

Optimizing Cloud-Fog Workloads: A Budget Aware Dynamic Scheduling Solution

Phuoc Hung Pham[1(✉)], Nguyen Thanh Binh[2], Md. Golam Rabiul Alam[3], and Md. Motaharul Islam[4]

[1] Providence College, Rhode Island, USA
hung205a2@gmail.com
[2] University of Finance – Marketing, Hochiminh, Vietnam
ntbinh@ufm.edu.vn
[3] BRAC University, Dhaka, Bangladesh
rabiul.alam@bracu.ac.bd
[4] United International University, Dhaka, Bangladesh
motaharul@cse.uiu.ac.bd

Abstract. With the rise of demanding computational applications, individuals and businesses reap significant benefits. However, processing power limitations on local devices hinder further progress. These local resources also require substantial financial and human investment. Fortunately, the adoption of cloud computing (CC) platforms alleviates these concerns. CC enables offloading computationally intensive tasks to the cloud, reserving local resources for simpler jobs. However, as CC operates on a pay-as-you-go model, finding an approach that minimizes both execution time and cost is crucial. While research efforts have attempted to address this challenge, solutions remain limited. This paper proposes a novel architecture leveraging cloud provider resources and local computing power on fog computing devices. The core of this framework is a dynamic task scheduling solution that minimizes completion time within the cloud service while considering network conditions and customer costs for service usage. Simulations and comparisons with existing scheduling approaches demonstrate that our proposed method delivers robust performance with significant cost savings for cloud users.

Keywords: Cloud Computing · Fog Computing · Task Scheduling · Distributed System · High Performance Computing

1 Introduction

The advent of the Internet of Things (IoTs) generates a huge demand for big data management. It is estimated that by the year 2025, there will be nearly 40 billion IoT devices in the world, most of which consist of sensors and low-power end devices that are not powerful for data processing [15]. Even with advanced devices, their processing power still falls short for complex business workflows with large data volumes, high frequencies, and intricate structures. Cloud computing (CC) [17] emerges as a solution, shifting heavy data processing and storage to powerful centralized servers in data centers. This

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024, LNCS 14814, pp. 31–46, 2024.
https://doi.org/10.1007/978-3-031-64608-9_3

leaves local devices with minimal tasks, maximizing cloud resource utilization for high performance. However, the challenge of extensive data transmission arises when confronted with bandwidth limitations. Moreover, the inherent system delays hinder the cloud's ability to deliver effective services for real-time applications. This is where the fog computing paradigm becomes pivotal. The benefits of fog computing go beyond just reducing latency for real-time applications. It also holds the key to unlocking intelligent location-based mobility support. With the ever-growing use of smart devices and wearables, maintaining user proximity during resource allocation becomes crucial. This is where the highly distributed and low-latency nature of fog computing shines. Combined with diverse network connections between cloud, fog, and mobile networks, fog seamlessly integrates with 5G technology, a boon for IoT [1] thanks to its high-speed data rates.

But fog computing's impact extends beyond simple data transfer. Vehicular networks can leverage their capabilities for task offloading and execution. These computationally intensive and latency-sensitive operations highlight the importance of bringing computing resources closer to the network edge. By doing so, fog computing tackles latency constraints and reduces the traffic burden on the cloud, paving the way for a more responsive and efficient mobile future.

While fog computing boasts numerous benefits, it's not without limitations. Resource-constrained fog devices, significantly less powerful than cloud servers, necessitate effective resource allocation and service placement methods to ensure end-to-end service based on user preferences like latency, bandwidth usage, and cloud cost. Thus, balancing these trade-offs for cloud-fog collaboration is where task scheduling strategies come in for Cloud Service Customers (CSCs). These strategies aim to distribute workload efficiently, and minimize completion time, as well as cloud cost for resource usage.

However, task scheduling is still a challenge [23]. Large-scale workflows involve diverse workloads, requiring efficient scheduling to map tasks onto processing systems while respecting task dependencies and minimizing completion time. This is a complex problem due to the NP-completeness of scheduling in heterogeneous environments like CC. Previous approaches mainly focused on minimizing execution time (makespan) without considering cost, which is a significant concern for CSCs paying for rented resources. For example, Amazon EC2 [12] charges CSCs according to the number of virtual machines initialized by CSCs and how long they have been used.

In heterogeneous environments like CC, exhaustive searches are impractical. Existing scheduling methods often trade performance for cost-effectiveness. Some prioritize execution time but incur high monetary costs, hindering CSCs from fulfilling demands while processing complex applications. Efficient task scheduling balancing completion time and cloud cost is vital for improved performance, Quality of Service (QoS), and user satisfaction, especially in active environments.

