QOS-BASED COST-EFFECTIVE OFFLOADING AS A SERVICE MIDDLEWARE FOR MOBILE CLOUD APPLICATIONS

Hamid Jadad[1] and Abderezak Touzene[2]

[1]Department of Computer Science, Dhofar University, Oman
[2]Department of Computer Science, Sultan Qaboos University, Oman

ABSTRACT

This paper proposes a distributed cloud middleware system that offers Offloading as a Service (OaaS) for mobile applications, aiming to satisfy user Quality of Service (QoS) requirements while reducing response time and overall cost. OaaS takes into consideration different features such as the user location (proximity to cloud resources); application requirements (e.g., OS, RAM, number of vCPUs); and desired QoS to offer an efficient offloading service that maximizes cloud resource utilization and reduces Virtual Machine (VM) rental costs. To ensure cost efficiency, OaaS dynamically selects VMs from various cloud providers based on pricing and adjusts the number of VMs in real-time to meet response time requirements. Using a predictive model based on queuing theory, the middleware can scale the VM pool up or down by forecasting workload demands. Simulation results confirm that our model outperforms existing algorithms in terms of response time and VM leasing costs, while meeting users' QoS requirements.

KEYWORDS

Mobile cloud, apps, offloading service, load balancing, prediction model, and queuing model.

1. INTRODUCTION

The rapid growth of smartphone usage has been driven by the affordability of smart devices and continuous technological advancements. One of the main drivers of this growth is the increasing demand for mobile [1]. As of May 2019, Google Play hosted approximately 2.1 million applications, while Apple's App Store offered around 1.8 million applications [2]These applications span a wide range of domains, including entertainment, business, healthcare, and education. Earlier mobile applications typically required low to moderate computation, limited memory, and minimal power consumption. However, modern applications such as multimedia processing, video gaming, speech recognition, and natural language processing are highly computation-intensive and demand substantial storage and processing resources, making local execution on mobile devices increasingly impractical.

Cloud computing offers an effective solution by providing elastic computation power and virtually unlimited storage at relatively low cost. Its flexibility, reliability, and cost-effectiveness have motivated widespread adoption across various domains. Several studies have explored cloud-based service provisioning, including IoT-cloud integration [3]and vehicular cloud services aimed at improving performance and availability [4] and [5].

To minimize latency and improve service performance, cloud providers have expanded their infrastructures globally by deploying geographically distributed data centers grouped into

regions. Each region typically contains multiple data centers offering similar services but at different prices depending on location-specific factors such as energy cost and taxation. For example, the hourly cost of an AWS EC2 m4.xlarge on-demand Windows instance differs across regions, with higher prices in the Asia Pacific region compared to the U.S. East region [6]. This pricing heterogeneity highlights the need for cost-aware resource management in cloud-based offloading systems.

Many existing mobile cloud offloading approaches adopt an application-level model in which identical versions of an application run on both the mobile device and the cloud server [7][8], [9]In such models, developers must predetermine which application components are offloadable. This requirement introduces development overhead and may lead to inefficient offloading decisions, particularly when large computation modules are transferred over the network, incurring transmission delays and bandwidth costs.

Cloud virtual machines (VMs) are typically leased for fixed minimum durations, such as one-hour billing periods for EC2 on-demand instances. When a VM is used for less than the billing period, the full cost is still incurred, leading to inefficient resource utilization. Several studies have explored sharing cloud resources among multiple mobile users[10].However, since a VM generally executes one task at a time, allocating multiple tasks to a single VM introduces a trade-off between maximizing VM utilization and minimizing response time. This trade-off has not been adequately addressed in existing work.

Another challenge is VM booting delay. VM startup times can range from tens of seconds to several minutes, depending on the cloud provider[11]. Many existing approaches assume on-demand VM creation during task scheduling[12-14]which can significantly increase response time and degrade offloading efficiency.

Latency further limits the applicability of mobile cloud offloading in large-scale, geographically distributed scenarios. Offloading requests to distant cloud data centers can substantially increase end-to-end response time. Approaches such as cloudlets[15] and mobile edge computing [16] aim to reduce latency by bringing computation closer to users. However, these solutions rely on localized deployment and are not well-suited for handling millions of concurrent applications originating from globally distributed users.

To address these challenges, this paper proposes an Offloading as a Service (OaaS) middleware framework that improves performance efficiency while reducing offloading cost. The framework leverages geographically distributed public cloud data centers and dispatches user requests to the nearest regional data center to minimize latency. To enhance scalability, the system employs distributed schedulers operating in parallel across regions, overcoming the performance limitations of centralized schedulers[17].

Unlike application-centric offloading approaches, the proposed OaaS framework adopts a middleware-level paradigm that treats mobile applications as cloud-resident services. It integrates a queueing-theory-based analytical performance model to guide SLA-aware and cost-efficient VM provisioning, enabling effective VM sharing, proactive VM pre-creation, and dynamic scaling. By jointly considering response time, SLA compliance, and cost efficiency, the proposed framework addresses key limitations of existing mobile cloud offloading solutions and clearly distinguishes itself from prior work.

The main contributions of this paper are as follows:

- Treating mobile applications as cloud-resident services, eliminating the need for dual mobile/cloud application versions.
- Introducing Offloading as a Service (OaaS) at the middleware level, rather than application-level or VM-centric offloading.
- Integrating a queueing-theory based analytical predication model that jointly drives VM scaling, SLA compliance and cost minimization.
- Enabling processor-sharing at the VM level, allowing multiple concurrent requests per VM while preserving QoS constraints.

The rest of the paper is organized as follows: Section 2 presents the proposed QoS-basedoffloading as a service system architecture. In section 3 presents our analytic prediction model used by the offloading as service middleware. Section 4 discusses the dynamic VM scaling algorithm. Section 5 presents experimental results. Finally, in section 6, we present our conclusion along with future work.

