Queueing Model based Dynamic Scalability for Containerized Cloud

Ankita Srivastava*, Narander Kumar
Department of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India

***Abstract*—Cloud computing has become a growing technology and has received wide acceptance in the scientific community and large organizations like government and industry. Due to the highly complex nature of VM virtualization, lightweight containers have gained wide popularity, and techniques to provision the resources to these containers have drawn researchers towards themselves. The models or algorithms that provide dynamic scalability which meets the demand of high performance and QoS utilizing the minimum number of resources for the containerized cloud have been lacking in the literature. The dynamic scalability facilitates the cloud services in offering timely, on-demand, and computing resources having the characteristic of dynamic adjustment to the end users. The manuscript has presented a technique which has exploited the queuing model to perform the dynamic scalability and scale the virtual resources of the containers while reducing the finances and meeting up the user's Service Level Agreement (SLA). The paper aims in improving the usage of virtual resources and satisfy the SLA requirements in terms of response time, drop rate, system throughput, and the number of containers. The work has been simulated using Cloudsim and has been compared with the existing work and the analysis has shown that the proposed work has performed better.**

***Keywords*—Cloud computing; scalability; containers; containerized cloud models; queueing model**

I. INTRODUCTION

Cloud computing has evolved into a highly dynamic computing model. It has gained attraction from various organizations due to its cost, availability, scalability, and security. It is an internet-based computing technology that provides higher-end computation and a shared pool of resources which are accessible on demand [1]. It has revolutionized the internet world through its hosting services and computational ability. Its unique technology has facilitated the user to pay for only those services and resources which have been demanded by them and further these resources can be increased and decreased depending upon the requirement. The potential and high capabilities have led to amplified productivity with reduced costs and flexibility as against the other IT industries [2]. The prime technology working behind the cloud is virtualization which enabled the cloud to instantiate various Virtual Machines (VMs) on one single physical machine (PM). Virtualization can occur at various levels like desktop, network, storage, and application [3]. It can affirm high performance, confidentiality, reliability, and security among VMs. One VM is isolated from the other VM on the same PM making it securely isolated. Despite various benefits exhibited through virtualization, applications demanding less isolation and maximum flexibility at runtime, VM virtualization may not be sufficient enough to satisfy all the QoS standards [4]. The container-based virtualization is gaining more popularity these days because of the more dynamic and flexible nature of the workload which varies highly with time. It expedites the seamless movement of applications from one architecture to another as against the VMs virtualization. Container executes on a kernel with the equivalent performance as VMs but with lesser cost than expensive VM runtime management overhead [5]. Containers provide a good platform to execute microservices on the cloud and they provide good support for the technologies such as fog computing, and the Internet of Things (IoT) [6]. As container technology gained popularity various large-scale IT industries providing cloud services have come up with their container-based cloud services.

The most renowned service models available in the cloud are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) with various energy efficient datacenters (DC) which are solely responsible for managing the scalability through resource management and load optimization [7]. With the PaaS service, the users can deploy any applications on the cloud. This model encapsulates the underlying infrastructure and facilitates the user to deploy the applications anywhere without giving a single thought about infrastructure management. One of the components of the PaaS service is containers and they are its enablers [8]. So, a user application can be deployed on a single cloud infrastructure as a unique block or deployed separately in different cloud infrastructures.

The key characteristic of the cloud to scale up has attracted a lot of users. The variation and fluctuation in workload have compelled the cloud providers to scale up the resources (VMs or containers) dynamically as per the requirement. The cloud has eased the process of obtaining and releasing resources but it can be challenging to decide how many resources are needed to handle a fluctuating workload. There is an urgent demand for a model which can provision and de-provision the resources dynamically at the burst of demands. Despite the development of container technology and harassing its potential, there is still room for the improvement in dynamic scaling of cloud resources. Insufficient scalability which is not competent enough to confront the variation in the workload intensity may lead to under-provisioning (UP) or over-provisioning (OP) of the resources. In the UP scenario, the performance of the cloud degrades and SLA is violated. While in OP there is low consumption of resources that are allocated resulting in a higher cost for the providers. As a result, in response to dynamic changes in the global arrival rate during

*Corresponding Author.

runtime, adaptation mechanisms are coveted for polished dynamic scalability. Appropriate dynamic scalability is the demand of the time and it affirms the performance of the SLA while making the cost low. An efficient technique for dynamic scalability is required for fulfilling the requirement of both the users and CSP.