Therefore, in this paper we propose a novel algorithm to optimize task scheduling for dynamic processing systems, allowing continuous task arrival. It considers network contention and cloud service costs, crucial factors for user satisfaction in a collaboration between the cloud network and fog computing resources to elevate QoS. Traditionally,

fog computing clients handled user interface functionalities and offline mobile processing. In our architecture, they leverage their processing power to execute dispatched tasks and store user data when needed. Fog devices comprise heterogeneous resources with storage and processing capacities less powerful than the cloud, e.g., such as routers, access points, and micro data centers. Additionally, there is a broker functioning as a centralized management node between fog devices and cloud resources. Moreover, extensive simulations demonstrate the efficiency and effectiveness of our approach compared to existing methods. The results highlight improved performance and a valuable trade-off between workflow performance and cost-effectiveness.

Subsequent sections of this paper are organized as follows: Sect. 2 equips you with knowledge of existing solutions, while Sect. 3 lays the groundwork by exploring the system description itself. Section 4 confronts the challenge head-on, introducing our unique method for tackling it. Then, Sect. 5 presents the rigorous testing process and its illuminating results. Finally, Sect. 6 delivers the concluding remarks and charts the course for future exploration.

2 Related Studies

This section covers the foundational elements of the proposed method, including the fog computing environment.

The explosion of data from countless IoT devices has exposed limitations in traditional cloud computing, namely slow communication and high monetary cost. To address these challenges, a new paradigm called fog computing has emerged. Imagine it as a bridge between the network edge – sensors, smartphones, etc. – and the distant cloud data centers, forming a three-tier architecture.

Think of the fog layer as a distributed network of mini data centers located near the devices. Instead of sending all data straight to the cloud, fog nodes handle local processing, storage, and tasks like data management and networking. These nodes, although less powerful than their cloud counterparts, offer several advantages.

- Firstly, by processing data locally, fog computing drastically reduces transmission time and bandwidth usage. This is crucial for real-time applications like healthcare, traffic management, and connected vehicles, where even slight delays can be detrimental.
- Secondly, fog computing improves the overall Quality of Service (QoS) by bringing application services closer to users. Consider an air quality monitoring system. Filtering and processing sensor data at the fog layer before sending it to the cloud significantly reduces data volume and network load, leading to faster analysis and actionable insights.

However, it's not all sunshine and rainbows. Bringing services closer to users can also lead to increased network congestion and energy consumption due to more data hops and communication within the network. In conclusion, fog computing offers a promising solution to the challenges of managing massive IoT data. By placing processing power closer to the devices, fog computing reduces latency, improves QoS, and optimizes network resources. Although there are potential drawbacks, task scheduling can mitigate these concerns, paving the way for a smoother and more efficient IoT ecosystem.

Several research proposals have been anticipated for efficient task scheduling in heterogeneous as well as homogeneous systems. Despite numerous research proposals, task scheduling still remains a challenging open problem in heterogeneous CC environments [8, 20]. Several graph template-based task scheduling methods are presented in [9, 11, 13, 18]. The obliviousness of network contention is the major shortcoming of those proposals. Although a network contention-based method of task scheduling is presented in [21], it ignores the monetary cost of cloud customers (CCs) in utilizing cloud resources. Authors in [24] introduce a cost-efficient approach to select the most proper system (private or public cloud) to execute the workflow according to a deadline constraint as well as cost savings. Lingfang et al. in [16] propose budget-conscious scheduling algorithms to satisfy strictly the budget constraint. Juan et al. proposed an algorithm of scheduling in [14], to schedule applications of massive graph processing. The schedule length and cost are considered in that proposal. Sen et al. in [25] and Chandrashekhar et al. in [22] present methods to minimize the schedule length or cost in dynamic cloud computing but do not study trade-off optimality.

Liu et al. [4] present an approach that can reduce service delay in heterogeneous fog networks but the most important cost-latency trade-off problem in heterogeneous fog networks is not considered. In [5], a dynamic tasking scheduling algorithm is proposed to reduce average service delay but the performance of the system is less in a more complex homogenous fog network. Raafat et al. [1] model the scheduling of IoT service requests as an optimization problem using integer programming in hybrid fog-cloud computing to minimize the overall service request latency, however, it does not include the cost usage.

Therefore, the envisioned proposal should take into account all three efficiency factors: i) completion time ii) network contention iii) cost of cloud resources with the intervention of fog computing in a dynamic cloud environment. This motivates us to provide a system that can solve a task scheduling approach for optimality in a dynamic cloud environment while taking into account those factors and navigating these trade-offs in the realm of scheduling strategies.

3 System Architecture

The following section gives an insight into our system architecture to address the issues discussed above:

The system to which the proposed algorithm is applied follows a distributed data flow model. Each application is modeled as a series of application modules or tasks that can communicate with each other. The output of one application module can serve as the input of the other. This data dependency between application modules is modeled as a directed acyclic graph (DAG), in which the vertices denote modules and the edges represent the dependencies among them. The tuple type (TT) of the data stream is represented on these edges. Tuples are data emitted from sensors or other application modules. These data streams contain raw data if emitted from sensors. In the case of the emission from other modules, each of them is labeled with a specific type which determines to what application module the tuple needs to be transmitted. Resources such as fog devices also tend to produce data streams in the form of usage details processed by the information collector to provide information on the state of the devices.