2. QOS BASED OAAS SYSTEM ARCHITECTURE

The general overview of our Performance based Offloading as a Service (OaaS) middleware architecture is shown in Figure 1. The overall scalable distributed system covers five regions (North America, South America, Europe, Russia, East Asia, and Australia) in the world depending on the availability of the cloud data centers. Note that the number of regions could be extended without a change in performance. The description of the different system components of OaaS is as follow

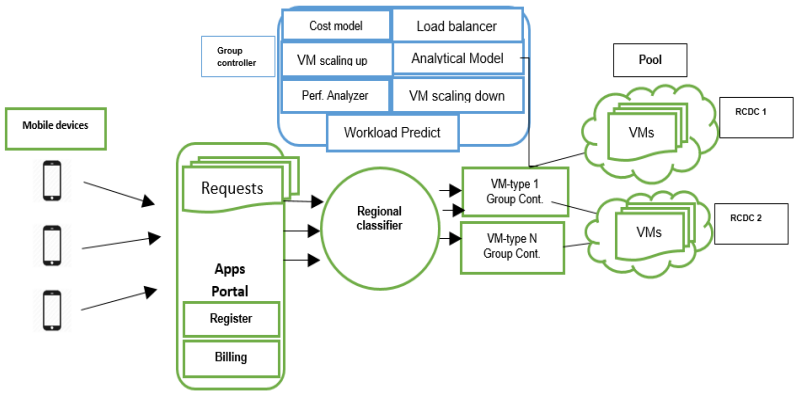

Figure 1: OaaS architecture for a single region with analytical model

- Mobile devices access directly to the OaaS portal for registering a new user, posting a new offloading request, and finally receiving the bill.
- Regional classifier will process the user request based on the user location and the user request parameter (O.S platform, maximum response time, etc.) to the appropriate region group controller. There are as many group controllers as different VM types. The VM type is by analogy to EC2 instance which is characterized by the O.S platform (Android, Apple, etc.), the number of vCPU, RAM, etc.
- Each group controller oversees many VMs of the same type provided by different region cloud center RCD.
- Load Balancer is a module in the VM-type group controller. It plays the role of a distributed scheduler by receiving a request from the regional classifier, then dynamically distributes received requests among a pool of available VMs. VMs are in different Cloud Data Centers (CDCs) within the same region.

- *Analytical model*: this component contains the algorithm (Section 4) that scales up and down adaptively the number of VMs within the pool of VMs to meet user QoS and maximize the VM utilization.
- *The workload predictor* is a module which retrieves the predicted workloads for next coming interval by getting the current number of VMs used within the pool and their computing capacity from the performance analyzer module. It also calculates the current average response time per request. In section 3, we present in more detail the analytical model which is the core module in our offloading service.
- *Cost Model:* The cost model used in this work is designed to support relative cost-aware virtual machine selection rather than to replicate exact cloud provider billing mechanisms. Given a set of available VM types across geographically distributed cloud data centers, the model selects the VM that offers the best trade-off between processing capacity and leasing cost for the current workload conditions.
- *VMs scaling up*: they are a part of VM-type group controller. It contains the VM scaling up algorithm (Section 4). It allocates the number of VMs required for the next interval. It receives the predicted workloads for the next interval from the Dynamic_VMs_scaling algorithm. It calls the cost model component to check the availability of CDCs and their capacity and selects a single VM with best fit in terms of capacity and minimum cost.
- *VMs scaling down*: The VM scaling down algorithm (Section 4) is implemented in this component. It aims to shut down a certain number of VMs for the next coming time interval. It obtains the predicted average response time for the next interval from the analytical model. More specification is presented in the next section.
- *Performance Analyzer*: it is a part of VM-type group controller. It monitors the current system's performance. It feeds the analytical model with the current number of VMs within the pool, and the user QoS requirements. It also provides the list of VMs with rental time of less than 10 minutes. It records the workloads of each interval to be used in the workload predictor.
- Virtual Machines (VMs): finally, each request is assigned to a pool of VMs that can meet user requirements. The VM can be shared by many apps concurrently taking into consideration VM utilization and complying with the users' QoS requirements.
- Scalability in the proposed OaaS framework is further supported by its distributed regional architecture. Each region operates independent group controllers and load balancers that schedule requests in parallel. This decentralized design avoids single points of congestion and enables incremental scaling by adding new regions or cloud data centers without modifying the core middleware logic.

3. OAAS ANALYTICAL MODEL

To guarantee an acceptable level of quality of service to the user a limited number of requests can share a processor [18]. Many research works have been conducted on mobile cloud offloading systems such as [7] and [19-21]. However, most of these works focus on architectural or algorithmic aspects of the offloading operation. Thus, there is a lack of analytical analysis to evaluate the performance of offloading systems that takes into consideration the limited processor sharing capability in the cloud environment. There is a potential for the offloading system provider to know the effectiveness of the offloading system and what elements influence its performance. The authors in [22] proposed an analytical model to express the performance of offloading systems in a mobile wireless environment. They investigate the impact of the probability of the unreachability of surrogates on the performance of the offloading system. Salah et al. [23] presented a Markovian analytical model to estimate the service response time for elastic cloud applications. Aljohani et al. [24] introduced a simple queueing model to analyze the performance metrics of a web server in the cloud computing under varying traffic loads. In [25]

the authors studied the performance of a content delivery network peer-to-peer content delivery network under the processor-sharing discipline.

Inspired by the above works and driven by the increasing need for such an analytical model for analyzing the performance of mobile cloud offloading systems, this section investigates specifically the performance of our OaaS system under the processor-sharing environment. The analysis results can provide useful guidance to the design of efficient offloading systems.

In this section, we present a queueing theory model for the OaaS system described in section 2. Recall that, the OaaS system contains independent parallel load balancers. Each load balancer is associated with a pool of VMs that can be located on different physical servers and within different cloud data centers. As the user requests arrive at the OaaS system, they are classified in the Regional Classifier (RC) based on the app's requirements and forwarded to the proper VM_type Group Controller's Load Balancer (LB). The LB assigns the arriving requests to the corresponding pool of VMs. Each VM may process multiple requests simultaneously. Unlike other works on offloading, we allow the VM processor to be shared by multiple requests at the same time. Processor sharing improves the VM utilization and reduces the number of leased VMs and hence reduces the offloading cost for both the client and the service provider.

