Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.039276
Article

Auto-Scaling Framework for Enhancing the Quality of Service in the Mobile Cloud Environments

Yogesh Kumar[1,*], Jitender Kumar[1] and Poonam Sheoran[2]

[1]CSE Department, DCR University of Science and Technology, Murthal, Sonepat, 131039, India
[2]BME Department, DCR University of Science and Technology, Murthal, Sonepat, 131039, India
*Corresponding Author: Yogesh Kumar. Email: sangwan130@gmail.com
Received: 19 January 2023; Accepted: 16 March 2023

Abstract: On-demand availability and resource elasticity features of Cloud computing have attracted the focus of various research domains. Mobile cloud computing is one of these domains where complex computation tasks are offloaded to the cloud resources to augment mobile devices' cognitive capacity. However, the flexible provisioning of cloud resources is hindered by uncertain offloading workloads and significant setup time of cloud virtual machines (VMs). Furthermore, any delays at the cloud end would further aggravate the miseries of real-time tasks. To resolve these issues, this paper proposes an auto-scaling framework (ACF) that strives to maintain the quality of service (QoS) for the end users as per the service level agreement (SLA) negotiated assurance level for service availability. In addition, it also provides an innovative solution for dealing with the VM startup overheads without truncating the running tasks. Unlike the waiting cost and service cost tradeoff-based systems or threshold-rule-based systems, it does not require strict tuning in the waiting costs or in the threshold rules for enhancing the QoS. We explored the design space of the ACF system with the CloudSim simulator. The extensive sets of experiments demonstrate the effectiveness of the ACF system in terms of good reduction in energy dissipation at the mobile devices and improvement in the QoS. At the same time, the proposed ACF system also reduces the monetary costs of the service providers.

Keywords: Auto-scaling; computation offloading; mobile cloud computing; quality of service; service level agreement

1 Introduction

Due to ubiquitous availability, mobile devices have become a universal interface for daily activities. However, it is still challenging to perform complex computation tasks on mobile devices due to the limited computation and battery resources [1]. Mobile cloud computing (MCC) has been envisioned as the potential solution for alleviating the resource limitations of mobile devices [2]. The key premise of MCC resources is that these resources can be acquired and released as per requirement, i.e., these

resources are available in an on-demand fashion [3]. In addition, the elasticity feature (i.e., auto-scaling) of cloud computing enables the cloud service providers (CSPs) to increase and decrease the resources as per the arriving offloading workloads [3]. Consequently, it aids in the optimum utilization of precious computing resources. However, auto-scaling brings new challenges for CSPs because offloading demands are dynamic and infrequent [4]. In addition, cloud resources take significant setup time, e.g., 150 s [5]. Therefore, the efficacy of auto-scaling solutions can influence the service level agreement (SLA) and quality of service (QoS). Consequently, the performance of the offloaded tasks would also degrade because offloading tasks need a quick response [6].

Understanding the implications of resource management becomes more critical when applications involve humans with uncertain actions as client-side users. Therefore, strict workload estimation mechanisms like time series forecasting [7] or machine learning [8] can suffer due to the significant setup time of new cloud virtual machines (VMs). Moreover, computation offloading involves data transmission over unreliable wireless networks that are generally controlled by third-party operators, e.g., Vodafone, AT&T, etc. Under such circumstances, any delays at the CSP end would further aggravate the miseries of the real-time tasks. However, the state-of-the-art MCC auto-scaling solutions rely on response time [4,8] or threshold rules [9,10], or the tradeoff between waiting costs and leasing costs [6]. Since MCC involves data transmission over unreliable wireless networks, CSPs cannot be sued for the violation in response time due to wireless network delays. At the same time, threshold-rule-based systems require application-specific manual tunings [11]. Similarly, the success of the tradeoff between waiting costs and service costs-based systems relies on the strictness of waiting costs. Moreover, such systems also require negotiation of waiting time, after which penalties would be imposed on the CSPs. Setting optimum waiting time is a tedious task as some smaller offloaded tasks can save energy if not delayed. Moreover, the optimum task waiting time can still elapse because of the data transmission over unreliable wireless networks.

To resolve these issues, this paper proposes an auto-scaling framework (ACF) that contributes to the following:

- An auto-scaling solution that strives to minimize the delays at the CSP end as per SLA negotiated QoS assurance level for the availability of service.
- An innovative solution for overcoming the VM startup overheads without truncation of running tasks.
- A series of experiments are conducted to compare the performance of the proposed ACF with different auto-scaling solutions.

We evaluated the proposed ACF system with CloudSim Simulator [12]. The experimental results reflect the efficacy of the ACF system over the tradeoff between waiting costs and leasing costs-based systems, threshold-rule-based systems, and response-time-based systems in terms of better energy savings at significantly lower monetary prices. In addition, it also minimizes delayed tasks and SLA violations.

The rest of the paper is organized as follows: Section 2 overviews the related research. Section 3 discusses the background of computation offloading. Section 4 presents the problem statement. Section 5 presents the design details of the proposed system. Section 6 elaborates on the experimental setup and results. Finally, Section 7 concludes the paper.

2 Related Work

Resource management decisions are a response to the performance degradation of the offloaded tasks at the CSP end. Since computation offloading from mobile devices involves unpredictable wireless networks, the research community has proposed a wide variety of mechanisms to tackle this issue. Some argue for using resources from proximate cloud platforms like edge computing [13], cloudlets [14], ad-hoc virtual clouds [15], or fog computing [16] for minimizing the long-term wide area network latencies of cloud computing. However, the proximate clouds generally have limited computing resources [17]. Therefore, these systems also need to lease computational resources from distant clouds, i.e., auto-scaling is still required in the proximate clouds to reduce leasing costs and deal with infrequent offloading tasks. Similarly, with remote cloud resources, auto-scaling can minimize the CSP's costs under infrequent offloading tasks [7]. Hence, auto resource management as per the arriving workloads is imperative irrespective of the remote platforms used for offloading the computation from mobile devices. Furthermore, the significance of queuing delays at the CSP end has also been highlighted by [6]. Hence this section reviews the contemporary research that focuses on minimizing the delays at the CSP end in proximate clouds and distant clouds.

