29] proposed a framework for analyzing the spot prices, and a classification method is created based on the spot price history, which considers all the geographical regions of the Amazon EC2. The proposed model in this research presents provisioning plans using the long-term price history, and applications created with this model would be able to run with time and cost constraints.

An accurate prediction on spot instance prices can play a significant role in offering an appropriate bid. Therefore, a wide range of research has focused on presenting predictive models on spot pricing [30], [31]. Agarwal et al. [17] predict spot instance prices using the LSTM recurrent neural network techniques. The advantage of this type of neural network is that the error rate is much lower compared to other neural networks due to the long-term memory available in this type of network. However, the accuracy of the proposed method may be improved by considering the timestamp of the spot price. The LSTM network has also been used by other researchers [18], [19], for spot price prediction.

Due to the unexpected termination of spot instances and the lack of stringent service level agreement (SLA) in these instances, extensive research has focused on improving fault-tolerance and reliability for the applications that utilize SIs. Accordingly, different policies such as checkpointing, migration, redundancy, etc. have been considered in literature [16], [25], [32]. In [16], the authors proposed a machine learning method to calculate the timing of performing the checkpointing procedure. In this study, the machine learning method is used to predict the optimal price. In the next step, using the forecasted price in the previous step, considering the defined deadline, and again using machine learning methods, the time of creating a checkpoint is calculated. The proposed approach improves reliability and reduces the cost of performing tasks to an acceptable level.

Domanal and Reddy [15] proposed a modified particle swarm optimization algorithm to assign bag of tasks (BoT) applications to on-demand and spot instances. A back-propagation neural network machine learning was used to estimate the price of spot instances and further uses a migration method to increase the efficiency of the resources. Given the fact that in some IoT applications tasks can be considered similar to a bag of tasks without communications, the proposed method can be utilized in some IoT contexts.

Resource provisioning and scheduling of a wide range of applications in the cloud computing environment have been widely studied in the researches [9]–[11]. In addition, the fog computing concept is considered for IoT applications to solve the problem of data communications latency [33], [34]. However, the context of scheduling IoT applications with loose deadlines on the cloud has not received much attention. On the other hand, considering the service provider's idle instances (i.e., Amazon EC2 spot instances) leads to significant execution cost savings. Therefore, the main goal of the present research is to propose a dynamic scheduling method which faces the challenges of using Amazon EC2 spot instances in the scheduling of IoT tasks with loose deadlines.

3. System Design

In this section, the proposed scheduling algorithm is presented. Two objectives are followed in this research; the first objective is to reduce the execution fees, and the second objective focuses on increasing the number of successfully executed tasks without deadline violation concerning the price paid.

A. Basic Definitions and Assumptions

Due to the extensive scope of the IoT environment, we are considering several specified IoT contexts for the proposed scheduling algorithm. Moreover, because of the diversity of cloud service provider's instances, we consider some predefined Amazon EC2 instances according to our assumptions and requirements. In this paper, we have assumed that the processing and memory requirements of each task submitted by the IoT devices are known as a priori. We have also assumed that tasks come with a user-defined deadline meaning that the execution of the task after the defined deadline is of no value.

The target environment in this research is the IoT applications, but due to the wide range of applications in this field, an initial batch of tasks with different features and specifications is considered. The proposed method is evaluated considering the following application categories (i.e., healthcare applications, intelligent agriculture, smart structures). Depending on the type of task and requirements, each task can be considered as one of the following processing categories:

- Data-intensive
- High data-intensive
- Compute-intensive
- High compute-intensive
- Compute-data intensive

Besides, some predefined cloud instances that are related to the application categories are considered, which will be demonstrated in the evaluation section. Moreover, along with each category, a "low" or "high" priority is also provided by the user, which will be used in the scheduling process. Fig. 1 denotes the architecture of the proposed scheduling algorithm.