The queuing modelling with processor sharing discipline is commonly used in modelling computer system and networks [26]. In our system, the server or Virtual Machine (processor) capacity is shared among all requests arriving to a given group controller within a specific region. However, the service rate per request at any time is inversely proportional to the total number of requests existing in the service at that time. In other words, the total rate at which the server performs work is constant, but if there are N requests in the system then each request is receiving service at $(1/N)$-th of the rate at which it would receive service if it had the server for itself. Of course, the rate at which requests receive service changes each time a new arrival joins the system and each time a completed unit departs [27].

The analytical model developed in this work relies on a set of commonly adopted assumptions in queueing-based performance[23-25]. Incoming offloading requests are assumed to follow a Poisson arrival process, which is widely used to model independent and large-scale user request patterns in mobile and cloud systems. Service times are assumed to be exponentially distributed, reflecting variability in task execution times and enabling tractable analysis. In addition, a process-sharing (PS) discipline is assumed at the virtual machine level, allowing multiple concurrent requests to share VM processing capacity. These assumptions are consistent with prior analytical studies on mobile cloud offloading and cloud service performance and provide a reasonable abstraction for large-scale offloading environment.

Since the VM_type Group Controllers of the OaaS system are independent, we capture the process of offloading to a single VM_type Group Controller managing a single pool of VMs as shown in figure 2. The process has the following features:

- Multiple clients can submit offloading requests independently.
- The load balancer assigns the arrival requests to the corresponding pool of VMs using different assignment strategies.
- VMs may belong to different cloud service providers.
- Each VM can process N requests simultaneously.
- Assuming all VMs in the pool have different computing capacities (EC2 vCPU).
- Each request has a different service time requirement.

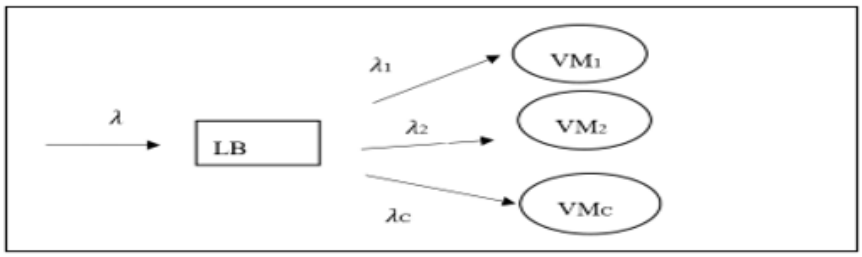

Figure 2: OaaS VM_type Group Controller Queueing Model

We model the VM-type Group Controller as queueing model.

- Arrival rate at the load balancer: we assume that incoming requests arrive at a rate of λ requests per unit of time following a Poisson distribution. This arrival rate λ is estimated by the system at different times in the day. A simple estimation could be using regression model based on the arrivals of previous time intervals
- Service time at each VM_i: we assume all the service times are independent and exponentially distributed with means of $1/\mu_i$. The service rate is directly related to the vCPU speed (from the EC2 instance) and the size of an average task (see Table 1).
- The VM_i load is given by $\rho_i = \lambda_i / \mu_i$. We assume $\rho_i < 1$.
- Discipline: we assume that each VM can serve a maximum of N requests simultaneously using the processor sharing discipline.
- Thus, the VM-type Group Controller is modeled as an M/M/C/N/PS queuing model.

We use this queuing model to estimate the average response time of the offloading service for a given request from a given individual user. The response time is a measure of the amount of time the offloading service consumes for processing a client request. Since the response time is an important measure of an application's performance, we can use it to determine whether the specified offloading service violates the Service Level Agreements (SLA). In the SLA, the accepted range of response times will be described as a Quality-of-Service criterion.

In this work, the Service Level Agreement (SLA) between the OaaS provider and mobile clients is defined primarily in terms of average response time, which represents the end-to-end latency experienced by an offloading request. Average response time is widely used as a primary QoS indicator in mobile cloud systems due to its direct impact on user experience and its suitability for analytical modeling [19],[23],and [28].

Here, we first analyze the performance of the offloading system of a single VM based on M/M/1/N/PS model and we estimate the average response time. Then we derive the response time of the OaaS system that contains a pool of VMs, which is modeled as M/M/C/N/PS under different assignment strategies.

3.1 Analyzing the Performance of a Single VM

In this section, we analyze an M/M/1/N/PS model and compute the average response time for a request. Recalling, the average response time of a request denoted $E(T)$ is defined as the sum of the average service time $E(S)$ and the average waiting time $E(W)$. Thus, to derive the average response time we first compute the average service time and average waiting time for a request.

We model a VM as an infinite capacity M/M/1/N/PS queue model with Poisson arrival rate λ, and exponentially distributed service time μ. In addition to that, the VM load is given by $\rho = \frac{\lambda}{\mu}$ and assumed ρ<1. We assume that each VM can serve a maximum of *N* requests simultaneously (processor sharing discipline). The arriving request will be served immediately if the number of requests in the VM is less than *N*, otherwise it should wait.

From [29] and [30] we can get the average service time as:

$$E(S) = \frac{1-\rho^N}{\mu(1-\rho)} \quad (1)$$

Like what we have done to derive the average service time, the average waiting time is derived by analyzing the system in case the number of requests in system is less than system capacity *N* and when the number of requests is more than the system capacity *N*. If there are less than *N* requests in the system, a request will not need to wait until it arrives at the system. So, from [29] and [30] the average waiting time is:

$$E(W) = \frac{\rho^N(1-\rho^N)}{\mu(1-\rho)^2} \quad (2)$$

As we mentioned above, the average response time is the sum of the average service time and average waiting time, thus from (1) and (2), we can get the average response time as:

$$E(T) = E(S) + E(W)$$

$$E(T) = \frac{(1-\rho^N)(1-\rho+\rho^N)}{\mu(1-\rho)^2} \quad (3)$$

3.2 Analyzing the Performance of a Pool of VMs

In this section, we use the results presented in the previous section to derive the average response time under different assignment strategies. Assume that the number of VMs in the system is *C*, and the service rate of VM$_i$is μ_i. The maximum number of concurrent users of VM$_i$is N_i. Let λ denote the total Poisson arrival rate. A user is dispatched to one of the *C* VMs independently in random using a probability distribution p_i , i=1,…..,*C*. So, each VM is an M/M/1/N/PS model as mentioned in the previous section.