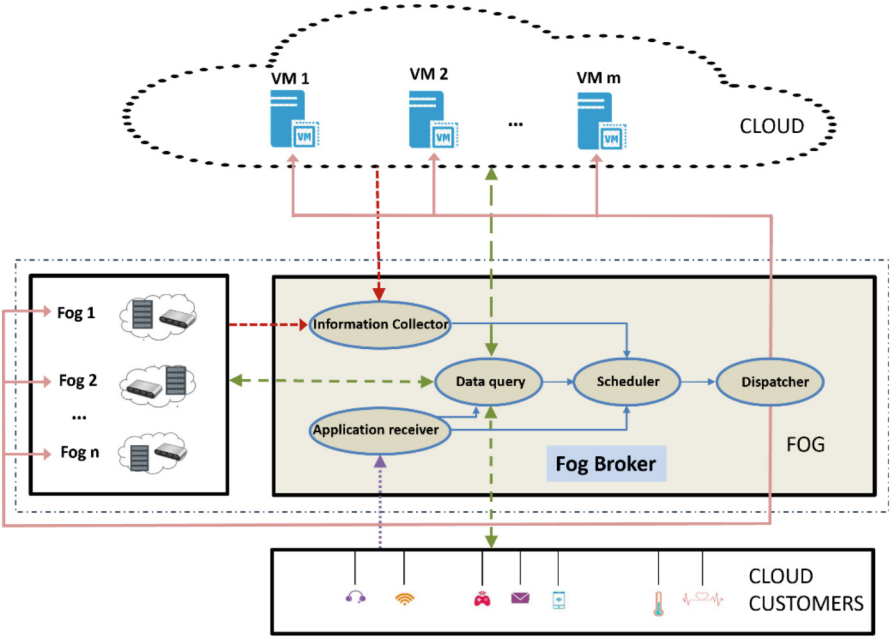


Fig. 1. Layering architecture of the proposal

The proposed architecture as depicted in Fig. 1, consists of three layers i) cloud provider (CP) layer: composed of a pool of VMs, ii) fog layer consisting of fog devices, for example, routers, access points, and micro data centers,…, and iii) customer (CC) layer including IoT devices such as sensors, smartphones, actuators, etc.… The fog layer is structured in a geographically distributed fashion, by placing its resources chiefly within the close vicinity of IoT devices, and consequently of the users. The workload is then divided into smaller chunks, and each chunk is sent from the associated CC to a suitable fog broker or node for processing and is temporarily stored in fog micro data centers as they cannot hold the data permanently due to storage limitations [7]. The main advantage of this approach is the reduction in the transmission time of a data packet as fewer hops need to be crossed, which happens to be crucial in latency-sensitive IoT applications e.g., healthcare, traffic, and connected vehicles. For instance, in an ambient air monitoring system, huge amounts of data gathered from air quality sensors need to be transferred through network layers to get filtered, aggregated, analyzed, and stored. Should these pieces of information get filtered at primary network levels, their volume would decrease, and the less voluminous the data becomes, the lower the network usage will be.

The fog broker acts as a centralized node for management. The functions of the broker in our proposal are as follows: i) It accepts users' requests from IoT devices ii) It manages processing resources (e.g., capacity, cost, and bandwidth) from both fog and cloud, and returns query data. Iii) It prepares an efficient, effective, and optimal schedule concerning input workflow, query data, and resource information. The broker uploads

the data to the CP layer through a single connection but VMs of the CP layer send data to the CC layer through the broker, then the broker divides those data into non-uniform chunks. Afterward, the broker delivers those chunks to corresponding CCs [3, 10].

In the next section, we formulate the problem and describe our proposed approach.

4 Problem Domain

Task scheduling of complex systems having hierarchical topology is analogous to the distribution of jobs or tasks of an application to a group of processors with heterogeneous processing capabilities for fulfilling the optimization goal of minimization of completion time. Therefore, a task graph and a process graph are fed as the inputs of task scheduling. The output is a schedule representing the assignment of tasks to processors.

4.1 Problem Formulation

Before problem formulation, we state here the used terms in the problem formulation. Then a dynamic cost-time aware task scheduling method is presented elaborately.

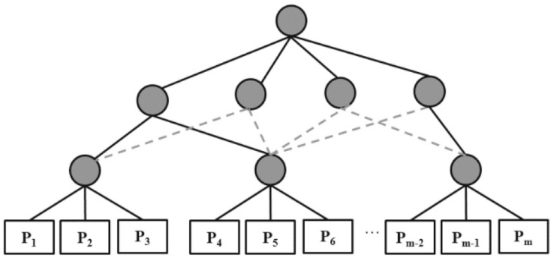


Fig. 2. A processor graph

Definition 1. A processor graph $PG = (N,D)$ demonstrated in Fig. 2 is a graph that represents the network topology between vertices (heterogeneous processors) that are virtual machines (VMs) on cloud servers or fog devices. In this graph, N is the finite set of vertices, and a directed edge $d_{kl} \in D$ means a directed link from vertex P_k to vertex P_l with $P_k, P_l \in N$. Each processor P_k controls the processing rate μ_k and bandwidth on the link connecting it to other processors. Due to the high stability of Local Area Network (LAN) compared with the Internet, the data transfer rate of internal communication among fog devices is always better than that of external communications between cloud VMs.