In proximate clouds, Reference [18] employs an admission control for the overwhelming offloading tasks. Chamola et al. [19] advocate balancing the workload between lightly and heavily loaded edge clouds. However, it would require frequent redistribution of running tasks. Meanwhile, Ma et al. [17] employ collaboration between edge clouds and distant clouds for dealing with the overwhelming offloading tasks. In addition, this framework proposes an optimization framework for scheduling the tasks between edge and leased cloud resources assuming that the arriving workload is known in advance. Hence some sort of resource management is imperative in proximate clouds. The key premise of ACF is that it can be employed at the proximate clouds for timely procurement of computational resources from the distant clouds as well as within proximate clouds.

The distant cloud auto-scaling systems can be classified into three broad categories: threshold-rule-based systems [5,6,20 – 23], response-time-based systems [4,7,8], and a tradeoff between different parameters [5,6,24]. Threshold-rule-based systems fix upper utilization (*UpTh*) and lower utilization (*LowTh*) for increasing or decreasing the number of running VM instances. The framework discussed in [9] used *UpTh* as 70% and *LowTh* as 30%. Similarly, [10] used *UpTh* as 75% and *LowTh* as 50%. The study discussed in [20] used *UpTh* as 80% and *LowTh* as 20% in M/M/s queuing systems with a focus on optimum utilization of the leasing interval of VMs. Meanwhile, [21] used an M/M/s/k queuing system with *LowTh* at 80%. Some different variants of threshold-rule-based systems are proposed in [22,23]. In [22], auto-scaling decisions are carried out after a pre-defined number of tasks, e.g., after 150 tasks. After executing the pre-defined number of tasks, the auto-scaling decisions are again carried out using threshold rules. Meanwhile, [23] used upper *UpTh* as 75%. Unlike other threshold-rule-based systems, it employs classification of the workload and increases/decreases the number of running VMs when significant changes in the workload are observed. However, such systems can suffer from sudden workload changes. So, threshold-rule-based systems are application specific and require a lot of manual tunings for adapting to the QoS requirements [11].

Response-time-based systems need pre-negotiation on the maximum execution time at the CSP end. System [4] exploits user-specified demands on response time for increasing or decreasing the number of running VM instances. Moreno-Vozmediano et al. [8] advocate machine learning mechanisms for workload prediction and employ the M/M/s queuing system for maintaining the VMs as per the negotiated response time. Similarly, Singh et al. [7] combined threshold rules and queuing theory to maintain the auto-scaling system's required response time. However, in the MCC environment, data

transmission occurs over unreliable wireless networks, so CSPs cannot be sued for the violation in response time due to network delays. Moreover, there would be delays at the CSP end due to the granted relaxations in the response time.

Ferber et al. [5] employed a tradeoff between service costs and insufficient VMs and decided to keep 10% extra VMs. Similarly, Mei et al. [24] used a tradeoff between server speed and server size for maximizing the profit according to the required service level for the SLA negotiated waiting time. Meanwhile, the work discussed in [6] exploits a tradeoff between service and waiting costs for maintaining the VMs as per arriving workloads. However, tradeoff-based systems also require strict tunings in waiting times and waiting costs. Hence, queuing delays at the CSP end would degrade the performance of the mobile devices. In contrast, the ACF system strives to minimize queuing delays the real-time tasks according to the SLA negotiated service assurance level without manually tuning the waiting costs or threshold rules. Table 1 compares some of the works in the literature considering their auto-scaling techniques for handling unpredictable offloading workloads.

Table 1: Related works on auto-scaling systems

Work	Objective	Technique	Proactive	Optimum utilization of VM leasing interval	Suitable for real-time applications
[4]	Minimize service costs	Response time	No	Yes	No
[5]	Minimize service costs	Tradeoff between service costs and insufficient VMs	No	No	No
[6]	Minimize total costs	Tradeoff between service costs and waiting costs	Yes	Yes	No
[7]	Minimize service costs	Response time and threshold-rule	Yes	Yes	No
[8]	Minimize service costs	Response time	Yes	No	No
[9]	Maximize VMs utilization	Threshold-rule	No	Yes	No
[10]	Minimize service costs	Threshold-rule	No	No	No
[20]	Maximize VMs utilization	Threshold-rule	Yes	Yes	No
[21]	Minimize SLA violations	Threshold-rule	Yes	No	No
[22]	Minimize service costs	Threshold-rule	Yes	No	No
[23]	Maximize service availability	Threshold-rule	Yes	No	No

(Continued)

Table 1: Continued

Work	Objective	Technique	Proactive	Optimum utilization of VM leasing interval	Suitable for real-time applications
[24]	Maximize profit	Tradeoff between server speed and server size	No	Yes	No
Our work	Maximize service availability	Minimize nonzero waiting tasks	Yes	Yes	Yes

3 Computation Offloading Background

Several researchers have highlighted that computation offloading is beneficial when the local execution energy/time of a task is higher than the cloud execution energy/time [25]. However, queuing delays at the CSP end can increase the energy dissipation of the mobile devices because mobile devices would dissipate the idle CPU energy (P^m_{idle}) and display energy (P^m_{disp}) during the waiting time at the CSP end. Furthermore, turning on the power saving mode (PSM) would increase the data transmission time [26]. Therefore, some mechanisms are needed to regulate the CSPs to ensure minimal delays for the offloaded tasks. Table 2 presents the list of notations used in this paper.