The arrival rate of VM$_i$ is:

$$\lambda_i = p_i \lambda \quad (4)$$

and the utilization rate of VM$_i$ is

$$\rho_i = \frac{\lambda_i}{\mu_i} \quad (5)$$

One way for the distribution probabilities policy is to adopt an even distribution to all VMs in the pool regardless of their computing probability:

$$p_i = \frac{1}{C} \quad (6)$$

$$p_i = \frac{\mu_i}{\sum_i \mu_i} \quad (7)$$

One more rational choice for the distribution policy is to balance the load (LB module) equally among the servers (VMs), so that $\rho_i = \rho_j$ for all i and j in the pool. To this end, this strategy uses the probability distribution:

which then gives

$$\rho_i = \frac{\lambda_i}{\mu_i} = \frac{\lambda}{\sum_i \mu_i} = \rho \quad (8)$$

The resulting system is stable as long as the offered load ρ is less than one. Then according to (3) and (7), the average response time of VM_i is

$$E_i(T) = \frac{(1-\rho_i^{N_i})(1-\rho_i+\rho_i^{N_i})}{\mu_i(1-\rho_i)^2}, i = 1, 2, \ldots, C \quad (9)$$

Therefore, the average response time of the whole system is

$$E(T) = \sum_{i=1}^{C} p_i E_i(T) = \frac{1}{\sum_{i=1}^{C} \mu_i} \sum_{i=1}^{C} \frac{\rho_i^{N_i}(1-\rho_i^{N_i})(1-\rho_i+\rho_i^{N_i})}{(1-\rho_i)^2} \quad (10)$$

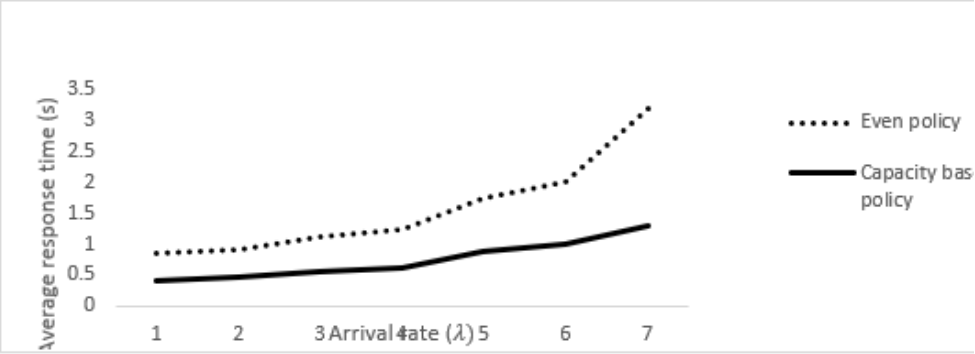

Figure 3: Comparison between even and capacity-based distribution policy

For measuring the performance of the OaaS system multi-servers under capacity based and even distribution policy. We set $N = 2, \mu_1 = 5, \mu_2 = 7$ for the two VMs. The probability distribution of the capacity distribution strategy is $p_1 = \frac{\mu_1}{\mu_1+\mu_2} = \frac{5}{5+7} = 0.42$, $p_2 = \frac{7}{5+7} = 0.58$. The probability distribution of the even distribution strategy is $p_1 = 0.5,\ p_2 = 0.5$. Figure 3 shows the average response time under capacity and even distribution policies. The results show that the capacity distribution strategy has minimum average response time compared to the even distribution strategy. For the mentioned reason, we decided to adopt the capacity-based policy for our load balancer module (LB).

While the analytical model relies on simplifying assumptions, it is not intended to replicate all aspects of real-world mobile cloud workloads. Rather, it serves as a predictive and decision-support tool that enables fast estimation of response time trends and resource requirements, which would be computationally expensive to obtain through continuous simulation or real-time measurement. The abstraction provided by the model allows the OaaS middleware to make timely VM scaling and scheduling decisions while preserving SLA constraints. More complex workload characteristics and non-Markovian behaviors can be incorporated as extensions in future work.

While average response time is the primary QoS metric considered in this study, other important QoS dimensions such as tail latency, service reliability, availability, and fault tolerance are not explicitly modeled. The focus on average response time allows the proposed analytical framework to remain tractable and enables fast prediction-driven scaling decisions within the OaaS middleware.

The proposed system is therefore positioned as a foundational framework that establishes the core mechanisms for SLA-aware offloading, resource scaling, and cost optimization. Extending the framework to incorporate richer SLA metrics is a natural and promising direction for future work.

4. DYNAMIC VMS SCALING ALGORITHMS

This section describes the dynamic VM scaling mechanism employed by the OaaS middleware to adapt resource provisioning to workload variations. Scaling decisions are directly driven by the analytical response time model introduced in Section 3. At each time interval, the predicted average response time (pRT), computed using Equation (10), is compared against the maximum allowable response time defined in the SLA (mRT). Based on this comparison, the system triggers either VM scaling up or scaling down to maintain QoS while minimizing cost.

The analytical model component periodically retrieves predicted workloads for the next interval, along with the current VM pool configuration, and computes pRT. A fixed time slot of 10 minutes is adopted to accommodate VM boot and pre-provisioning delays. If pRT exceeds mRT, the scaling-up algorithm is invoked; otherwise, the scaling-down algorithm is applied.

Dynamic_VMs_Scaling_Algorithm()

Input:
– Predicted workload arrival rate (λ)
– Current number of VMs (C) and their service rates (μ_i)
– Maximum allowable average response time (mRT) defined in the SLA
– Maximum number of concurrent requests per VM (N)
Output:
– Decision to scale up or scale down the number of VMs for the next time interval
Decision Logic:
The algorithm computes the predicted average response time (pRT) using the analytical model (Equation (10)). If pRT exceeds mRT, the scaling-up algorithm is triggered; otherwise, the scaling-down algorithm is invoked.