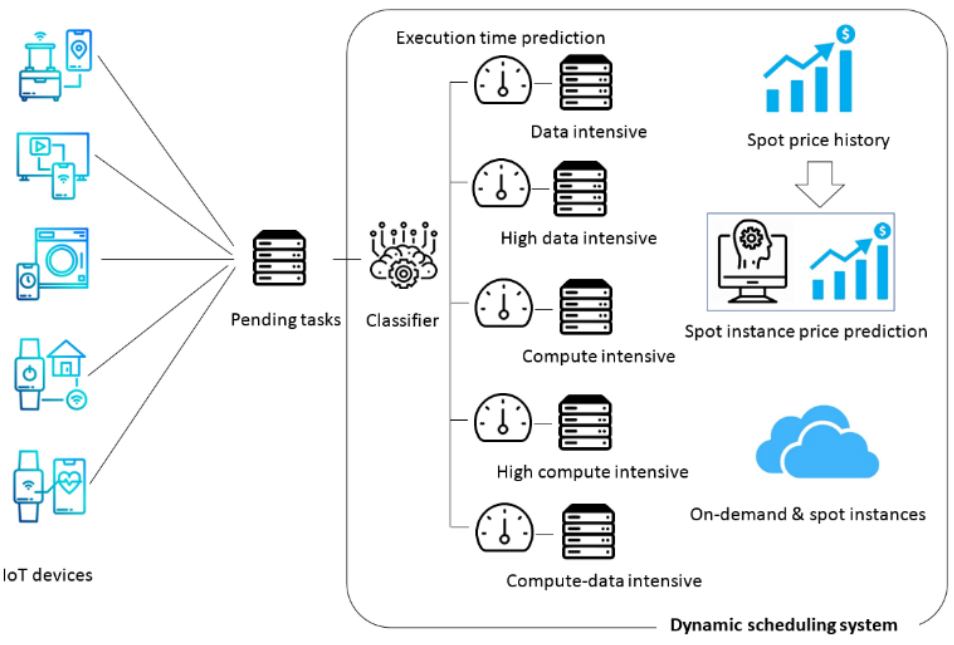

Fig. 1. The architecture of the proposed scheduling method

The aforementioned classification will be performed by the information provided by the user and the context of the submitted task. When a task scheduling request is submitted, it will be transmitted to the cloud along with this information. In other words, the IoT task category determines the task's class and requirements for the scheduling process. As a result, the scheduling procedure will determine the appropriate mapping of IoT tasks on cloud instances.

Data-intensive applications, which, as their name suggests, usually have a higher ratio of data to the amount of processing, which respectively has higher main memory demands. In the high data-intensive category, according to the tasks' requirements twice as much memory has been considered compared to the data-intensive applications, and the number of virtual cores has been doubled, respectively. However, for compute-intensive applications have a higher ratio of processing requirements to the amount of data processed. Therefore, instances with 50 MIPS or less are considered which the Amazon EC2 compute-optimized instances are the best fit. In the high compute-intensive category, applications with high computational requirements are considered. Therefore, instances with 100 MIPS processing power and 32 GBs of main memory are used. Moreover, for compute-data intensive applications, tasks with relatively high computation and data requirements are considered. Respectively for this category, instances with 32 GB of main memory and 50 MIPS of processing power are considered. All categories consider the Amazon elastic block store (EBS) for the secondary memory requirements. The details of the assumed instances will be described in the evaluation section of the paper.

B. Application Model

Typically a task γ_β is considered for processing in IoT, where $\beta = (i,j)$ is a pair consisted of $i = \{1,2,3,4,5\}$ that denotes one of the five categories of the application. Moreover, $j = \{1,2,3,\dots,n\}$ represents the number of times that task γ has been executed. On the other hand, σ_i denotes the instance that is considered for task processing. Therefore, the desired task will be processed on a σ_i instance at cost pi for a period of T. With these notations, the total execution cost to process a task can be calculated as Equation (1):

$$\textit{Execution-cost}\ (\gamma_\beta) = \sum_{j=1}^{n} (\,(T(\sigma_i) \times p_i)) \tag{1}$$

It should be noted that for each execution, the 1-hour time slot of the leased instance might not be used completely. In this case, if the submitted task uses an available instance, there will be no execution costs. The most important challenge in using spot instances is the possibility of resource invocation during task execution. With this interpretation, it is possible to use multiple instances (spot and on-demand) at different prices. Thus, price modeling is described as follows:

$$\textit{Total-cost}\ (\gamma_\beta) = \sum_{j=1}^{m} (\sum_{i=1}^{n} (T(\sigma_i) \times p_{i.t})\,) \tag{2}$$

In this equation, $\sigma_i = \{\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}, \dots, \sigma_{i,n}\}$ denotes the set of processing resources on which the γ_β task will be executed on that resource. Moreover, due to the price fluctuation of spot instances, $p_{r.t}$ denotes the price of the spot instance σ at moment t.

This research also aims to increase the number of tasks that have been completed successfully before the deadline. These tasks can be modeled considering the following equation:

$$\textit{Success}\ (\gamma_{\beta,j}\,) = \frac{\mu_{\beta,j}}{deadline(\gamma_{\beta,j})} \tag{3}$$

In this equation, $\mu_{\beta,j}$ indicates the turnaround time of task γ_β in the *j-th* execution. On the other hand, the output of this equation will have two states; either greater than one (>1) or lower than or equal to one (<=1). If the value of the expression is lower than or equal to one, it means that task γ_β is executed without failure, and the processing result is returned to the user before the deadline. However, if the result is greater than one, it denotes that the execution of the task has failed, and the scheduled task has not been completed before the defined deadline. For example, consider an IoT task with a defined deadline of 5 mins. If the turnaround time of the task is 4 minutes and 35 seconds the output of Eq.3 will be less than one which denotes the successful execution of the aforementioned tasks. However, if the turnaround time of the task exceeds the deadline the outcome of Eq.3 will be greater than one which means the execution of this task has violated the defined deadline. Of course, this relationship considers a single execution (j-th) of the task. Therefore, it is possible to extend the relationship by examining the average failure or failure of a task over several executions.

C. Proposed Scheduling Algorithm

Regarding spot instances for the execution of the IoT tasks, in some cases, it will lead up to a 70-90% reduction in the execution fees [21]. However, to utilize spot instances in the scheduling process, some special considerations must be examined. One of the basic features of spot instances is that there will be no guarantee whatsoever on the start and the termination time of the processing instances. In other words, based on the supply and demand for spot instance capacity and the availability of the instances in different geographical zones, your instance may be terminated unexpectedly during execution.