This work has proposed a dynamic scaling approach for the containerized cloud. The approach enables us to acquire the dynamic and scalable nature of cloud computing and analyze its efficaciousness. The model tries to estimate the future resource demand and provision the resources in a dynamic way for mitigating the SLA violation and reducing the cost incurred by the system. The work is simulated in Cloudsim and the work is evaluated under the various quantity of workload. The main contribution of the paper goes as:

- A queuing model is proposed to estimate and acquire the behavior of containerized DC.
- The load balancer model and container model are discussed.
- The mathematical formulation has been derived from the analytical model for various QoS measures.
- Simulation of the work is performed on Cloudsim.

The remaining paper is compiled as follows: the literature study associated with the work is done in Section II. The proposed work is discussed in Section III. Section IV performs the results and discussion and lastly, the work is concluded in Section V.

II. Related Work

Containerization is not a novice concept of computer science. It was existing back in 1972 on Linux or Unix systems in different ways [9]. It aided the developer in providing an efficacious programming environment which has a quite reduced operational cost. Docker has adopted container technology and led to the start of open containers in the industries like Google, Microsoft, and many more and it is getting popularity day by day due to its isolation strategy. Fig. 1 describes the container in the cloud system with its private OS, interface, and file system. Cloud containers provide a thin encapsulation over the application so its deployment is relatively easier and faster. Initially, it started with the VMs which are light. These technologies possessed an isolated OS on which the application can be deployed [10]. Containers have several benefits over VMs [11]. Firstly, compared to virtual machines, containers use host system resources far more efficiently. Second, starting and stopping the containers only takes a minute time. Next, the mobility container prevents inter-system dependency conflict and guarantees its separate functioning from the system on which it is hosted. Fourth, unlike VMs, which are frequently not distributed production environments, containers possess the feature of being exceedingly lightweight, enabling end users to operate dozens or more of them simultaneously. Fifth, instead of having to go through hours-long installation and configuration hassles, end users of apps can instantly download and run sophisticated software. Additionally, unlike virtual machines (VMs), which strive to virtualize an external environment, a container's primary goal is to make an application fully portable and independent [12].

The containerized cloud is emerging as one of the most challenging issues over the past few years and a lot of work has been published in this regard. This section studies the relevant work associated with the scalability and performance of container-based cloud models. Some study is associated with scalability to provide better insight into scalability and some shows the scalability in container clouds.

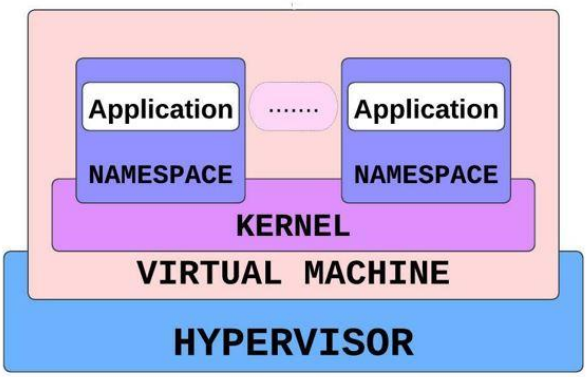

Fig. 1. Container structure.