Table 2: Description of notations

Symbol	Description	Symbol	Description
S^m_F	Processing speed of mobile device	T^q_i	Waiting time suffered by the i^{th} the task
S^{VM}_F	Processing speed of virtual machine (VM)	α	Probability of nonzero delay at cloud
P^m_{comp}	Power dissipation by mobile device during computation	$W_q(0)$	Probability of zero delay at cloud
P^m_{comm}	Power dissipation by mobile device during communication	T_q	Overall waiting time in cloud system
P^m_{idle}	Power dissipation by mobile device during idle	S	Number of running VMs
P^m_{disp}	Power dissipation by mobile device display	λ	Task arrival rate
P^m_w	Power dissipation by mobile device during waiting	μ	Task service rate
T^{Tr}_i	Transmission time of the i^{th} the task	P_0	Probability of empty cloud system
T^{VM}_i	Execution time of the i^{th} the task on VM	β	Smoothing factor
T^q_i	Waiting time the of i^{th} the task on VM	ψ	Leasing rate of VMs
T^L_i	Execution time of the i^{th} the task on mobile device	A	Penalty rate on delayed tasks
T^R_i	Overall remote execution time of the i^{th} the task	K	Number of tasks in the cloud system

(Continued)

Table 2: Continued

Symbol	Description	Symbol	Description
E_i^L	Overall remote execution energy of the i^{th} the task	D	SLA negotiated waiting time
E_i^L	Overall local (mobile device) execution energy of the i^{th} the task	Wl_i^{VM}	Number of instruction in the i^{th} the task

We have considered a sample computation offloading model to show the effects of queuing delays on the performance of offloaded tasks. Let the mobile device be denoted by a 5-tuple $M = \{S_F^m, P_{comp}^m, P_{comm}^m, P_{idle}^m, P_{disp}^m\}$ where S_F^m is the processing capacity of a mobile device (in a million instructions per second), P_{comp}^m is the power consumption by the mobile device during local execution, and P_{comm}^m is the power consumption by the mobile device during communication. Assume that T_i^R and T_i^L are the cloud computation times and local computation times for the i^{th} the task, respectively, then the computation can be offloaded when cloud energy dissipation for the i^{th} the task of a mobile Job (E_i^R) is lesser than the local energy dissipation (E_i^L), i.e.:

$$E_i^R < E_i^L \tag{1}$$

Local execution energy E_i^L can be estimated as:

$$E_i^L = T_i^L \times \left(P_{comp}^m + P_{disp}^m\right) \tag{2}$$

If T_i^{Tr}, T_i^{VM}, and T_i^q are the transmission time of the input/output data over the wireless network, execution time on cloud VMs, and waiting time for getting the VMs, respectively, then E_i^R can be estimated as:

$$E_i^R = T_i^{Tr}\left(P_{comm}^m + P_{idle}^m + P_{disp}^m\right) + T_i^{VM}\left(P_{idle}^m + P_{disp}^m\right) + T_i^q\left(P_{idle}^m + P_{disp}^m\right) \tag{3}$$

4 Problem Statement

Let an MCC service provider be modeled as an M/M/s queuing system. It tenants identical VMs from infrastructure as a service (IaaS) provider. Let λ be the task arrival rate, μ be the task service rate, and α be SLA negotiated QoS assurance level for nonzero delay at the CSP end. Incoming tasks are served on a first-come, first-serve order. Tasks not served are buffered and delayed with no assumption on the limit of backlogged tasks buffer size. Since queuing delays at the CSP end would increase the energy dissipation at the mobile devices, therefore the fundamental objective of the ACF system is to maintain VMs according to the SLA negotiated QoS assurance level for the nonzero delay, i.e.:

$$1 - W_q(0) \leq \alpha \tag{4}$$

where $W_q(0)$ is the probability of zero wait time at the CSP end. To find $W_q(0)$, let T_q represents the waiting time in the queue in the steady state. Then,

$$W_q(0) = \Pr\left\{T_q = 0\right\} = \Pr\{< s - 1 \text{ in system}\} \tag{5}$$

$$= \sum_{n=0}^{s-1} p_n = P_0 \sum_{n=0}^{s-1} \frac{\lambda^n}{\mu^n \angle n} \tag{6}$$

where p_n is the probability of n number of tasks in the system, i.e.:

$$P_n = \begin{cases} \frac{\lambda^n}{\mu^n \angle n} P_0 & (0 \leq n < s) \\ \frac{\lambda^n}{s^{n-s}\mu^n \angle s} P_0 & (n \geq s) \end{cases} \tag{7}$$

Here P_0 is the probability that the system is idle. Since the sum of probabilities is 1, therefore P_0 can be determined as:

$$P_0 = \left(\sum_{n=0}^{s-1} \frac{\lambda^n}{\mu^n \angle n} + \sum_{n=s}^{\infty} \frac{\lambda^n}{s^{n-s}\mu^n \angle s} \right)^{-1} \tag{8}$$

Now the sum of infinite series in Eq. (8) would be:

$$\sum_{n=s}^{\infty} \frac{\lambda^n}{s^{n-s}\mu^n \angle s} = \frac{\lambda^s}{\mu^s \angle s} \frac{1}{(1-\rho)} \tag{9}$$

where, $\rho = \lambda / s\mu$ is the traffic intensity. So, the probability of an idle system would become:

$$P_0 = \left(\sum_{n=0}^{s-1} \frac{\lambda^n}{\mu^n \angle n} + \frac{\lambda^s}{\mu^s \angle s\,(1-\rho)} \right)^{-1} \tag{10}$$