Before considering the spot instance for the execution of each IoT task, one crucial matter must be carefully studied. It must be monitored that in the case of an unexpected termination of a spot instance during the execution of the task, the user-defined deadline will not be violated. Thus, executing a task with a tight deadline on a spot instance increases the risk of task failure. Moreover, according to the Amazon EC2's spot instance specifications, in case the service provider decides to terminate the instance before the end of the 1-hour time slot, there will be a warning 2 minutes before the termination of the instance [35]. Consequently, in the proposed algorithm it is assumed that this period will be sufficient to launch a new on-demand instance in case of revocation of the spot instance. Additionally, one of the other specifications of the spot instances is that if the instance is terminated before the 1-hour slot ends, users will not be charged for the last time slot.

To overcome the limited bandwidth of the internet and to minimize the overhead of data transmission to the processing resources, researchers have suggested using the fog computing (or edge computing) concept for IoT processing [33], [34]. However, we have assumed cloud instances in geographical regions that are closer to the IoT objects. Considering this policy in the scheduling procedure reduces the latency of the data transmissions between the devices and the cloud.

Algorithm 1 represents the pseudo-code of the proposed scheduling algorithm.

Algorithm1- The scheduling algorithm

```
1. Set timer and add γ to the pending tasks queue;
2. Category(γ) =Task classification and queuing;  /*(DI, HDI, CI, HCI, and DCI ) */
3. if  (Running_Number(γ) == 1)
        Create γ's task table;
        Schedule_OnDemand(γ);
   else
        Machine_Learning (γ's Task Table, ID γ) /*Predict the IoT task's execution time */
        Repeat
            Schedule(γ);
            Random_Wait(5);     /* Wait for a random time of maximum 5 mins if not scheduled */
        Until(γ is scheduled);
end if
```

The scheduling steps of the proposed algorithm are as follows:

Step 1: The pending tasks are classified according to the IoT context. Therefore, a timer is set up to save the entry time of the task. This timer will contain the task ID and time stamp, which will be used in the scheduling algorithm as µ.

Step 2: After fulfilling the classification of the tasks in step1, the tasks are inserted into their respective queues. Here, we assume there are five different working categories. The reason for considering this policy is to not miss a task because of a busy processor. Moreover, considering five queues for the assumed IoT tasks minimizes the dependency of the tasks in different categories on each other. Considering a single queue for all the submitted tasks may lead to the following scenario; the processing resource that is considered for the execution of the task in front of the queue may be busy, but other processing resources are idle. In this case, an IoT task from a different context increases the turnaround time of the tasks from other contexts. Therefore, tasks with different requirements from different IoT contexts are scheduled independently, and considering multiple queues in the scheduling process increases the utilization of the processing resources. Moreover, scheduling tasks regarding this policy increases the average number of successful tasks. This research concentrates on task scheduling in the IoT environment, and inherently IoT applications are real-time. Thus, they must be executed before any time is wasted and time constraints are compromised. Consequently, each task is ready to be executed after entering the system.

Step 3: Since each pending task enters the related queue, it is sent to the machine learning step to predict the required execution time. The reason for doing so is that tasks do not wait for the machine learning step when it comes to execution. If a task has been submitted to the system for the first time, it will be executed on an on-demand instance. The reason for considering this strategy is to minimize the possibility of the failure of tasks without any execution history. The important thing to keep in mind is that as the number of executed tasks in each category increases, the accuracy of the machine learning method will increase. The proposed scheduling algorithm uses the actual execution time after the execution is fulfilled as training data for the neural network. At the same time, the instance type, geographical region, and the current time are sent to the LSTM neural network to predict the price of the instance as the user's bid. The LSTM network presented in [18] that has been used for the price prediction of the spot instances has been trained with Amazon EC2's 3-month price history.

Step 4: Algorithm 2 represents the pseudo-code of the scheduling procedure for each IoT task. In this step, based on the task's estimated execution time and the defined deadline, the mapping of the tasks to the processing resources is determined using the method described below:

1. If the value of *(deadline – wait_time)*, which is considered as slack time is greater than or equal to *(predicted_executiontime x 2)*, the task is considered safe to be executed on a spot instance. The reason we have considered this equation is that if the task's execution fails due to spot instance eviction, there will be enough time to reschedule the failed task. The worst-case occurs when exactly the resource is evicted at the end of the task execution and before the results are saved. Therefore, there will be enough time to reschedule the tasks either on a new spot instance or an on-demand resource. It should be mentioned that if the required spot instance is available, then the mapping of the task is fulfilled on the available spot instance. However, this is only possible if the resource is available until the intended task is executed. Otherwise, the algorithm bids for a new spot instance or sends a request for an on-demand instance.
2. If the slack time is less than *(predicted_executiontime x 2)* but larger than or equal to the predicted execution time, then the priority of the task is considered to define the type of the resource for execution. If the defined priority of the task is high, then the task is scheduled on an on-demand instance to avoid the failure of high priority tasks due to the possibility of resource eviction in spot instances. However, if the priority of the task is low, then the scheduling algorithm considers the task to be scheduled on a spot instance. If the required spot instance is available and scheduling the task on the spot instance is possible considering the deadline, then the task is mapped to be executed. However, if the required spot instance is not available or scheduling the task is not possible according to scheduling constraints, the algorithm tries scheduling the desired task on a new spot instance.
3. If the computed slack time is lower than the task's predicted execution time, then this task is scheduled on a critical instance. Accordingly, this policy decreases the possibility of failure in tasks than are at risk of deadline violation.