Definition 2. A task graph, as shown in Fig. 3, is denoted by a Directed Acyclic Graph (DAG) $G = (V, E, W, C)$ where the vertices set $V = \{v_1,v_2,...,v_k\}$ presents the parallel application modules or tasks. The edge $e_{ij} = (v_i,v_j) \in E$ of the DAG symbolized as a communication link between the task v_i and task v_j. The communication time from task

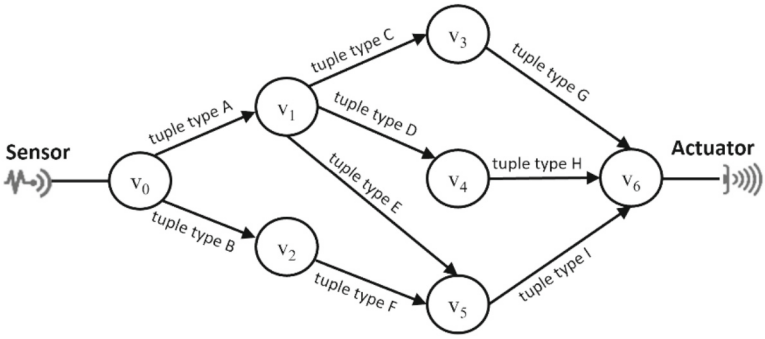


Fig. 3. A sample DAG

v_i to task v_j is denoted by $c(e_{ij})$, whereas $d(e_{ij})$ represents the transferred data. The weight $w(v_{i,} P_k)$ of task v_i is its computation time on processor P_k. It is supposed that a task v_i contains a set of preceding subtasks $prec(v_i)$ and a set of successive subtasks $succ(v_i)$. A task without any predecessors, $prec(v_i) = 0$, is a start-up task v_{entry}, and a task that does not have any successors, $succ(v_i) = 0$, is an end task v_{end}. The workload belonging to task v_i is represented as $l(v_i,P_l)$,, which delimits amount of work (e.g. the number of instructions) processed with certain computing resource P_l. Let $t_s(v_i,P_l)$ denote Start Time of task v_i on processor P_l. Hence, the finish time of that task is given by $t_f(v_i, P_l) = t_s(v_i, P_l) + w(v_i, P_l)$.

A sample representation of an IoT application, consisting of 7 modules ranging from v_0 to v_6 and tuple types A to I, is depicted in Fig. 3. In a sense-process-actuate scenario, all the information emitted from IoT sensors is transferred to module v_0 as raw data, and then the generated output tuples are delivered to modules v_1, v_2, and from v_1 to other modules v_3, v_4, $v_{5,}$ and from v_2 to v_5 respectively. After that, output tuples from v_3, v_4, v_5 are delivered to their destination v_6. After an application loop has been completed, the actuator is activated if necessary. For example, a bell might begin to play sounds.

Assume that the conditions mentioned below are satisfied:

Condition 1. Application made up of task v_1, task v_2 …task v_n, the broker knows the sequencing of tasks, e.g. task v_1 first, task v_2 second.

Condition 2. When there are several different tasks running, if new tasks need to be started at anytime, the broker will check available processors and allocate the most appropriate processors to the tasks to result in the shortest completion time.

Condition 3. Some tasks of the task graph are already scheduled, we may reschedule them upon the arrival new task.

Condition 4. A task can start its execution after all of its parent tasks have already been executed. Each task appears only once in the schedule.

Condition 5. The ready time $t_{ready}(v_i, P_l)$ is the time that processor P_l completes its last running assigned task and be ready to execute task v_i. Therefore,

$$t_{ready}(v_i, P_l) = \max\{t_f(v_y, P_l), \max_{e_{zi} \in E, v_z \in prec(v_i), k \in N} (t_f(v_z, P_k) + c(e_i^{kl}))\} \quad (1)$$

where v_y is a task being executed at processor P_l and $prec(v_i)$ is a set of preceding tasks of v_i, and $c\left(e_i^{kl}\right)$ is the communication time of connection between processors P_k and P_l. It can be defined as:

$$c(e_i^{kl}) = \begin{cases} (do_{zi})\frac{1}{bw_{kl}} & \text{if } k \neq j \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Here do_{zi} is volume of outgoing data transferred from P_k to P_l.