After rearranging the above equation, we get:

$$\sum_{n=0}^{s-1} \frac{\lambda^n}{\mu^n \angle n} = \frac{1}{P_0} - \frac{\lambda^s}{\mu^s \angle s\,(1-\rho)} \tag{11}$$

This gives

$$W_q(0) = P_0 \left(\frac{1}{P_0} - \frac{\lambda^s}{\mu^s \angle s\,(1-\rho)} \right) = 1 - \frac{\lambda^s P_o}{\mu^s \angle s\,(1-\rho)} \tag{12}$$

5 Design Details

This section details the proposed auto-scaling mechanism by following the MAPE-K (i.e., Monitor, Analyze, Plan and Execute over a shared knowledge base) architecture [27]. All these entities perform their Job during each clock interval and form a close loop together, as depicted in Fig. 1. One key feature of this architecture is that it provides a systematic way of organizing the different entities of the cloud system.

5.1 Monitor

In this phase, ACF continuously monitors the low-level parameters like the number of tasks arrival, the execution time of tasks, and the number of delayed tasks. This information is shared with the knowledge base for analysis and planning phases.

5.2 Analysis

In this phase, ACF estimates the average service rate (μ) of tasks during a clock interval and the total task arrived during a clock interval (λ). Using this information, it predicts the task arrival rate for the next clock interval (λ_p^{t+1}) based on the predicted λ_p^t and observed arrival rates λ_0^t of offloading tasks

in the previous clock interval using EWMA (Exponential Weighted Mean Moving Average) technique, i.e.:

$$\lambda_p^{t+1} = \beta\lambda_o^t + (1-\beta)\,\lambda_p^t \tag{13}$$

Meanwhile, we have used the mean square error estimation method to update the smoothing factor periodically (β) value.

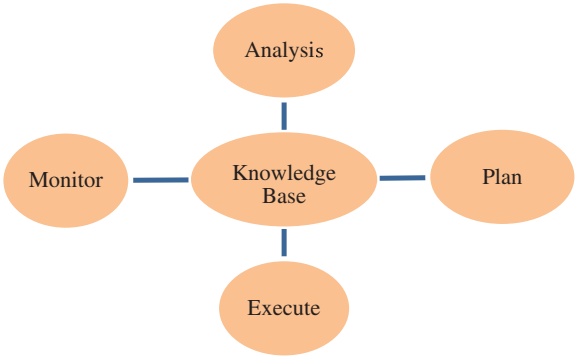

Figure 1: MAPE-K loop [27]

Algorithm 1: Proposed Auto-Scaling Scheme

```
1 procedure Instance_estimator (λ_p^{t+1}, μ, s, α)
2 temp = s; //running VMs
3 compute traffic intensity ρ = λ_p^{t+1}/sμ
4 if (ρ > 1)
5       then increase VMs until (ρ < 1)
6 else s = s/2;
7       if (ρ > 1)
8             then increase VMs until (ρ < 1)
9       end if
10 end else
11 Compute Z = 1 − W_q(0) according to Eq. (4)
12 if (α < Z) then
13      while (α < Z) do
14          s++;
15          Compute Z = 1 − W_q(0) according to Eq. (4)
16      end while
17 end if
18 if (s > temp) then
19      start (s – temp) VMs;
20 end if
21 else
22      annotate (temp – s) VMs as hibernated.
```

(Continued)

Algorithm 1: Continued

23 stop(*temp – s*) VMs when leasing time quantum completing
24 end else
25 end procedure

5.3 Planning

In this phase, the proposed auto-scaling mechanism is used for updating the number of running VMs as per SLA negotiated QoS assurance level (α) for availability. The pseudo-code for this scaling mechanism is depicted in Algorithm 1. Inputs to this algorithm are predicted task arrival rate (λ_p^{t+1}), mean service rate (μ), the number of running VM instances, and required QoS assurance level (α). First, it computes the traffic intensity in line 3. Subsequently, it checks whether the system is stable. If the system is unstable, it increases the VMs count until traffic intensity is less than 1 (lines 4 and 5). If the system is already stable, it decreases the VMs count to half and stabilizes the system again (lines 6–10). The intuition of this alternate phase is to check whether the VMs requirement has decreased for the current clock interval. Next, it computes the QoS assurance level with the stabilized VMs count (line 11). If the QoS requirement is not met with the stabilized system VMs, it increases the VMs count until the required QoS requirements are met (lines 12–17). Subsequently, it checks whether the new VMs requirement is higher than the existing VMs, then places an order to start new VMs (lines 18–20). If the available VMs are more than the required, it first annotates the surplus VMs as hibernated and does not immediately stop them. However, if the leasing time quantum of a hibernated VM is nearing completion, it proceeds to shut down the surplus VMs (lines 21–24).

5.4 Execution

Rather than immediately shutting down the surplus VMs, ACF provides an innovative scheduler for optimizing VMs leasing time quanta. Algorithm 2 depicts the pseudo-code of the scheduler. Unlike [20], the key premise of the proposed scheduler is that it does not remove the running tasks from the surplus VMs. In addition, if a hibernated VM is required in the system, it withdraws the VM with maximum remaining leasing time quanta (lines 5–8). This arrangement aids in the identification of VMs that need to be shut down because only hibernated VMs need to be shut down. Furthermore, it also avoids the continuous profiling of leasing time quanta of the surplus VMs.