The primary objective of the proposed scheduling algorithm is to schedule tasks on spot instances to reduce costs. Moreover, the proposed scheduling algorithm tries to avoid the failure of a task, which in turn increases the successfully executed tasks of the system.

Algorithm2- The scheduling procedure for each task

```
if (Slack_Time(γ) >=Predicted_ExeTime(γ)x2)
        Schedule_Available(γ);      /*Try to schedule using the
        available instances*/
        Price_LSTM(category(γ));  /*Use the LSTM neural network to
        offer a bid for a new spot instance*/

else if (Slack_Time(γ) >Predicted_ExeTime(γ))
    if (Task_Priority==HIGH)
            Schedule-Available(γ);      /* Try to schedule using the
             available instances */
            Schedule_OnDemand(γ );   /* Launch an on-demand
             instance if not scheduled on available instances */

    else
            Price_LSTM(category(γ));     /*Try to schedule low priority
             tasks on spot instances */
    endif
else
        Schedule_Critical(γ);      /*Schedule tasks with risk of deadline
        violation on a critical instance */
endif
```

D. Task Migration

In this study, since we utilized spot instances, it is likely that it will be retrieved during the processing of a task. Therefore, it is necessary to present an approach that minimizes the number of failed tasks, and simultaneously does not increase the processing overhead and costs of the system. Thus, after receiving the 2-minute warning due to resource eviction, the waiting time of the task is deducted from the defined deadline. According to the computed value *(deadline – wait_time)*, two different policies are considered:

1- If the calculated value is larger than the predicted execution time, then the scheduling algorithm reschedules the task all over again.

2- If this value is smaller than or equal to the remaining execution time, it denotes that the waiting time of the task within the system was too high. However, if we schedule the task on an on-demand from the beginning, the defined deadline for the tasks will be violated. Accordingly, the task is migrated to a critical instance to resume processing. However, because of the high cost of a critical instance compared to other instances, the scheduling algorithm tries to maximize the utilization of the rented critical instance. Therefore, while using the critical instance other tasks may be scheduled on the critical instance so that the leased period in fully utilized.

In this research, a machine learning approach is considered to predict the price of spot instances. Therefore, the LSTM neural network-based method proposed in [9] is implemented to predict spot instance prices.

4. Performance Evaluation

In this section, experimental results and the performance evaluation of the proposed scheduling algorithm is presented.

A. Experimental Setup

To evaluate the proposed scheduling algorithm, the modeling and simulation are conducted using the CloudSim toolkit [36]. The CloudSim toolkit is an open-source cloud simulator developed by the CLOUDS laboratory at the University of Melbourne. This simulator provides classes for defining data centers, virtual machines, applications, users, processing resources, and policies for managing different components of a scheduling system. These components can be put together by users to evaluate new strategies for deploying cloud computing platforms (policies, scheduling algorithms, load balancing, mapping policies, and so on).

1) Spot Price History Dataset

Most service providers, present a price history for spot instances that have been gathered and also made available on the web as datasets [37]. Amazon EC2 offers a 90-day price history on spot instances that can be used for analysis [38]. However, this price history is also provided by third parties which in some cases are beyond 90 days [39]. The spot price history presents the price of a particular instance in a specific geographical region over a specified period that can be modified by the user (i.e., last day, last week, last 90 days, etc.). We have considered the dataset provided by Amazon EC2 to train the LSTM neural network to predict prices within the scheduling algorithm. The scheduling algorithm uses the predicted price as the user's bid to request a spot instance. Table 1 presents the instances that have been considered in the scheduling process.

Table 1. The assumed instances in the scheduling algorithm

Instance	RAM(GB)	vCPU	ECU	On –demand price
r5.xlarge	32	4	19	$0.436
r5.2xlarge	64	8	37	$0.872
c5.2xlarge	16	8	39	$0.708
c5.4xlarge	32	16	73	$1.416
m4.2xlarge	32	8	26	$0.768
c5.9xlarge	72	36	139	$3.186

To schedule the IoT tasks, the r5.xlarge and r5.2xlarge are assumed for the data-intensive and high data-intensive categories. Also, the c5.2xlarge and c5.4xlarge are assumed for the execution of compute-intensive and high compute-intensive tasks. Moreover, the m4.2xlarge instance is considered for the compute-data intensive category. Finally, the c5.9xlarge is assumed as a critical instance for the scheduling procedure.

2) IoT Tasks Dataset

Due to the execution of the IoT tasks on cloud instances in the proposed system, we will need specific configurations according to the task's requirements. Every IoT application has its requirements that must be considered in the scheduling process. Therefore, we considered using the IoT tasks dataset provided online [40] for the evaluation procedure. This dataset offers IoT tasks from various domains such as agriculture, healthcare, and smart city. However, we have added task priority and task category to the tasks in this dataset.