Condition 6. Let $[t_A,t_B] \in [0,\infty]$ be an idle time interval on processor P_l in which no task is executed. A free task $v_i \in V$ can be scheduled on processor P_l within $[t_A,t_B]$ if

$$\max\{t_A, t_{ready}(v_i, P_l)\} + w(v_i, P_l) \leq t_B. \quad (3)$$

4.2 Task Scheduling Solution

Given a task graph $G = (V, E, W, C)$ and a processor graph $PG = (N,D)$, our approach is to produce the most appropriate scheduled list of tasks. Because our method tries to minimize the completion time while considering the network contention and cloud cost, the following section exemplifies the formation of Earliest Finish Time (*EFT*) and the cost of task v_i on a processor from its Earliest Start Time (*EST*) as well as the element costs.

EST of a task v_i executed on a processor P_l can be calculated as follows:

$$EST(v_i, P_l) = \begin{cases} t_{ready}(v_i, P_l), & if\ v_i \neq v_{entry} \\ 0, & otherwise \end{cases}, \quad (4)$$

Suppose that p_{il} is the probability of task v_i processed at processor P_l, then the overall task arrival rate for each arrival rate λ_{il} at processor P_l can be calculated as:

$$\sum p_{il}\lambda_{il}$$

Thus, given service rate Ω_l at processor P_l, the computation time to execute task v_i on processor P_l is defined as:

$$w(v_i, P_l) = |\frac{l(v_i)}{\Omega_l - \sum p_{il}\lambda_{il}}| \quad (5)$$

Consequently, *EFT* of the task v_i is designed as:

$$EFT(v_i, P_l) = w(v_i, P_l) + EST(v_i, P_l), \quad (6)$$

Additionally, the algorithm also takes into account the cost for using cloud resources or fog resources to finish tasks. Given $x_{v_i}^{f_l} \in \{0, 1\}$, the cost $C(v_i,P_l)$ to execute task v_i at VM P_l or local fog device P_l is defined by:

$$C(v_i, P_l) = (C_{comm}^{(v_i,P_l)} + C_{disc}^{(v_i,P_l)}) * x_{v_i}^{f_l} + C_{proc}^{(v_i,P_l)} + C_{queue}^{(v_i,P_l)} + C_{str}^{(v_i,P_l)} + C_{mem}^{(v_i,P_l)}, \quad (7)$$

In Eq. (7), each cost is calculated as follows:

Processing cost is expressed as:

$$C_{proc}^{(v_i,P_l)} = c_1 * w(v_i, P_l), \quad (8)$$

where c_1 is the processing cost per time unit of workflow execution on processor P_l.

Let t_{assign} be the time point when task v_i is assigned to processor P_l. c_2 be the queuing cost per time unit of the task v_i until it can be executed. Then the queuing cost is as:

$$C_{queue}^{(v_i,P_l)} = c_2 * (EST(v_i,P_l) - t_{assign}(v_i,P_l)) \quad (9)$$

Assume that the amount of money per time unit for transferring outgoing data from processor P_k to P_l is c_3, then the communication cost is defined as follows:

$$C_{comm}^{(v_i,P_l)} = c_3 * (\sum_{v_z \in (prec(v_i) \cap exec(k))} do_{zi}) \frac{1}{bw_{kl}} \quad (10)$$

We presume that the distribution of disconnection events between a cloud or a fog device and clients is a Poisson distribution with the parameter μ_T, which represents the stability of the network. The expected number of arrivals over an interval of length τ is $E[N_T] = \mu_T * \tau$. Let L be a random variable for the length of an offline event, μ_L be the mean length and c_4 be the disconnection cost per unit time. Therefore, the expected duration of a disconnection event, which can affect the completion time of task v_i, is $\mu_T * \tau * \mu_L$. Hence, the cost of disconnection can be derived as:

$$C_{disc}^{(v_i,P_l)} = c_4 * (\mu_T * \tau * \mu_L), \quad (11)$$

Let c_5 be the storage cost per data unit and st_i be the storage size of task v_i on processor P_l. Then we can present the storage cost of task v_i on processor P_l as:

$$C_{str}^{(v_i,P_l)} = c_5 * st_i, \quad (12)$$

In addition, the memory cost of processor P_l for task v_i is computed as follows:

$$C_{mem}^{(v_i,P_l)} = c_6 * s_{mem}, \quad (13)$$

where s_{mem} is the size of the memory used and c_6 is the memory cost per data unit.

Lastly, an optimal function that calculates the tradeoff U can be formulated as a convex combination of EFT and monetary cost for each processor $P_l \in N$ in fog or cloud where task $v_i \in E$ running as follows:

$$U = Min \sum_{v_i,v_j \in V, P_k \in N} (\partial \frac{EFT(v_i,P_l) - Min[EFT(v_j,P_k)]}{Max[EFT(v_j,P_k)] - Min[EFT(v_j,P_k)]} + (1-\partial) \frac{C(v_i,P_l) - Min[C(v_j,P_k)]}{Max[C(v_j,P_k)] - Min[C(v_j,P_k)]}), \quad (14)$$

where $\partial \in [0, 1]$ is a cost-conscious factor that represents a user's preference for the completion time and the monetary cost.

By considering the above function that combines $cost(v_i,P_l)$ and $EFT(v_i,P_l)$, we can determine the most appropriate schedule. This indicates that its combination of $cost(v_i,P_l)$ and $EFT(v_i,P_l)$ should demonstrate the minimum value of the tradeoff U.