Algorithm 2: Scheduler

1 procedure SCHEDULER (Task *t*, *vmslist*)
2 boolean *VMAvailable* = false;
3 Search for a free non-hibernated VM and allocate it
4 set *VMAvailable* = true;
5 else if (*VMAvailable* == false)
6 Draw a hibernated VM with the highest remaining leasing time quanta and allocate it
7 set *VMAvailable* = true;
8 end else if
9 else
10 pendingList.addLast(t);
11 end else
12 end procedure

6 Experimental Setup and Results

We evaluated the performance of ACF using the CloudSim simulator [12]. The experimental scenario is as follows: the MCC service provider leases VMs from infrastructure as a service provider and provides software services to mobile users. The auto-scaling mechanism of ACF operates at the MCC end and ensures the availability of VMs as per the SLA negotiated availability level (α). In other words, ACF takes care of under-provisioning as well as over-provisioning at the MCC service provider end.

6.1 Experimental Setup

In this system, 2.5 GHz dual-core VMs are leased on per-hour time quanta with a $0.145 price, similar to Amazon [28]. Each leased VM serves mobile users on a space-shared policy. In addition, we also fixed the upload data rate (80 Kbps) and download data rate (130 Kbps) because the objective of ACF is to highlight the implications of queuing delays on the performance of mobile devices. Similarly, we have not considered the queuing delays in the computation offloading decisions of tasks at the mobile devices because the queuing delays occur due to the adoption of improper auto-scaling policies at the CSP end. The performance of the ACF system is compared with the following state-of-the-art auto-scaling systems:

- **DBCOF (Demand Based Computation Offloading Framework) [6]:** It is based on the tradeoff between leasing costs (LC) and waiting costs (WC) that minimizes the expected total costs (TC), i.e.,
$$Min\, E\,(TC) = E\,(LC) + E\,(WC) \tag{14}$$
where
$$E\,(LC) = \Psi \times s \tag{15}$$
and
$$E\,(WC) = A \times K \tag{16}$$
The variables ψ, s, A, and K specify the leasing rate of VMs for an interval, the number of running VM instances, the waiting cost of a delayed task, and the total number of tasks in the M/M/s queuing system, respectively. So, if T_q is the waiting time in the system, and D is the SLA negotiated maximum waiting time after which penalties would be imposed on the MCC service provider, then the waiting cost of a delayed task can be computed as per the following:
$$A = \frac{\Psi}{D} + \sum_{i=2}^{y} \Psi \times i \tag{17}$$
where
$$y = \left\lceil \frac{T_q}{D} \right\rceil \tag{18}$$
For fair comparisons, we have also incorporated the waiting cost function of DBCOF in ACF. At the same time, we also incorporated the hibernated VMs solution of ACF in DBCOF.
- **ACF(OP):** It is a static variant of the ACF system that assumes peak offloading workloads are known in advance and does not use any auto-scaling mechanism.
- **Threshold-rule-based system [20]:** It needs rules for the upper utilization threshold and lower utilization threshold for carrying out auto-scaling decisions.

- **Response-time-based system [7]:** It mandates some pre-negotiation on the maximum execution time at the CSP end. For this work, response time is kept as the sum of execution time at the CSP end and SLA negotiated waiting time.

During the evaluations, we used the VM execution time and mobile execution time results of the bubble sort application. These results are abstracted from DBCOF [6] and are depicted in Fig. 2. If S_F^{VM} is the processing speed of cloud VMs, then the execution time of the i^{th} the task can be mapped into the CloudSim task's workload (wl_i^{VM}) by using:

$$wl_i^{VM} = T_i^{VM} \times S_F^{VM} \tag{19}$$

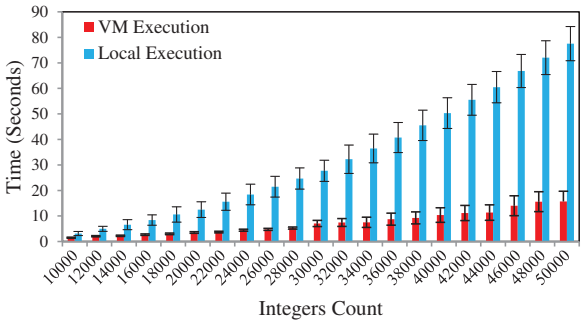

Figure 2: Execution time of bubble sort on mobile device and remote VM [6]

In addition, we kept the configuration of the mobile device as a 1.2 GHz CPU. The power consumption stats of the mobile device are obtained using the PowerTutor tool [29] and listed in Table 3. Jobs arrival follows the Poisson distribution with a mean of 5 Jobs during the busy hours (i.e., from 8:00 AM to 8:00 PM) and 2.5 Jobs during the non-busy hours (i.e., from 8:00 PM to 8:00 AM). Each Job consists of 5 offloadable tasks connected by the linear topology. We used 10 mobile devices during simulations.

Table 3: Parameters of the device profile

Symbol	Estimated value
P^m_{comp}	400 mW
P^m_{disp}	900 mW
P^m_{comm}	750 mW
P^m_{idle}	50 mW

We collected the following metrics during the evaluation:

- **Energy Savings:** It is the ratio of the mobile devices' energy consumption during local execution and remote execution.
- **Monetary costs:** These comprise the leasing costs of VMs along with waiting costs.
- **Utilization rate:** It is the share of the time the VMs were serving tasks during running hours.
- **SLA violation rate:** For a fair comparison with other systems, we have considered the percentage of tasks whose waiting time surpassed the SLA negotiated waiting time (D) as the SLA violation rate. During the experiments, we kept D as 2 s. In addition, we also considered the percentage of tasks that suffered nonzero waiting time as an SLA violation because it is the intended objective

of ACF. The SLA negotiated QoS assurance level for zero waiting (α) is kept at 0.01%, which is near zero.