4.1 Vm Scaling Up Algorithm

The VM scaling-up algorithm incrementally provisions additional VMs until the predicted response time satisfies the SLA constraint. At each iteration, a VM is selected from available cloud data centers based on the minimum cost-to-capacity ratio to ensure cost efficiency. The response time is recalculated after each addition using Equation (10). This process continues until pRT ≤ mRT, ensuring SLA compliance with minimal resource cost.

VM Scaling Up Algorithm

Input
- **pRT:** Predicted average response time for the next time interval
- **mRT:** Maximum allowable average response time defined in the SLA

Output
- An updated virtual machine pool with additional VMs provisioned to satisfy QoS requirements and maximize resource utilization for the next time interval

Decision Logic:
- **Condition evaluated:**
  Predicted average response time (pRT) vs. SLA threshold (mRT)
- **Decision rule:**
  - If pRT>mRT → add a VM
  - Else → no action
  - Maintain QoS while minimizing cost

4.2 Vm Scaling Down Algorithm

The VM scaling-down algorithm aims to reduce over-provisioning while preserving QoS. VMs are prioritized for removal based on the shortest remaining rental time to minimize cost waste. A VM is removed only if the recalculated pRT remains within the SLA threshold. Pending requests are allowed to complete before VM termination, ensuring service continuity.

VMs Scaling Down Algorithm

Input
- **pRT:** Predicted average response time for the next time interval
- **mRT:** Maximum allowable average response time defined in the SLA
- **LRRT:** List of virtual machines with remaining rental time less than or equal to 10 minutes

Output

- A reduced set of active virtual machines that maintains QoS compliance while

Decision Logic

- The algorithm safely removes virtual machines only when the predicted average response time remains within SLA limits.
- Virtual machines nearing the end of their rental period are prioritized to minimize cost waste.
- Response time is recalculated after each potential removal to ensure QoS is never violated

By explicitly linking the VM scaling decisions to the analytical model, the proposed algorithm ensures that resource provisioning is both SLA-aware and cost-efficient. This tight integration between analytical performance modeling and operational scaling distinguishes the proposed OaaS from heuristic-based scaling approaches.

Real-world cloud pricing models are inherently complex and vary across providers, regions, and pricing schemes (e.g., on-demand, reserved, and spot instances). Modeling such heterogeneity in full detail would significantly increase computational complexity and hinder fast decision-making within the OaaS middleware. Therefore, this work adopts an abstract cost model that enables fast, scalable, and provider-agnostic decision-making, which is essential for large-scale offloading scenarios involving millions of requests. The proposed framework is, however, extensible, and more sophisticated pricing models—such as region-dependent pricing, dynamic spot markets, and multi-provider cost heterogeneity—can be seamlessly integrated as future enhancements.

5. PERFORMANCE EVALUATION OF THE OAAS

The performance of the proposed OaaS framework is evaluated using simulation-based experiments. Simulation is adopted as an initial validation step to allow controlled, repeatable, and scalable evaluation of system behavior under varying workload intensities and SLA constraints. This approach enables systematic comparison of different resource management strategies while isolating the impact of the proposed analytical and scaling mechanisms.

In this work, scalability is considered from an architectural and algorithmic perspective, rather than as an absolute performance guarantee. The proposed OaaS framework adopts a distributed regional architecture with parallel schedulers, enabling workload distribution across geographically separated cloud data centers. This design allows the system to scale horizontally by adding regions and virtual machines as demand increases.

To demonstrate the effectiveness of the proposed approach, several simulations studies are conducted. In experiment 1, we will start by validating our analytical model and comparing its results with simulation results. In experiment 2, we studied the effect of the scale up algorithm and the way it selects the best VMs to be added to the pool to reduce the response time and the VM leasing cost. In experiment 3, we study the impact of having requests with different SLAs and its effects on the performance of OaaS system. In Experiment 4, we compare our approach with one of the recent approaches in the literature presented in [28], we name it Zhang method.

5.1 Simulation Setup

The simulation models are implemented using the C++ programming language. We have used the following metrics to evaluate their performance:

1. Average response time: the average time a request takes to get a response from the system including the waiting time in the queue and the execution time.
2. Average VMs utilization: the average utilization rate of all VMs in the pool per hour.
3. Number of allocated VMs per hour.
4. Cost of average unutilized rate of VMs per hour.

The request arrival rate is based on the model presented in [32]. This model represents the number of mobile applications requests for a specific hour of the day (figure 4). We assume each request requires 100 FLOPS and 1 MB of memory as in the [28] paper.

Some assumptions were made to run the simulations unless stated otherwise. All arrived requests require the same VM-type A with the following specifications: 10 MHz CPU, 10 MB memory and costs 0.06 $ per hour. We set the maximum CPU rate for each VM in the OaaS simulation to 10 requests per second. The simulation is repeated 10 times, and we display the average of each performance metric for each method. We started both simulations at 12 am with 5 VMs, similar to the number of VMs used in [28].

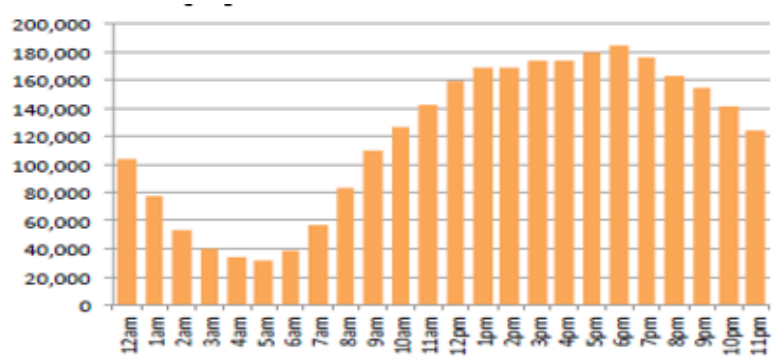

Figure 4 :Number of requests per hour of the day [32]

5.2 Experiment 1: Oaas Analytical Model Validation

The objective of this experiment is to validate the accuracy of the proposed analytical model by comparing its predicted response times with simulation-based results under identical workload conditions. This validation step is essential to assess whether the analytical predications reliably capture system behavior and can be safely used as a decision-support mechanism for VM scaling and SLA enforcement.