Due to evaluate the performance of the proposal, we have compared our approach with state-of-the-art task scheduling algorithms. These minimize the total completion time of the workflow or reduce the cloud's task processing cost. Algorithm 1 is Greedy approach to cost reduction, which allocates tasks to processors following greedy principle that minimizes cloud cost. In algorithm 2, network contention-aware task scheduling [21] is presented. In the meantime, algorithm 3 shows that our approach, a dynamic Cost-Time aware method, stands on both network contention and cloud cost. Therefore, it can justify the tradeoff of cloud cost and completion time.

Algorithm 1. Greedy approach for cost reduction

Input : Task graph G, processor graph PG
Output : Scheduled list of tasks
Function greedyForCostScheduling(G, PG)
1. Sort task v_n in the descending order by its priority;
2. **for** each $v_n \in V$ **do**
3. Find the most appropriate processor P_l that minimizes the cost of accomplishing the task v_n;
4. Assign v_n to P_l;
5. **end**
6. **return** scheduled list of tasks;

Algorithm 2. Contention aware task scheduling

Input : Task graph G, processor graph PG
Output : Scheduled list of tasks
Function networkCostScheduling (G, PG)
1. Sort task v_n in the descending order by its priority;
2. **for** each $v_n \in V$ **do**
3. Find the most appropriate processor P_l that allows EFT of v_n, considering network bandwidth usage;
4. Assign v_n to P_l;
5. **end**
6. **return** scheduled list of tasks;

Algorithm 3. Dynamic Cost-Time aware method

Input : Task graph G, processor graph PG
Output : Scheduled list of tasks
Function DCTScheduling (G, PG)
1. Sort task v_n in the descending order by its priority;
2. **for** each $v_n \in V$ **do**
3. Find the most appropriate processor P_l that allows EFT of v_n, considering network bandwidth usage and monetary cost;
4. Assign v_n to P_l;
5. **end**
6. **return** scheduled list of tasks;

5 Implementation and Analysis

5.1 Experimental Settings

Several different experiments have been performed to evaluate the efficiency of our approach with varying conditions such as communication costs and upcoming tasks, network usage as well as a varying number of application modules. To check the performance, we compare the results obtained from the proposed dynamic Cost-Time aware Scheduling (DCTS) and the existing ones: Contention aware Task Scheduling (CaTS) [19] just taking account of network contention, Greedy approach for Cost Reduction (RCR) concerning the monetary cost. The experimental setup configurations are provided in Tables 1, 2, 3 and 4.

Task graphs are created with number of tasks in the graph ranging from 10 to 120. The task graphs are scheduled on a multiprocessor system a combination of 22 heterogeneous VMs and 8 fog devices located at the local system of CCs for the above algorithms. The metrics used for comparison are the completion time and monetary cost. We assume the communication network is fully connected. Each communication link has its own randomly generated cost. And simulation settings are developed in Java with JDK-7u7-i586 and Netbeans-7.2. The proposed algorithm is evaluated using the iFogSim simulator. This JAVA-based simulator is an extension of CloudSim [6] designed to simulate fog computing environments [2]. CloudSim is a framework for modeling and simulation of cloud computing infrastructures and services. In our simulation, we use MI (Millions of Instructions) and MIPS (Million Instructions per Second) to represent the processing capacity of processors.

5.2 Experimental Results

In the following figures, we present the experimental results to demonstrate that our method, DCTS, can be more efficient in terms of schedule length and cloud cost. Figures 4 and 5 show that although GCR obtains the worst result regarding the completion time, it provides the highest cost saving for CCs. In contrast, CaTS delivers the best performance but maximum cost. In the meantime, our solution conduces to the benefits

Table 1. Features of the simulated system

Parameter	Value
Topology model	LAN, fully connected
Operating system	Windows 10 Professional
Number of processors	30
Number of tasks	[10, 110]
Processing rate	[15, 800]
Bandwidth	[20, 100, 512, 1024] Mbps
Cost per time unit executed on processor P_1	[0.2, 0.6]
Cost per outgoing data unit from processor P_1	[0.1, 0.5]
Cost of waiting time	[0.2, 0.5]
Cost of a disconnection time	[0,03, 0.3]

Table 2. Latency between devices.

Source	Destination	Latency (ms)
Sensor	End device	0.6
Actuator	End device	0.1
End device	Fog device	2.0
Fog device	Proxy-server	4.0
Proxy-server	Cloud datacenter	100.0

Table 3. Configuration of devices.

Device type	CPU (MIPS)	RAM (MB)	Uplink bandwidth (Mbps)	Downlink bandwidth (Mbps)
Cloud Data Center	44,800	40,000	100	10,000
Proxy-server	2800	4000	10,000	10,000
Fog device	2800	4000	10,000	10,000
End device	1000	1000	10,000	270
Cloud Data Center	44,800	40,000	100	10,000

of balance between acceptable completion time for workflow and the corresponding cost for utilizing cloud resources. In particular, compared with CaTS, our method can save nearly 19% cost for CCs and it is 21% faster than GCR.