B. Evaluation

The performance of the proposed scheduling method has been evaluated, considering the following criteria:
1. Execution cost
2. Number of successful tasks
3. Task turnaround time

We will compare the proposed method with the modified particle swarm optimization (MPSO), and modified throttled methods proposed in [15]. Although the proposed methods in this research consider the bag of tasks (BoT) application, the IoT tasks that we have assumed in this paper are similar to BoT since there are no data communications between them. Moreover, we will also study these criteria using migration-enabled and migration-disabled modes.

1) Execution Cost

To show the cost savings, the cost of task execution in each category is compared with the method that only considers the scheduling of the tasks on on-demand instances. As can be seen, the migration-enabled method leads to a more significant reduction in execution costs. The reason for the reduced cost is the policy that prevents the re-execution of failed tasks due to spot instance interruption. In the migration-enabled mode, two different scenarios can be imagined for a task being processed in the event of spot instance eviction depicted in the task migration section.

The main goal of the scenarios mentioned above is to ensure that tasks with high priority are executed before the deadline constraint. In these circumstances, some essential issues must be identified and considered. First, the assumed critical instance is this study which has the highest cost compared to all the other assumed instances. These instances are used to minimize task failures and are utilized for specific high priority tasks with limited deadlines. Secondly, eliminating a task wastes the cost of processing a task, meaning that the cost required to process a task on on-demand instances will be wasted. Third, scheduling and executing a task from the beginning will increase the turnaround time and create surplus execution costs. For this reason, in the conditions outlined in the proposed method, the migration of a task will be preferred to reduce execution costs.

In general, the proposed method performs its utmost to execute tasks on spot instances. It is worth noting that although the initial classification is performed, at the moment of task mapping, the resources of the other categories are also examined. The goal is to increase resource utilization and reduce total execution costs. Fig. 2 compares the execution costs of the proposed method considering on-demand and spot instances in the defined task categories.

the scheduling process. The results shown in this diagram are the average of 5 runs, and the number of tasks considered in each run is between 9000 and 10000 tasks. The IoT applications are mainly healthcare and agriculture. For instance, considering the data-intensive category, the proposed method without migration leads to a 23% reduction in execution costs compared to scheduling with on-demand instances. However, the proposed method with migration leads to a 37% cost saving in this category. The reason that the proposed method with migration yields higher cost savings is that incomplete tasks due to spot instance eviction are not executed from the beginning. Therefore, the number of required on-demand instances and leased periods are reduced which leads to more cost savings compared to the approach without migration. In the compute-intensive category, the proposed method earns 13% and 29% savings compared to on-demand scheduling.

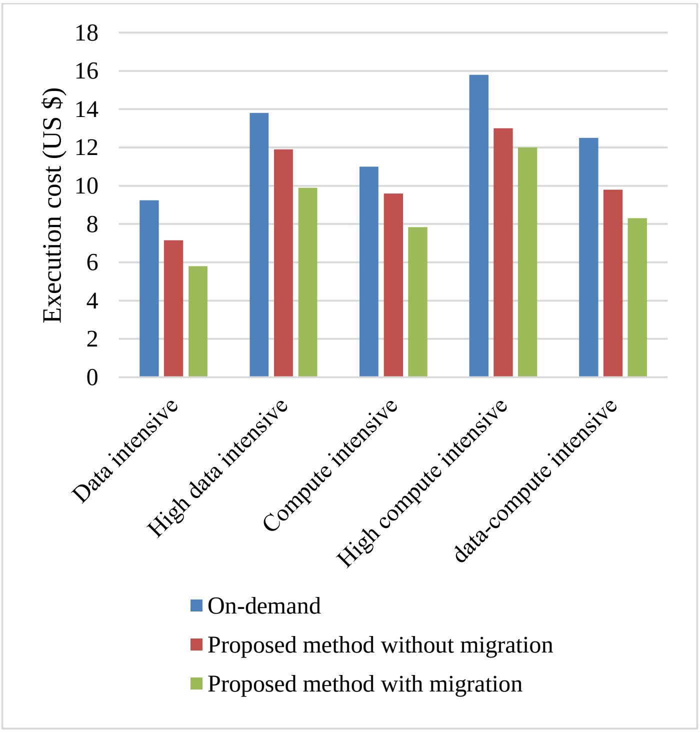

Fig.2. The execution costs of considering on-demand and spot instances

2) Number of Successful Tasks

One of the main goals of the proposed scheduling algorithm is to maximize the number of tasks executed before the deadline constraint. Accordingly, some important issues require attention:

1. In the modified particle swarm optimization method [15], the processes are sent as task stacks consisted of at least ten tasks to the load balancer. However, sending tasks in stacks can significantly increase resource utilization and prevent resources from being wasted and idle. Furthermore, since this approach may lead to an increased waiting time for the tasks, it is not appropriate for IoT tasks with tight deadlines.

2. Migration is done when the processor and memory efficiency is lower than a specified threshold, as detailed in [15]. The modified throttled method also does not include a plan for migrating tasks. However, resource retrieval is possible when using spot instances, and a mechanism must be considered for these conditions. With the dynamic migration method, the successful execution of the tasks can be increased significantly.