To this end, we study the effect of arrival rate during the day (Figure 4) and its impact on the system and compare the results given by our analytical model and the simulation results. From figures 5, we can see that the shapes of the results of both analytical and simulation are very similar, indicating strong agreement between the two approaches. This close match confirms that, despite relying on simplifying assumptions, the analytical model provides accurate and stable estimates of system performance across varying workload intensities.

Since the analytical model forms the core of the proposed OaaS framework, its accuracy and computational efficiency are critical to the success of the system. Compared to simulation-based estimation, analytical response-time prediction enables significantly faster decision-making, which is essential for timely VM scaling and dynamic SLA-aware resource management.

5.3 Experiment 2: Efficiency of OaasVM Selection for Scale Up Algorithm

In this simulation, we will evaluate the efficiency of the OaaS system after using the analytical model to improve the system. The analytical model is used to predict the number of VMs that will handle the predicted workload of the next time slot. The main challenge facing this incremental approach is how to determine the VM capacity each time it needs to add until reaching the required agreed average response time. Our scaling up algorithm chooses the VM with minimum cost and capacity (or service rate) to be added in the pool until reaching the agreed average response time.

Here, we show the efficiency of this strategy compared with choosing VM with maximum capacity and VM with the same capacity strategies. We begin the period with 4 VMs with capacity to process 10 requests per second. We assume the cloud has three types of VMs the algorithm can select one VM from these types of each time need to scale up. We call them Same_VM_Capacity(capacity=10 requests/second, cost=1$ per/hour), Minimum_VM_Capacity (capacity=4 requests/second, cost=0.4$ per/hour), and Maximum_VM_Capacity (capacity=16 requests/second, cost=1.6$ per/hour). Now we compare the efficiency of the algorithm when using each strategy. We use the same workloads model used in figure 4. We use the average total cost of renting VMs and average total utilization rate of VMs as metrics to measure the performance of the proposed algorithm.

To evaluate the efficiency of the OaaS system after using the analytical model, we show the efficiency of the scaling up algorithm. The algorithm is choosing the VM with minimum cost and capacity to be allocated to handle the predicted workloads. This strategy ensures that the algorithm maximizes the utilization rate of VMs and minimizes the total rental cost of VMs. When choosing the same VM capacity to add to the pool, the VM with the minimum capacity, and the VM with the maximum capacity.

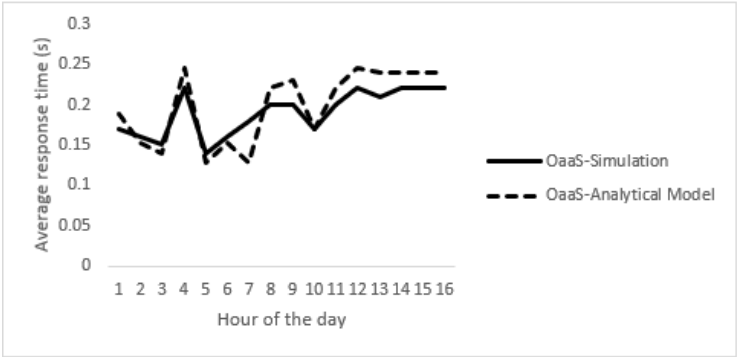

Figure 5: The average response time of a request according to the hour of the day

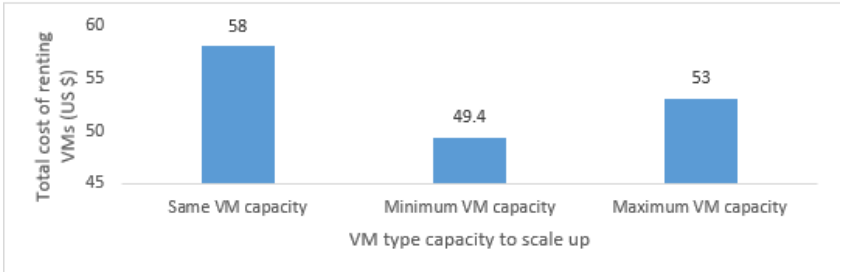

Figure 6. The average total cost of renting VMs at the end of period

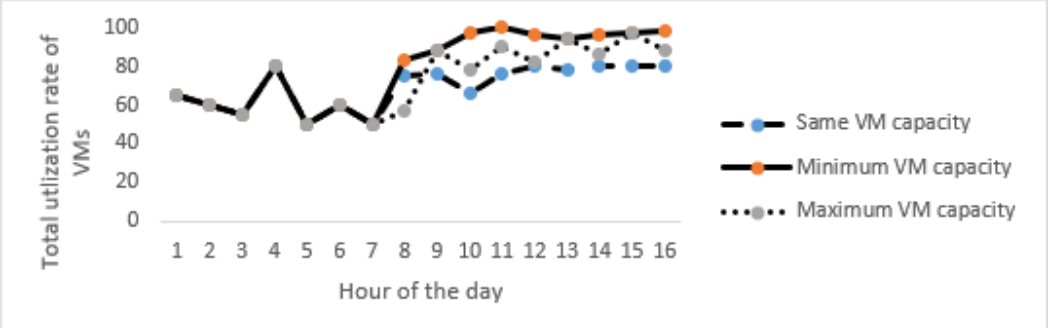

Figure 7. Average total utilization rate of VMs according to the hour of the day

From both figures 6 and 7, we can see that the Minimum_VM_capacity strategy, which is used by the proposed scaling up algorithm, provides the minimum average total cost for renting VMs at the end of the simulation period. In addition, it maximizes the utilization rate of the VMs. This makes the algorithm cost-effective while it meets the QoS requirements compared to both the Same_VM_capacity and maximum_VM_capacity strategies.

5.4 Experiment 3: Oaas Handling Different Requests Slas

In this scenario, we study the performance of the OaaS system after plugging the analytical model as a prediction to scale up and down the number of VMs for the next interval. We want to show how the OaaS system can handle different incoming requests of different SLAs. We set the maximum average response time (*mRT1*) for SLA1 to 0.035 second, and (*mRT2*) for SLA2 to 0.045 seconds. We set the VM service rate to 50 request/seconds. We assume the VMs are identical and even use distribution strategies.