6.2 Experimental Results

The expected energy savings with ACF, DBCOF, ACF(OP), threshold-rule-based systems, and response-time-based systems are 3.568X, 3.537X, 3.576X, 3.513X, and 3.509, respectively. As the energy savings difference between all systems is marginal, we plotted the ACF and DBCOF in Fig. 3. Meanwhile, the key difference prevails in the monetary costs. The monetary costs incurred by the ACF, DBCOF, ACF(OP), threshold-rule-based systems, and response-time-based systems are $35.66, $37.17, $52.2, $43.05, and $45.12, respectively.

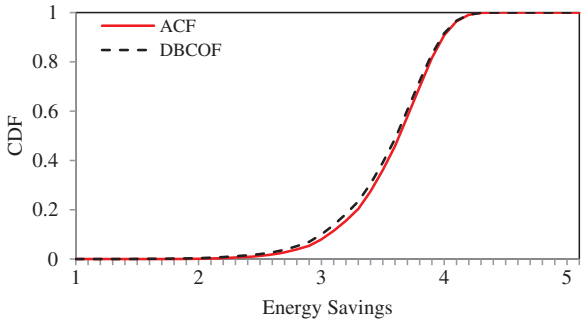

Figure 3: Energy savings by ACF and DBCOF with *D* as 2 s

Fig. 4 shows the results of monetary costs with all the systems. It is the lesser number of delayed tasks in the ACF system, due to which it spends 4.06% lesser monetary costs than DBCOF and saves 0.87% extra energy savings compared to DBCOF. Similarly, the ACF system spends 26.52% less on monetary costs than the response-time-based systems. In addition, the ACF system also achieves 1.65% extra energy savings over response-time-based systems because these systems keep a minimum number of VMs for the SLA-negotiated response time. The energy savings in ACF(OP) is highest because it is based on the assumption that the peak workload arrival rate is known in advance. Meanwhile, threshold-rule-based systems forced us to try different combinations for upper and lower CPU utilization. Finally, we achieved the best results in terms of energy savings for a lower CPU utilization rate of 30% and an upper CPU utilization rate of 50%.

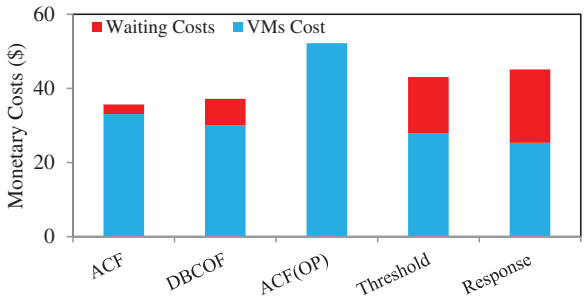

Figure 4: Monetary costs spent by each system with *D* as 2 s

The utilization rate in ACF, DBCOF, ACF(OP), threshold-rule-based systems, and response-time-based systems are 35.6%, 40.8%, 22.1%, 42.4%, and 46.12%, respectively. Meanwhile, SLA violations

in ACF, DBCOF, threshold-rule-based, and response-time-based systems are 0.64%, 1.61%, 2.39%, and 4.21%, respectively. Utilization and SLA violation rates of all systems are depicted in Fig. 5. To demonstrate the efficacy of the proposed system ACF over DBCOF, we have also plotted the SLA violations of both systems over the day in Fig. 6. The SLA violations in ACF are low because it deploys a slightly higher number of VMs. Fig. 7 depicts how ACF manages the acquisition of VMs over the day. At the same time, the percentage of tasks that suffered nonzero delays in ACF, DBCOF, threshold-rule-based, and response-time-based systems are 1.34%, 2.76%, 3.86%, and 6.23%, respectively. Fig. 8 depicts the percentage of tasks that suffered nonzero delays.