Scalability has once been qualified as the key feature contributing to the efficient working of cloud-based services. A deep scaling methodology has been introduced in [13] where three components have been included for effective resource utilization. First, it forecasted the workload then it mapped the workload intensity to the approximated CPU utilization and lastly, an auto scale method is developed for maximizing the CPU utilization. A proactive elastic model was defined in [14] which resolved the scalability issues in cloud-based IoT systems. It utilizes the ant colony optimization technique along with the Markov chain for scheduling the resources efficiently which enhances the performance and maintains the QoS measures. It improved the response time and request throughput. A benchmarking method is proposed that defined a framework for scalability benchmarking tools for quantifying the scalability. It included the scalability metrics and measurement methods to specify the achievement of the given service level objectives. It also provided the facility for configuring the scalability parameters for getting an efficient response [15]. The author in [16] proposed a container-based autoscaling procedure that used a heuristic technique for utilizing the resources efficiently. It improved the execution time, throughput, response time, and the minimum number of containers. The author in [17] addresses the two major scalability metrics volume scalability and quality scalability. Volume scalability is highly influenced by the scaling of service volume while quality scalability is affected by the service quality provisioned. These parameters quantify the technical scalability and helped in assessing the impact of demand on the service. Besides, they also aided in designing and performing scalability testing with the motive of the identification of the components that affect the scalability performance.

Scalability in container clouds has made the processing of cloud applications lightweight and efficient. An automatic scaling method is discussed in [18] where it reduces the response time, energy consumption, and better CPU utilization. An analytical model based on the stochastic technique for the container-based DC has been discussed in [19]. It studied and analyzed the performance of the cloud system with respect to mean job delay and job rejection probability. It created a framework for container emulation and assessed the same against the suggested stochastic technique. Through experimental development, the suggested model is validated using actual data. Insight into DC planning is provided to system designers by numerical verification. Another approach is introduced in [20] in which AWS autoscaling is implemented which facilitate estimating the future workload. It applied a future prediction algorithm using Prophet API. It studied the CPU utilization and the creation of new EC2 instances when the workload is heavy. An auto-scaler-based model has been discussed in [21] which provide the architecture for the container-based application. It has included a monitoring mechanism, prediction model, time series model, and decision mechanism. The prediction utilized the time series to predict the future workload. It has provided better provisioning and speedy elasticity. The author in [22] introduced a framework for auto-scaled containerized applications which is governed by workload demand. It offered both reactive and proactive scaling. Reactive scaling was implemented using the threshold rules and proactive scaling utilized a neural network. It ensured the requirement of QoS. Another container-based module was developed in [23] which provided efficient provisioning. It used an adaptive function tree for scalable container provisioning. It mitigated the provisioning cost further by using a fetching mechanism showing the quality of on-demand and I/O efficiency. It turned out to be providing better scaling, response time, and provisioning. A horizontal scaling technique is discussed in [24] which configured the services in a docker container while the workload was balanced using the load balancer. It calibrated the infrastructures depending on the number of predicted users. It expanded the infrastructure and processing capability in a short duration and offered a fault-tolerant system for medium and small-scale industries. Another technique for resource utilization in a cloud-based application is discussed in [25] for container clouds leveraging the vertical elasticity of Docker. The resource coordinator and monitoring policies are implemented during the execution of tasks. Scalability parameters are the configurable parameters in the procedure.

To the greatest of our knowledge and as of this time, there hasn't been any research available for the effectiveness and dynamic scalability of containers published in the literature. The existing work does not consider the dynamic scalability in the containers which has provided a cost-effective solution to the virtualization. It is of utmost importance to identify the number of containers required to cope with the highly dynamic workload to satisfy the SLA and QoS requirements. Dynamic scalability is attained only when there is neither overprovisioning nor under-provisioning. Overprovisioning may result in higher costs as more containers will be engrossed while under-provisioning leads to SLA violation. Therefore, the main distinction between our study and the studies listed above is that in addition to forecasting workload, we also forecast the future need for computing resources. Furthermore, in contrast to most techniques that focus on only one factor (CPU utilization), our model provides cloud providers with more information about the timely scaling and descaling of containers' and VMs' volume. This not only decreases the cost incurred by the users but also improves the user's experience and also mitigates the financial burden of service provider and infrastructure cost due to the efficient and wise usage of the resources.