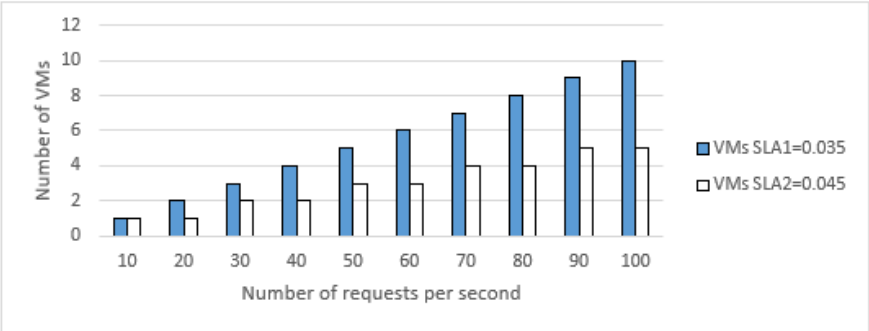

Figure 8: Number of VMs required for two different SLAs requests

From Figure 8, we can see that overall, the number of required VMs increases as the number of requests increases to meet SLA requirements. Furthermore, it is obvious that the OaaS system allocates more VMs to meet SLA1 compared to SLA2. Because SLA1 has smaller *mRT* (Maximum average response time) and as the number of incoming requests increases, the OaaS system scales up the number of VMS to ensure meeting the SLA1 requirements.

5.5 Experiment 4: Performance Oaas Versus Zhang Method

To assess the effectiveness of the proposed OaaS framework, we compare its performance against a recent state-of-the-art cloud task scheduling approach proposed by Zhang and Zhou (2018). This method was selected as a representative baseline due to its relevance to dynamic

task scheduling and resource allocation in cloud environments. Zhang's approach uses a static VM allocation (5 VMs in this experiment) policy and assigns requests to available VMs one at a time. Other requests should wait until the current requests complete.

The results in figure 9 also show that the OaaS by maintained a good VMs utilization and scaling up the number VMs when needed it manages to keep a lower response time, 0.2 seconds at high load (6 PM), when the Zhang algorithm has a larger response time for the same period and the same load. (0.9 seconds). Clearly, Zhang algorithm performs badly after the period of 7 AM where the load is increased and the fixed number of VMs becomes highly saturated which is reflected in figure 4 by an exponential shape for the response time. This issue might constitute violating the QoS requirements. On the contrary, in the proposed approach, the shape of the response time curve is almost constant while varying the load in accordance with the requests SLA. This confirms the scalability of our performance based OaaS system.

The comparison demonstrates that the proposed OaaS framework maintains stable response time and higher VM utilization under increasing workload conditions, whereas the baseline approach suffers from performance degradation due to static resource allocation.

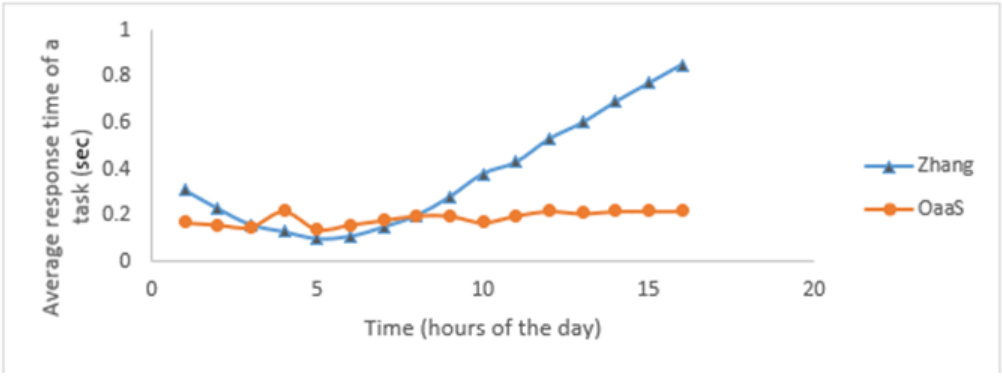

Figure 9. Average response time of a request according to the hour of the day

While simulation-based evaluation provides valuable insights into the behavior and effectiveness of the proposed framework, it does not capture all characteristics of real-world cloud environments. Factors such as network variability, hardware heterogeneity, and provider-specific operational constraints are abstracted to ensure tractable and repeatable analysis.

Nevertheless, the simulation results represent a necessary first step in validating the proposed analytical model and resource scaling mechanisms. Although the architecture is designed to support scalable operation, the experimental evaluation does not claim to exhaustively demonstrate performance on an Internet-wide scale. Instead, the results provide evidence of scalability trends and architectural feasibility. Comprehensive validation through real-world deployment and comparison against a broader range of state-of-the-art approaches remains an important direction for future work.

6. CONCLUSION

This paper presented an Offloading as a Service (OaaS) middleware framework for scalable and cost-efficient mobile application offloading in geographically distributed cloud environments. By integrating a queueing-theory-based analytical model with SLA-aware, cost-driven VM provisioning, the framework enables effective dynamic resource scaling.

Simulation results validated the accuracy of the analytical model and demonstrated stable response-time performance under increasing workloads, outperforming a recent state-of-the-art baseline that relies on static resource allocation. Although the evaluation is simulation-based, the results indicate clear scalability trends and architectural feasibility.

Future work will focus on deploying the OaaS framework on commercial cloud platforms to evaluate real-world performance under heterogeneous workloads and pricing models. Additional extensions will explore machine learning–based prediction, richer QoS metrics, and large-scale multi-region deployments.

REFERENCES

[1] GSMA corporate, "The mobile economy report," 2017.

[2] Statista, "Number of apps available in leading app stores as of 1st quarter 2019." Accessed: Jun. 30, 2019.

[3] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "Integration of Cloud computing and Internet of Things: A survey," Future Generation Computer Systems, vol. 56, no. Supplement C, pp. 684–700, 2016.

[4] Al Ridhawi, M. Aloqaily, B. Kantarci, Y. Jararweh, and H. T. Mouftah, "A continuous diversified vehicular cloud service availability framework for smart cities," Computer Networks, vol. 145, pp. 207–218, 2018.

[5] R. Alakbarov and O. Alakbarov, "Procedure of Effective Use of Cloudlets in Wireless Metropolitan Area Network Environment," International Journal of Computer Networks and Communications, vol. 11, no. 1, 2019.