The evaluation of the effect of varying numbers of processors on the cloud cost and the schedule length only in DCTS with a stable number of tasks is also made and given in

Table 4. Description of application tuples.

Tuple type	CPU length	Network length
TT_A	44,800	40,000
TT_B	2800	4000
TT_C	2800	4000
TT_D	1000	1000
TT_E	1400	5000
TT_F	1000	1000
TT_G	1000	500
TT_H	1000	500
TT_I	1000	4000

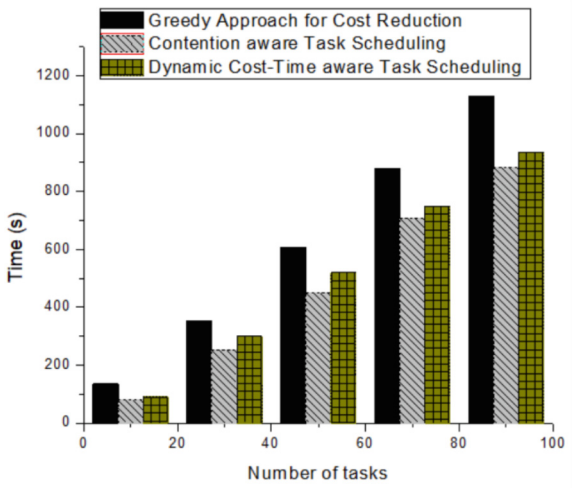


Fig. 4. Schedule length comparison

Figs. 6 and 7, respectively. It is clear to see that there is a great improvement in the speed obtained through DCTS. This improvement rises as the number of processors increases. However, the cost is higher.

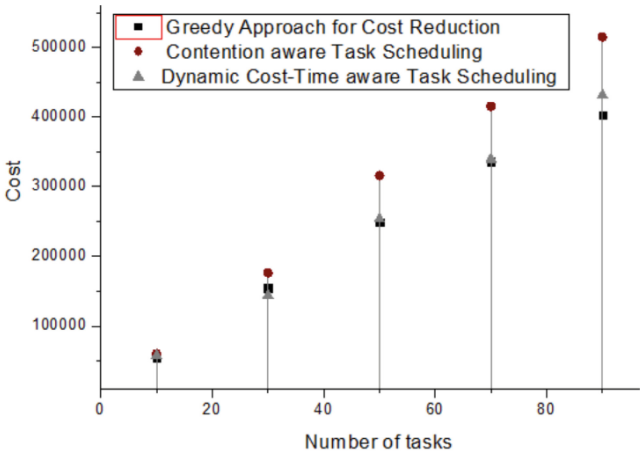


Fig. 5. Cost comparison

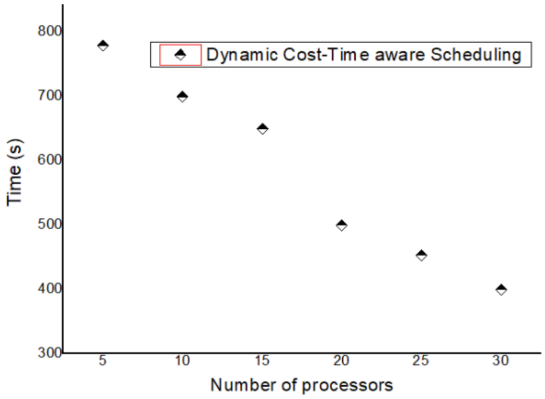


Fig. 6. Schedule length with numbers of processors

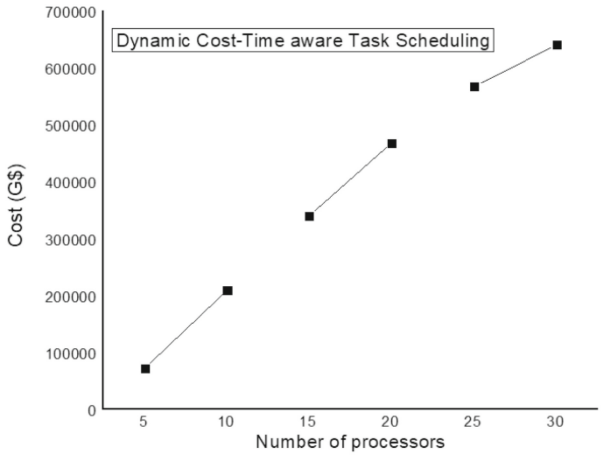


Fig. 7. Cost with numbers of processors

6 Conclusion

This paper proposes a collaboration between fog and cloud resources within a cloud platform to leverage the combined computing power of both internal and external infrastructure. Additionally, we introduce a novel method to extend dynamic task scheduling, aiming to achieve desired completion times while balancing system performance and cloud service costs. Furthermore, we conduct simulations to assess our approach and compare it with alternative methods. The experimental findings demonstrate that the proposed scheduling approach yields improved performance while requiring less monetary expenditure. In future work, we plan to refine our scheduling method across various scenarios, including considerations of energy consumption, to enhance trustworthiness and effectiveness with optimal efficiency.