III. Proposed Work

A. Problem Formulation

A model consisting of DC consists of PMs which has the capability of holding various VMs which are further profound enough to hold various containers representing the real practical scenario of current existing cloud services. A hypervisor is held responsible for allocating various VMs to a PM while multiple containers can be allocated to a VM. The task execution request raised from the different users is being sent to the load balancer (LB). This LB routes the traffic to the PMs for execution. These requests are sent to a buffer system which is linked to the LB queue from where it is sent to the containers for the allocation of the resources and their execution. The tasks from the queue are allocated to the containers as per the availability of the resources. Whenever the user demands a new container with a particular requirement of the resources, the establishment of SLA between the final users and the CSP is agreed upon by the delivery of the requested QoS. If the breach in the agreed SLA happens the CSP is supposed to pay the penalty to consumers. The flow of the tasks happens as end users put in the request and it is sent to the LB. This LB receives the requests and distributes the tasks to the PMs as per the allocation policy utilized. Each task is allocated a unique container. As the task request increases the VM scales up the through the addition of the container for the execution of the request. Mostly the companies utilizing the features of the famous company Docker [26], use at least 18 containers simultaneously. Let us consider the m PMs which are represented as $P = \{ P_1, P_2, P_3 \dots \dots P_m\}$ which can hold a maximum of up to n VMs represented as $V = \{V_1, V_2, V_3 \dots \dots V_n\}$. A VM can contain maximum l containers represented as $C = \{C_1, C_2, C_3 \dots \dots C_l\}$. So, a PM can be scaled maximum to n VMs which can accommodate a maximum of $n \times c$ containers.

B. Queuing Model

The PMs undertaken in the DC have a similar configuration. The requests generated by the end users are sent to the queue and are served at each node in the DC based on a first come first serve basis. When the requests are executed and served well then they exit from the system. This paper has assumed that each request is served in only one container and one container will serve only one request. The DC is modeled using the open Jackson queuing model represented in Fig. 2.

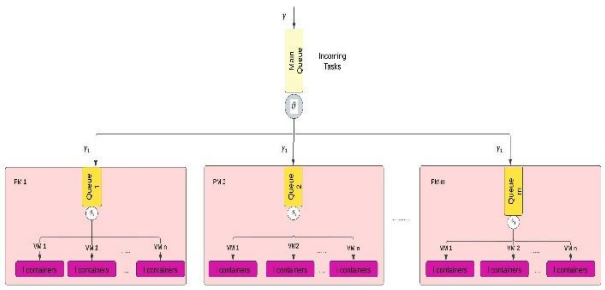

Fig. 2. Queuing model.

C. Load Balancer Model

The LB is held responsible for managing the huge load which comes with cloud computing in the form of an ample number of requests from the end users. To serve the requests well the LB is modeled as an M/M/1 queuing model having the provision of the infinite capacity task requests buffer and the arrival of requests is supposed to be one by one [27]. The Markov chain with the continuous time of the LB model, and state t depicts the task number which means $t-1$ tasks are waiting to get allocated in the queue. The arrival of the tasks happens at the γ rate similar to that of the Poisson procedure in which the arrival duration of two immediate tasks is independent and the distribution is exponential according to the rate $1/\gamma$. The serving time to the task at the PM in the LB is exponentially distributed over the ϑ rate and $1/\vartheta$ is the mean serving time. If $\alpha < 1$ the M/M/1 is assumed to be stationary, where $\alpha = \frac{\gamma}{\vartheta}$. Let the probability be π_t be for the t^{th} state. The following equations can be summated utilizing the balanced equation [28]:

$$\gamma\pi_0 = \vartheta\pi_1 \quad (1)$$

$$(\gamma + \vartheta)\pi_t = \gamma\pi_{t-1} + \vartheta\pi_{t+1}, \; t \geq 1 \quad (2)$$