[6] P. ALGUACIL, "Comparing the geographical coverage of AWS, Azure and Google Cloud." Accessed: Jun. 18, 2019.

[7] B. Chun, S. Ihm, and P. Maniatis, "Clonecloud: elastic execution between mobile device and cloud," EuroSys '11, pp. 301–314, 2011.

[8] S. Yang, D. Kwon, H. Yi, Y. Cho, Y. Kwon, and Y. Paek, "Techniques to minimize state transfer costs for dynamic execution offloading in mobile cloud computing," IEEE Trans. Mob. Comput., vol. 13, no. 11, pp. 2648–2660, 2014.

[9] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9.

[10] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "MuSIC: Mobility-aware optimal service allocation in mobile cloud computing," IEEE International Conference on Cloud Computing, CLOUD, pp. 75–82, 2013.

[11] M. Mao and M. Humphrey, "A Performance Study on the VM Startup Time in the Cloud," in Proceedings of IEEE International Conference on Cloud Computing (CLOUD), IEEE, 2012, pp. 423–430.

[12] P. Pillai, M. Satyanarayanan, Y. Abe, W. Richter, and K. Ha, "Just-in-time provisioning for cyber foraging," p. 153, 2013.

[13] S. Chilukuri, S. Bollapragada, S. Kommineni, and K. Chakravarthy, "RainCloud - Cloudlet selection for effective cyber foraging," IEEE Wireless Communications and Networking Conference, WCNC, pp. 1–6, 2017.

[14] S. Durga, S. Mohan, J. D. Peter, and S. Surya, "Context-aware adaptive resource provisioning for mobile clients in intra-cloud environment," Cluster Comput., vol. 1, pp. 1–14, 2018.

[15] M. Satyanarayanan, "Cloudlets," p. 1, 2013.

[16] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices," IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 3590–3605, 2016.

[17] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in Proceedings of the 8th ACM European Conference on Computer Systems, 2013, pp. 351–364.

[18] A. N. Dudin, "Analysis of queueing model with processor sharing discipline and customers impatience," Operations Research Perspectives, vol. 5, no. June, pp. 245–255, 2018.

[19] E. Cuervo et al., “MAUI: Making Smartphones Last Longer with Code Offload,” MobiSys’10, pp. 49–62, 2010.
[20] H. Wu, Y. Sun, and K. Wolter, “Energy-Efficient Decision Making for Mobile Cloud Offloading,” IEEE Transactions on Cloud Computing, vol. 7161, no. c, 2018.
[21] W. Junior, E. Oliveira, A. Santos, and K. Dias, “A context-sensitive offloading system using machine-learning classification algorithms for mobile cloud environment,” Future Generation Computer Systems, vol. 90, pp. 503–520, 2019.
[22] S. Ou, K. Yang, A. Liotta, and L. Hu, “Performance Analysis of Offloading Systems in Mobile Wireless Environments,” 2007.
[23] Salah and R. Boutaba, “Estimating Service Response Time for Elastic Cloud Applications,” pp. 12–16, 2012.
[24] A. M. D. Aljohani, D. R. W. Holton, and I. Awan, “Modeling and performance analysis of Scalable Web Servers Deployed on the Cloud,” 2013.
[25] X. Zhang and B. Yin, “Performance analysis of CDN-P2P networks based on processer-sharing queues,” in 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2017, pp. 24–27.
[26] Kleinrock, Queueing systems, volume 2: Computer applications, vol. 66. wiley New York, 1976.
[27] E. G. Coffman and R. R. Muntz, “Waiting Time Distributions for Processor-Sharing Systems,” vol. 17, no. 1, pp. 123–130, 1970.
[28] P. Y. Zhang and M. C. Zhou, “Dynamic Cloud Task Scheduling Based on a Two-Stage Strategy,” IEEE Transactions on Automation Science and Engineering, vol. 15, no. 2, pp. 772–783, 2018.
[29] C. Knessl, “On finite capacity processor-shared queues,” SIAM J. Appl. Math., vol. 50, no. 1, pp. 264–287, 1990.
[30] X. Zhang, B. Yin, and H. Shi, “Performance analysis of multi-server based on processor-sharing queue,” International Conference on Advanced Communication Technology, ICACT, vol. 2016-March, pp. 843–848, 2016.
[31] P. Y. Zhang and M. C. Zhou, “Dynamic Cloud Task Scheduling Based on a Two-Stage Strategy,” IEEE Transactions on Automation Science and Engineering, vol. 15, no. 2, pp. 772–783, 2018.
[32] Böhmer, B. Hecht, J. Schöning, G. Bauer, A. Krüger, and G. Bauer, “Falling asleep with Angry Birds, Facebook and Kindle: a large-scale study on mobile application usage,” Proceedings of the 13th international conference on Human computer interaction with mobile devices and services, vol. ACM, pp. 47–56, 2011.

AUTHORS

Hamid A. Jadad is an Assistant Professor of Computer Science at Dhofar University, Oman. He holds a Ph.D. in Computer Science from Sultan Qaboos University and an M.Sc. in Internet, Computer and System Security from the University of Bradford, UK. His research focuses on mobile cloud computing, task offloading, and context-aware systems, with several publications in international journals and conferences. Dr.Jadad is actively involved in curriculum development, particularly for inclusive education, and is currently leading the design of an MSc in Artificial Intelligence. He holds industry certifications including CCNP, CCNA (Security), and CompTIA Security+.

AbderezakTouzene received the BSc. degree in Computer Science fromthe University of Algiers in 1987, M.Sc. degree in Computer Science from Orsay Paris-Sud University in 1988, and Ph.D. degree in Computer Science from the Institute Polytechnique Grenoble (France) in 1992. He is a full Professor and Head of the Department of Computer Science at Sultan Qaboos University in Oman. His research interests include Cyber Security, Smart City, Smart Grid, Mobile and Cloud Computing, Parallel and Distributed Computing, Wireless Sensors and Mobile Networks, Internet of Things (IoT), Network on Chip (NoC), Interconnection Networks, Performance and Evaluation, Numerical Methods.