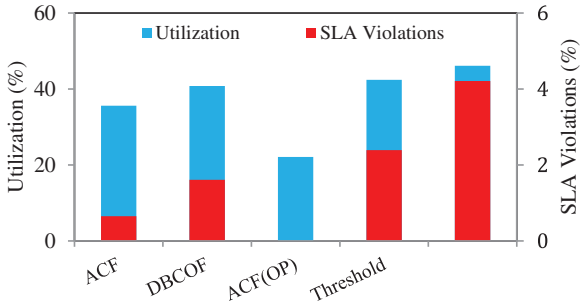

Figure 5: Utilization and SLA violations in each system with *D* as 2 s

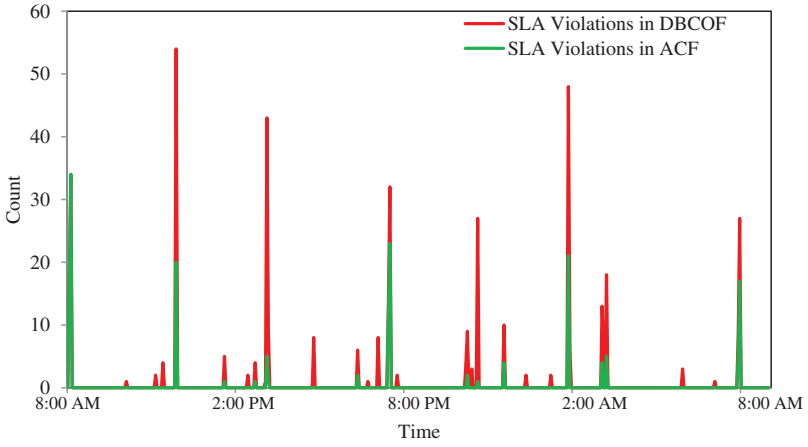

Figure 6: SLA violations over the day with *D* as 2 s

To demonstrate how DBCOF gets impacted by the SLA negotiated waiting times, we decreased *D* from 2 to 1 s. Now the expected energy savings with ACF and DBCOF are 3.568X and 3.567X, respectively. The energy savings in the DBCOF system increased because of the increase in the severity of the waiting costs. At the same time, the monetary cost expenditure in DBCOF rises sharply. The monetary costs spent by ACF and DBCOF systems are $41.83 and $47.79, respectively, i.e., the ACF system spends 12.77% less than the DBCOF system and achieves nearly the same energy savings. Hence the success of the tradeoff between waiting costs and leasing costs-based system relies on the severity of waiting costs for the SLA negotiated waiting time. Fig. 9 depicts the monetary costs of both systems. In addition, the percentage of tasks that suffered nonzero waiting time in the ACF system is still lesser than the DBCOF system. The percentage of tasks that suffered nonzero waiting in the ACF system is 1.34%, and the percentage of tasks that suffered nonzero waiting in the DBCOF system is 2.05%.

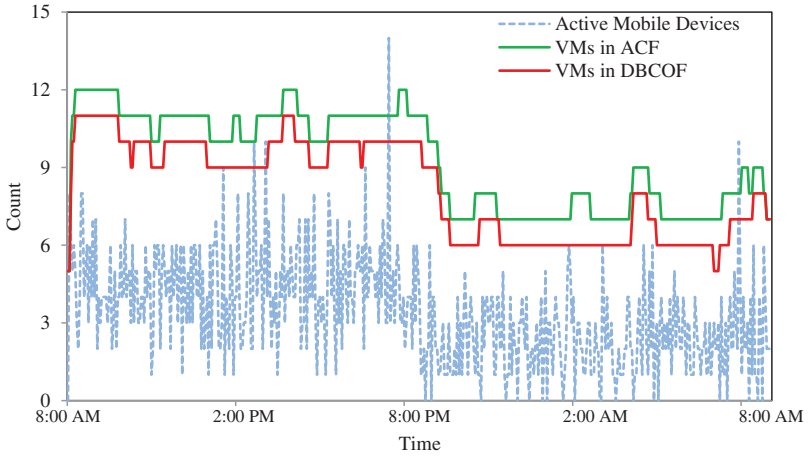

Figure 7: Number of running VMs over the day with D as 2 s

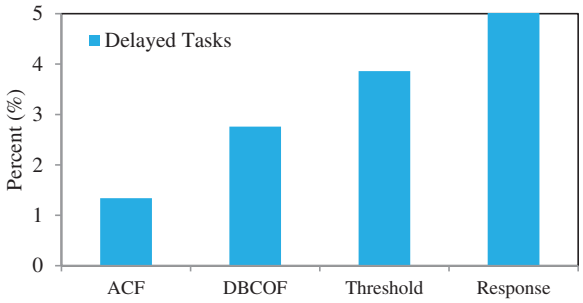

Figure 8: Percentage of tasks that suffered nonzero delays with D as 2 s

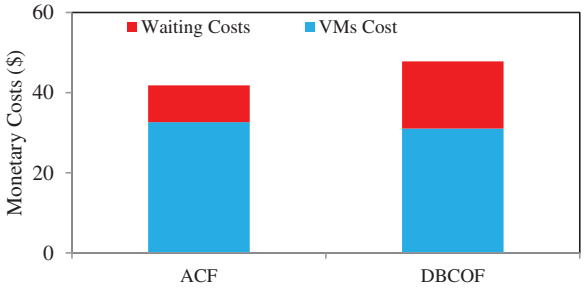

Figure 9: Monetary costs when D is 1 s

7 Conclusions and Future Work

In this work, the problem of real-time execution requirement of mobile tasks in a mobile cloud computing environment is addressed with an auto-scaling solution that strives to minimize the delayed tasks without manual tunings in the waiting costs or threshold rules. This paper contributed to the two key points. First, it provided an auto-scaling solution that maximizes the service availability per SLA negotiated assurance level. Second, it provided an innovative solution for optimizing the leasing time quanta of VMs during the scale-down phases without removing the running tasks from the surplus VMs which are completing their leasing time quanta. Consequently, this arrangement reduced the

monetary costs of CSPs due to reduced SLA violations. In addition, it also increased energy savings on mobile devices due to a lesser number of delayed tasks. The series of experiments vindicate the efficacy of the proposed system.

As part of future work, we plan to extend this framework to consider the provisioning of adaptive extra VMs for dealing with the VM startup overheads. In addition, we plan to incorporate the hybrid leasing plans of VMs, i.e., an optimal combination of reserved and on-demand VMs because reserved are available at discounted rates according to the usage hours.

Acknowledgement: Authors acknowledge the Computer Science and Engineering Department of Deenbandhu Chhotu Ram University of Science and Technology for providing various facilities in the research lab.

Funding Statement: This research work is funded by TEQIP-III under Assistantship Head: 1.3.2.2 in PFMS dated 29.06.2021.

Conflicts of Interest: The authors declare that they have no conflict of interest to report regarding this research work.

References

[1] P. Bahl, R. Y. Han, L. E. Li and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proc. of the 3rd ACM Workshop on Mobile Cloud Computing and Services*, Low Wood Bay Lake District UK, pp. 21–28, 2012.

[2] J. Kumar, A. Rani and S. K. Dhurandher, "Convergence of user and service side perspectives in mobile cloud computing environment: Taxonomy and challenges," *Proceedings of the International Journal of Communication Systems*, vol. 33, no. 18, pp. 1–38, 2020.