From equation 1 and 2, it can be written

$$\pi_t = \alpha^t\pi_0, \; t \geq 1 \quad (3)$$

According to the normalized equation,

$$\sum_{t=0}^{\infty}\pi_t = 1 \quad (4)$$

One can deduce that,

$$\pi_0 = 1 - \alpha \quad (5)$$

And then the steady-state probability of t tasks in the queue can be given as:

$$\pi_t = (1-\alpha)\alpha^t, \quad t \geq 0 \quad (6)$$

The number of tasks on an average queued in LB can be deduced as:

$$U_{lb} = \sum_{t=0}^{\infty} t\pi_t = \alpha(1-\alpha)\sum_{t=0}^{\infty} t\alpha^{t-1} = \frac{\alpha}{1-\alpha} = \frac{\gamma}{\theta-\gamma} \quad (7)$$

The average response time that the tasks in the queue obtained can be evaluated through Little's law [29] given as:

$$R_{lb} = \frac{U}{\gamma} = \frac{1}{\theta-\gamma} \quad (8)$$

D. Container Model

The paper has considered a DC containing various PMs having various VMs designated to hold one or more container instances executing on it. The local Scheduler (LS) and the runtime component (VMs) are the two major holdings of a PM. Fig. 3 demonstrate the placement of VMs and containers and LS in a PM. The container is executed on these VMs as an isolated thread in a similar namespace with a guest OS shared among other containers in the same VM. The hypervisor performs the operations that include resource management for placing the containers in the pool of VMs in accordance with the workload being requested from the users.

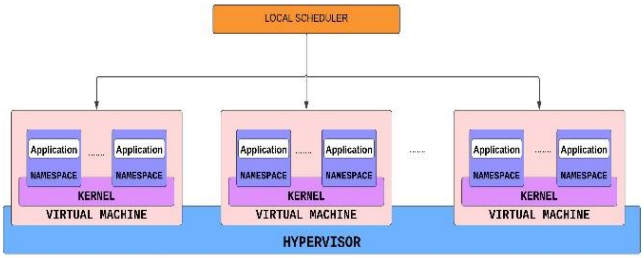

Fig. 3. PM structure.

Let's consider CDC contains m PMs with LS modeled as M/M/1/C model [30] with n VMs and l containers $\eta = n \times l$ and thus making the queue full. This implies that PM is exhausted with the resources and is not in a condition to accept any new task until it gets finished up with the tasks previously allotted to it. So, it will reject the incoming new tasks. According to Burke [31], the departure procedure in the queue M/M/1 follows the Poisson process with the same rate γ. Thus, the tasks arriving at each PM follow the Poisson rate with $\gamma_1 = {}^{\gamma}/_{m}$ and each task is served with a service time exponentially distributed with an average ${}^{1}/_{\vartheta_1}$. As the queue is finite in size so, for all the values of γ_1 and $\vartheta_1$the system is stable. The i^{th} PM with t tasks in the queue has an equilibrium probability that can be defined as:

$$\pi_i^t = \begin{cases} \frac{(1-\omega)\omega^t}{(1-\omega^{\eta+1})}, & \omega \neq 1 \\ \frac{1}{\eta+1}, & \eta \neq 1 \end{cases} \quad (9)$$

The rate at which tasks are lost at the i^{th} PM at the LS queue can be obtained as:

$$L_i = \gamma_1\pi_i^{\eta} = \gamma_1\frac{(1-\omega)\omega^{\eta}}{(1-\omega^{\eta+1})} \quad (10)$$

The LS queue has i^{th} PM whose throughput is given as:

$$N_i = \gamma_1(1-L_i) = \gamma_1\frac{1-\omega^{\eta}}{1-\omega^{\eta+1}}, \omega \neq 1 \quad (11)$$

Similarly, the volume of tasks available in i^{th} PM at the queue is:

$$D_i = \sum_t^{\eta} t\pi_i^t = \begin{cases} \frac{\omega}{1-\omega} \times \frac{1-(\eta+1)\omega^{\eta}+\eta\omega^{\eta+1}}{1-\omega^{\eta+1}}, & \omega \neq 1 \\ \frac{\eta}{2}, & \omega = 1 \end{cases} \quad (12)$$