3. Another point is that the two methods mentioned in [15] do not differentiate between the submitted tasks and their requirements. The structure of the input tasks described in these two methods, differs in that the tasks in the modified particle swarm optimization method are sent in batch while the other method considers the serial transmission of the tasks to load balancers, and the requirements of the tasks are ignored. However, if we consider these methods in environments with different processing requirements such as IoT, the resource utilization will be reduced, and some processes will not be processed before the deadline constraint. Given the above, the number of successfully executed tasks can be seen in Fig. 3.

The total number of IoT tasks submitted to the system in this experiment is assumed as 1000, and as denoted in Fig. 3, the proposed method without migration achieves a 2% and 3% increase in the number of successful tasks compared to MPSO and modified throttled methods. Moreover, the proposed method with migration receives a 3% and a 4.1% increase compared to the aforementioned methods. As mentioned in the previous section, the proposed method with migration does not reschedule the tasks in the case of spot instance eviction. Therefore, the number of tasks that violate the defined deadline reduces considering the migration policy. The experimental results denote that the proposed method with migration achieves a 98.1% success rate in scheduling tasks before the defined deadline.

3) Task Turnaround Time

Given that the tasks are processed batch in the modified particle swarm optimization method [15], we have also considered this approach. In the modified throttled method [15], the tasks are executed as soon as the tasks enter the load balancer and the resources are idle, which leads to a reduced task turnaround time. In the modified particle swarm optimization method, the amount of time required to categorize tasks is relatively high. Accordingly, due to the nature of the IoT environment in which tasks are submitted at random times, it will result in increased task turnaround time. Generally, since the proposed method processes the tasks as soon as they enter the system, the turnaround time of the proposed method is less than the MPSO method [15]. However, in some cases a short time is required for the machine learning algorithm to offer a bid and for the spot instance to become available, which slightly increases the task turnaround time of the proposed method. It must be mentioned that this reduces the execution fees, which makes this acceptable. Predicting the execution time of tasks can increase the turnaround time, but in the proposed method, the task table is sent to the machine learning algorithm as soon as the task enters the related queue so as not to waste time on CPU allocation. Fig. 4 shows the turnaround time of tasks within the system.

The number of assumed tasks is 1000 and the results denoted in Fig. 4 are the average of 5 runs. Although the modified throttled method leads to a lower turnaround time compared to the proposed method without migration, the proposed method achieves a less turnaround time compared to MPSO. As predicted, the turnaround time decreases considering the migration approach in the scheduling procedure.

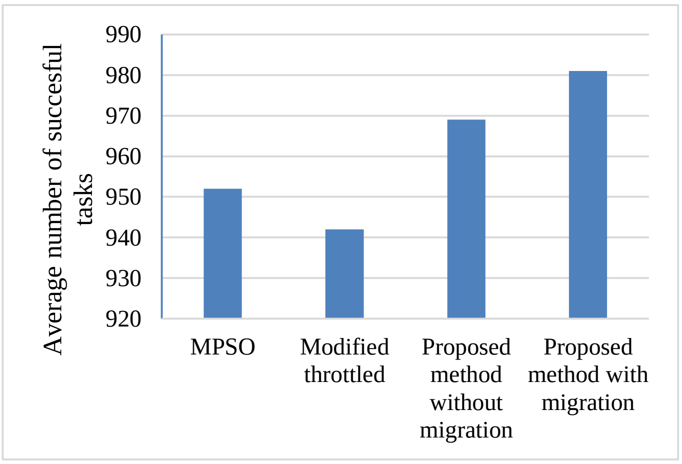

Fig.3. The number of executed tasks before the deadline

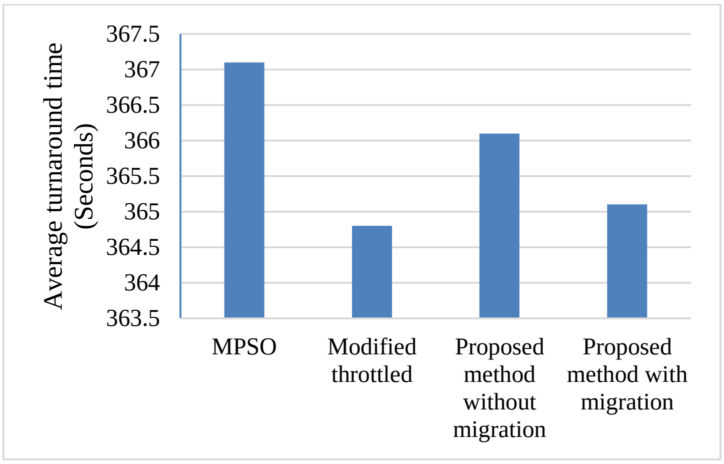

Fig.4. The average task turnaround time

5. Conclusion and Future Work

In this research, a dynamic scheduling algorithm is proposed that increases the reliability of executing IoT tasks with deadline constraints on unreliable cloud instances such as Amazon EC2 spot instances. In the proposed scheduling method, we pursue two goals. The first goal is to reduce the execution costs and the second goal is to increase the number of successful tasks executed before the deadline. Since the main purpose of this work was not price prediction, the proposed methods in literature have been considered for the price prediction step. Experimental results denote that 98.1% of input tasks to the presented scheduling system are processed before the deadline. On the other hand, one of the economic challenges of the IoT environment, namely lower execution costs, improves significantly on this system. The proposed method shows improvements than the most recent methods in this context in terms of the number of successful tasks, turnaround time, and execution costs. In other words, despite the reduction in execution costs, this method leads to the lower turnaround time for the scheduled IoT tasks.