References

1. Raafat, O., Mazin, A., Taha, L., Khaled, E.: Scheduling Internet of Things requests to minimize latency in hybrid Fog–Cloud computing. Future Gen. Comput. Syst. (2020)
2. Gupta, H., Dastjerdi, A.V., Ghosh, S., Buyya, R.: iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Softw. Pract. Exp. **47**(9), 1275–1296 (2017)
3. Entezari-Maleki, R., Sarrafzade, N., Sousa, L.: A genetic-based approach for service placement in fog computing. J. Supercomput. (2022)
4. Liu, Z., Yang, X., Yang, Y., Wang, K., Mao, G.: DATS: dispersive stable task scheduling in heterogeneous fog networks. IEEE Internet Things J. **6**(2), 3423–3436 (2018)
5. Yang, Y., Zhao, S., Zhang, W., Chen, Y., Luo, X., Wang, J.: DEBTS: delay energy balanced task scheduling in homogeneous fog networks. IEEE Internet Things J. **5**(3), 2094–2106 (2018)
6. Cloudsim. https://code.google.com/p/cloudsim/downloads/list (2014)
7. Ehsan, M., Saima, I.: Novel approaches for scheduling task graphs in heterogeneous distributed computing environment. Int. Arab J. e-Technol. **12**(3) (2014)

8. Fatma, A.O., Mona, M.A.: Genetic algorithms for task scheduling problem. J. Parallel Distrib. Comput. **70**(1), 13–22 (2010)
9. Gopalakrishnan, A.K.: A subjective job scheduler based on a backpropagation neural network. HCIS **3**(1), 1–15 (2013). https://doi.org/10.1186/2192-1962-3-17
10. Hung, P.P., Bui, T.A., Eui-Nam, H.: Optimal collaboration of thin–thick clients and resource allocation in cloud computing. Pers. Ubiquit. Comput. **18**(3), 563–572 (2014)
11. Huynh, T.T.B.: Multi-objective genetic algorithm for solving the multilayer survivable optical network design problem. J. Converg. **5**(1), 20–25 (2014)
12. Amazon Web Services. http://aws.amazon.com/ec2/
13. Joel, W., et al.: SODA: an optimizing scheduler for large-scale stream-based distributed computer systems. In: International Conference on Middleware, pp. 306–325 (2008)
14. Juan, L., Sen, S., Xiang, C., Zhongbao, Z.: Cost-conscious scheduling for large graph processing in the cloud. In: IEEE International Conference on High Performance Computing and Communications, pp. 808–813 (2011)
15. Pham, H., Eui-Nam, H.: An adaptive procedure for task scheduling optimization in mobile cloud computing. Math. Prob. Eng. **2015**, 1–13 (2015)
16. Lingfang, Z., Veeravalli, B., Xiaorong, L.: ScaleStar: budget conscious scheduling precedence-constrained many-task workflow applications in cloud. In: Advanced Information Networking and Applications (AINA), pp. 534–541 (2012)
17. Hung, P.P., Bui, T.-A., Morales, M.A.G., Van Nguyen, M., Huh, E.-N.: Optimal collaboration of thin–thick clients and resource allocation in cloud computing. Pers. Ubiquit. Comput. **18**(3), 563–572 (2013)
18. Byungsang, K., Chan-Hyun, Y., Yong-Sung, P., Yonggyu, L., Wan, C.: An adaptive workflow scheduling scheme based on an estimated data processing rate for next generation sequencing in cloud computing. J. Inf. Process. Syst. **8**(4), 555–566 (2012)
19. Mohammad, A., Salama, H., Sulieman, B.: On static scheduling of tasks in real time multiprocessor systems: an improved GA-based approach. Int. Arab J. e-Technol. **11**(6) (2013)
20. Nguyen, D.M., Eui-Nam, H.: Cost and efficiency-based scheduling on a general framework combining between cloud computing and local thick clients. In: Computing, Management and Telecommunications, pp. 258–263 (2013)
21. Oliver, S.: Communication contention in task scheduling. IEEE Trans. Parallel Distrib. Syst. **16**(6), 503–515 (2005)
22. Chandrashekhar, S., Rajnikant, B.: Priority based dynamic resource allocation in cloud computing. In: International Symposium on Cloud and Services Computing, Mangalore (2012)
23. Siva, T.M., Srikant, R.: Stochastic models of load balancing and scheduling in cloud computing clusters. In: IEEE Infocom, pp. 702–710 (2012)
24. Ruben, V., Vanmechelen, K., Broeckhove, J.: Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds. In: Cloud Computing Technology and Science (CloudCom), pp. 320–327 (2011)
25. Sen, S., Jian, L., Qingjia, H.: Cost-efficient task scheduling for executing large programs in the cloud. Parallel Comput. J. (2013)