[3] M. Armburst, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz *et al.,* "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkley, tech. Rep, UCB/EECS-2009-28, 2009.

[4] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik *et al.,* "COSMOS: Computation offloading as a service for mobile devices," in *Proc. of the 15th ACM Int. Symp. on Mobile Ad-hoc Networking and Computing*, Philadelphia Pennsylvania USA, pp. 287–296, 2014.

[5] M. Ferber, T. Rauber, M. H. C. Torres and T. Holvoet, "Resource allocation for cloud-assisted mobile applications," in *Proc. of the 5th IEEE Conf. on Cloud Computing*, Honolulu, HI, USA, pp. 400–407, 2012.

[6] J. Kumar, A. Malik, S. K. Dhurandher and P. Nicopolitidis, "Demand-based computation offloading framework for mobile devices," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3693–3702, 2017. https://doi.org/10.1109/JSYST.2017.2706178

[7] P. Singh, A. Kaur, P. Gupta, S. S. Gill and K. Jyoti, "RHAS: Robust hybrid auto-scaling for web applications in cloud computing," *Cluster Computing*, vol. 24, pp. 717–737, 2021.

[8] R. Moreno-Vozmediano, R. S. Montaro, E. Huedo and I. M. Llorente, "Efficient resource provisioning for elastic cloud services based on machine learning techniques," *Journal of Cloud Computing*, vol. 8, no. 5, pp. 1–18, 2019.

[9] C. Mei, D. Taylor, C. Wang, A. Chandra and J. Weissman, "Sharing-aware cloud-based mobile outsourcing," in *Proc. of the Fifth Int. Conf. on Cloud Computing*, Honolulu, HI, USA, pp. 408–415, 2012.

[10] S. Imai, T. Chestna and C. A. Varela, "Elastic scalable cloud computing using application-level migration," in *Proc. of the IEEE/ACM Fifth Int. Conf. on Utility and Cloud Computing*, Chicago, IL, USA, pp. 91–98, 2012.

[11] T. L. Botran, J. M. Alonso and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014. https://doi.org/10.1007/s10723-014-9314-7

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, Cesar A. F. De Rose and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience (SPE)*, vol. 41, no. 1, pp. 23–50, 2011.
[13] ETSI, Mobile edge computing (MEC): Framework and reference architecture, V2.1.1, 12019.
[14] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.
[15] A. J. Shalini Lakshmi and M. Vijayalakshmi, "CASCM^2: Capability-aware supply chain management model for QoS-Driven offload-participator selection in Fog environments," *Sadhana Journal*, vol. 47, pp. 1–14, 2020.
[16] M. Adhikari, M. Mukherjee and S. N. Srirama, "DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multi-level feedback queueing in IEEE," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5773–5782, 2020.
[17] X. Ma, S. Wang, S. Zhang, P. Yang, C. Lin *et al.,* "Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing," *Proceedings of the IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 968–980, 2021.
[18] D. T. Hoang, D. Niyato and P. Wang, "Optimal admission control policy for mobile cloud computing hotspots with cloudlet," in *Proc. of the IEEE Wireless Communications and Networking Conf. (WCNC)*, Paris, France, pp. 3145–3149, 2012.
[19] V. Chamola, C. Tham and G. S. S. Chalapathi, "Latency aware mobile task assignment and load balancing for edge cloudlets," in *Proc. of the IEEE Int. Conf. on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Kona, HI, USA, pp. 587–592, 2017.
[20] M. S. Asalnpour, M. Ghobaei-Arani and A. N. Toosi, "Auto scaling web applications in clouds: A cost-aware approach," *Journal of Network and Computer Applications*, vol. 95, pp. 26–41, 2017. https://doi.org/10.1016/j.jnca.2017.07.012
[21] R. N. Calheiros, R. Ranjan and R. Buyya, "Virtual machine provisioning based on analytical and QoS in cloud computing environments," in *Proc. Int. Conf. on Parallel Processing*, Taipei, Taiwan, pp. 295–304, 2011.
[22] A. Biswas, S. Majumdar, B. Nandy and Ali El-Haraki, "A hybrid auto-scaling technique for clouds processing applications with service level agreements," *Journal of Cloud Computing: Advances, Systems, and Applications*, vol. 6, no. 29, pp. 1–22, 2017.
[23] M. A. Razzaq, J. A. Mahar, M. Ahmad, N. Saher, A. Mehmood *et al.,* "Hybrid auto-scaled service-cloud-based predictive workload modeling and analysis for smart campus system," *EEE Access*, vol. 9, pp. 42081–42089, 2021.
[24] J. Mei, K. Li, A. Ouyang and K. Li, "A profit maximization scheme with guaranteed quality of service in cloud computing," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3064–3078, 2015. https://doi.org/10.1109/TC.2015.2401021
[25] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can computation offloading save energy?," *IEEE Computer Society*, vol. 43, no. 4, pp. 51–56, 2010. https://doi.org/10.1109/MC.2010.98
[26] B. G. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti, "CloneCloud: Elastic execution between mobile devices and cloud," in *Proc. of the Sixth Conf. Computer Systems, EuroSys*, Salzburg Austria, pp. 301–314, 2011.
[27] A. Computing, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, pp. 1–34, 2006.
[28] Amazon ec2. http://aws.amazon.com/ec2/. (accessed on May 04, 2021).
[29] M. Gordon, L. Zhang, B. Tiwana, R. Dick, Z. M. Mao *et al.,* "PowerTutor: A power monitor for android-based mobile platforms," 2013. [Online]. Available: http://ziyang.eecs.umich.edu/projects//powertutor/