As for future directions, the initial pre-processing of the IoT tasks can be considered in edge computing resources and then scheduled on cloud computing resources for further processing. Considering AmazonEC2 spot blocks and reserved instances in the scheduling procedure can be considered as future work. Price forecasting can also be introduced as future work using intelligent methods that offer price bids according to the task's priority to decrease the possibility of out of bid failures for sensitive tasks.

6. References

[1] M. Bhatia and S. K. Sood, “Exploring Temporal Analytics in Fog-Cloud Architecture for Smart Office HealthCare,” *Mob. Networks Appl.*, vol. 24, no. 4, pp. 1392–1410, Aug. 2019, doi: 10.1007/s11036-018-0991-5.

[2] M. Bhatia, S. K. Sood, and S. Kaur, “Quantumized approach of load scheduling in fog computing environment for IoT applications,” *Computing*, 2020, doi: 10.1007/s00607-019-00786-5.

[3] M. Conti, A. Dehghantanha, K. Franke, and S. Watson, “Internet of Things security and forensics: Challenges and opportunities,” *Future Generation Computer Systems*, vol. 78. Elsevier B.V., pp. 544–546, 01-Jan-2018, doi: 10.1016/j.future.2017.07.060.

[4] A. Čolaković and M. Hadžialić, “Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues,” *Comput. Networks*, vol. 144, pp. 17–39, 2018.

[5] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, “Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing,” *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, 2019.

[6] M. Aazam, S. Zeadally, and K. A. Harras, “Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities,” *Futur. Gener. Comput. Syst.*, vol. 87, pp. 278–289, 2018.

[7] P. M. T. Grance, “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,” *Recomm. Natl. Inst. Stand. Techology*, vol. 145, p. 7, 2011, doi: 10.1136/emj.2010.096966.

[8] D. Kumar, G. Baranwal, Z. Raza, and D. P. Vidyarthi, “A Survey on Spot Pricing in Cloud Computing,” *J. Netw. Syst. Manag.*, vol. 26, no. 4, pp. 809–856, Oct. 2018, doi: 10.1007/s10922-017-9444-x.

[9] C. K. Swain, N. Saini, and A. Sahu, “Reliability aware scheduling of bag of real time tasks in cloud environment,” *Computing*, vol. 102, no. 2, pp. 451–475, 2020.

[10] R. G. Martinez, A. Lopes, and L. Rodrigues, “Planning workflow executions when using spot instances in the cloud,” in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing - SAC ’19*, 2019, pp. 310–317, doi: 10.1145/3297280.3297313.

[11] H. M. Nguyen, G. Kalra, T. J. Jun, S. Woo, and D. Kim, “ESNemble: an Echo State Network-based ensemble for workload prediction and resource allocation of Web applications in the cloud,” *J. Supercomput.*, vol. 75, no. 10, pp. 6303–6323, 2019.

[12] “Amazon EC2 Spot – Save up-to 90% on On-Demand Prices.” https://aws.amazon.com/ec2/spot/ (accessed Mar. 12, 2019).

[13] "Preemptible VMs - Compute Instances | Google Cloud." https://cloud.google.com/preemptible-vms/ (accessed Apr. 16, 2019).

[14] "Announcing low-priority VMs on scale sets now in public preview | Azure Blog and Updates | Microsoft Azure." https://azure.microsoft.com/en-us/blog/low-priority-scale-sets/ (accessed Mar. 28, 2020).

[15] S. G. Domanal and G. R. M. Reddy, "An efficient cost optimized scheduling for spot instances in heterogeneous cloud environment," *Futur. Gener. Comput. Syst.*, vol. 84, pp. 11–21, 2018, doi: 10.1016/j.future.2018.02.003.

[16] A. K. Mishra, D. K. Yadav, Y. Kumar, and N. Jain, "Improving reliability and reducing cost of task execution on preemptible VM instances using machine learning approach," *J. Supercomput.*, vol. 75, no. 4, pp. 2149–2180, Apr. 2019, doi: 10.1007/s11227-018-2717-7.

[17] S. Agarwal, A. K. Mishra, and D. K. Yadav, "Forecasting price of amazon spot instances using neural networks," *Int. J. Appl. Eng. Res.*, vol. 12, no. 20, pp. 10276–10283, 2017.

[18] M. Baughman, K. Chard, I. Foster, R. Wolski, and C. Haas, "Predicting Amazon Spot Prices with LSTM Networks," pp. 1–7, 2018, doi: 10.1145/3217880.3217881.

[19] H. Al-Theiabat, M. Al-Ayyoub, M. Alsmirat, and M. Aldwair, "A Deep Learning Approach for Amazon EC2 Spot Price Prediction," in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018, pp. 1–5, doi: 10.1109/AICCSA.2018.8612783.

[20] A. Deldari, M. Naghibzadeh, and S. Abrishami, "CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud," *J. Supercomput.*, vol. 73, no. 2, pp. 756–781, 2017.

[21] Z. Cai, X. Li, R. Ruiz, and Q. Li, "Price forecasting for spot instances in Cloud computing," *Futur. Gener. Comput. Syst.*, vol. 79, pp. 38–53, 2018, doi: 10.1016/j.future.2017.09.038.

[22] S. Shastri and D. Irwin, "HotSpot: Automated Server Hopping in Cloud Spot Markets Supreeth," *Proc. 2017 Symp. Cloud Comput. - SoCC '17*, pp. 493–505, 2017, doi: 10.1145/3127479.3132017.

[23] Z. Shen, R. van Renesse, Q. Jia, H. Weatherspoon, and W. Song, "Smart spot instances for the supercloud," vol. 1, no. 212, pp. 1–6, 2016, doi: 10.1145/2904111.2904114.

[24] I. Jangjaimon and N. F. Tzeng, "Effective Cost Reduction for Elastic Clouds under Spot Instance Pricing Through Adaptive Checkpointing," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 396–409, 2015, doi: 10.1109/TC.2013.225.

[25] S. Yi, A. Andrzejak, and D. Kondo, "Monetary Cost-Aware Checkpointing and Migration on Amazon Cloud Spot Instances," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 512–524, 2012, doi: 10.1109/TSC.2011.44.

[26] D. Poola, K. Ramamohanarao, and R. Buyya, "Fault-tolerant workflow scheduling using spot instances on clouds," *Procedia Comput.* Sci., vol. 29, pp. 523–533, 2014, doi: 10.1016/j.procs.2014.05.047.

[27] S. Basu *et al.*, "An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment," *Futur. Gener. Comput. Syst.*, vol. 88, pp. 254–261, Nov. 2018, doi: 10.1016/j.future.2018.05.056.

[28] H. M. Dipu Kabir, A. S. Sabyasachi, A. Khosravi, M. A. Hosen, S. Nahavandi, and R. Buyya, "A cloud bidding framework for deadline constrained jobs," in *Proceedings of the IEEE International Conference on Industrial Technology*, 2019, vol. 2019-February, pp. 765–772, doi: 10.1109/ICIT.2019.8755137.

[29] J. Fabra, J. Ezpeleta, and P. Álvarez, "Reducing the price of resource provisioning using EC2 spot instances with prediction models," *Futur. Gener. Comput. Syst.*, vol. 96, pp. 348–367, Jul. 2019, doi: 10.1016/j.future.2019.01.025.

[30] V. Khandelwal, A. Chaturvedi, and C. P. Gupta, "Amazon EC2 Spot Price Prediction using Regression Random Forests," *IEEE Trans. Cloud Comput.*, pp. 1–1, 2017, doi: 10.1109/TCC.2017.2780159.

[31] D. Liu, Z. Cai, and X. Li, "Hidden markov model based spot price prediction for cloud computing," *Proc. - 15th IEEE Int. Symp. Parallel Distrib. Process. with Appl. 16th IEEE Int. Conf. Ubiquitous Comput. Commun. ISPA/IUCC 2017*, pp. 996–1003, 2018, doi: 10.1109/ISPA/IUCC.2017.00152.

[32] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, "SpotCheck: designing a derivative IaaS cloud on the spot market," *Proc. Tenth Eur. Conf. Comput. Syst. - EuroSys '15*, pp. 1–15, 2015, doi: 10.1145/2741948.2741953.

[33] A. A. Mutlag, M. K. A. Ghani, N. al Arunkumar, M. A. Mohammed, and O. Mohd, "Enabling technologies for fog computing in healthcare IoT systems," *Futur. Gener. Comput. Syst.*, vol. 90, pp. 62–78, 2019.

[34] M. Goudarzi, H. Wu, M. S. Palaniswami, and R. Buyya, "An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments," *IEEE Trans. Mob. Comput.*, 2020.

[35] "Amazon EC2 Spot Two-Minute Warning is Now Available via Amazon CloudWatch Events."https://aws.amazon.com/about-aws/whats-new/2018/01/amazon-ec2-spot-two-minute-warning-is-now-available-via-amazon-cloudwatch-events/(accessed Apr. 14, 2019).

[36] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exp.*, vol. 41, no. 1, pp. 23–50, 2011.

[37] "AWS Spot Pricing Market | Kaggle." https://www.kaggle.com/noqcks/aws-spot-pricing-market (accessed Apr. 07, 2019).

[38] "Spot Instance Pricing History - Amazon Elastic Compute Cloud." https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances-history.html (accessed Apr. 07, 2019).

[39] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing Amazon EC2 spot instance pricing," *Proc. -* 2011 *3rd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2011*, vol. 1, no. 3, pp. 304–311, 2011, doi: 10.1109/CloudCom.2011.48.

[40] "25 Datasets for Deep Learning in IoT | Packt Hub." https://hub.packtpub.com/25-datasets-deep-learning-iot/ (accessed Mar. 26, 2020).