The number of tasks undergoing the service is:

$$S_i = 1 - \frac{1-\omega}{1-\omega^{\eta+1}}, \omega \neq 1 \quad (13)$$

So, the tasks waiting in the queue can be given as:

$$W_i = D_i - S_i \quad (14)$$

The CPU utilization can be given as:

$$U = \frac{N_i}{\vartheta_1} \quad (15)$$

The waiting time for the tasks at i^{th} PM is:

$$WT_j = \frac{W_i}{\gamma_1} \quad (16)$$

Thus, the response time at i^{th} PM is evaluated as:

$$R_i = \frac{D_i}{N_i} = \frac{1}{\vartheta_1 - 1} - \frac{\eta\gamma_1^{\eta}}{\vartheta_1^{\eta+1} - \gamma_1^{\eta}}, \omega \neq 1 \quad (17)$$

For the later part of PM, each VM is modeled as the servers with l servers and the queue is not available with these servers i.e., M/M/l/l [32] where l depicts the volume of containers available with each VM. As stated in [31], each virtual machine's incoming tasks follow a Poisson process with a rate of γ_2, indicating that each container receives an equal amount of requests. Since there are n VMs so each will get the tasks with an arrival rate $\gamma_2 = {\gamma_1}/{n}$. This will provide a balance system as each VM has a similar configuration. The service rate of each container can be taken as ϑ_2. The task incoming at the VM can be visualized as a birth and death process. In a state $a < l$, the rate of the incoming task is $\gamma_a = \gamma_2$ where $\gamma_l = 0$. While in the state $a = 0,1,2 \ldots . l$, the death rate $\vartheta_a = a\gamma_2$. Let s_a be the stationary probability with a tasks in the a^{th} VM. It is observed: $\gamma_2 s_{a-1} = a\mu_2 s_a$, $(0 \leq a \leq l)$, from the local balance equation. With $\delta = \gamma_2/\vartheta_2$, it can be written as:

$$s_a = s_{a-1}\frac{\delta}{a} = s_0\frac{\delta^a}{a!}, 0 \leq a \leq l \quad (18)$$

After the application of standardization condition [33], s_0 can be generalized as:

$$s_0 = \frac{1}{\sum_{a=0}^{l}\frac{\delta^l}{l!}} \quad (19)$$

It can be further deduced:

$$s_a = \frac{\frac{\delta^a}{a!}}{\sum_{a=0}^{l}\frac{\delta^l}{l!}}, 0 \leq a \leq l \quad (20)$$

The loss probability L_a for the tasks lost at a^{th} VM, as the VM was full, is recognized as:

$$L_a = \frac{\frac{\delta^l}{l!}}{\sum_{a=0}^{l}\frac{\delta^l}{l!}} \quad (21)$$

As there is no queue for the VM, so the tasks' volume in the VM

$$D_a = \delta(1 - s_a^{loss}) \quad (22)$$

As earlier, the response time is evaluated as:

$$R_a = \frac{1}{\vartheta_2} \quad (23)$$

As it is already known, when the LB sends the request, a job can only be carried out by one PM and in one VM by a container. Thus, the response time of the tasks before they went for execution can be summed as:

$$R = R_{lb} + R_i + R_a \quad (24)$$

With a similar analysis, the task being rejected in DC is:

$$L = L_a + L_i \quad (25)$$

IV. Results and Discussion

The model proposed above is simulated through a series of experiments to analyze its effectiveness. The simulation has been performed on a personal computer with a 2.30 GHz Intel Core i3 processor and 4GB of RAM. The simulation tool used is Cloudsim. Initially, the DC is configured with 5 PMs and each PM is capable of supporting 10 VMs which varies to 50 VMs while each VM can accommodate a maximum of up to 18 containers. The arrival rate of the task varies from 1000 to 10,000 tasks per second. The task in the queue requests for execution which is serviced in 0.0001 seconds. The maximum capacity of the queue is 300. The LS service the request on an average of 0.001 seconds. The experiment is performed with 100 repetitions for efficient analysis.

A. Response Time

The response time of the system is very much affected by the volume of VMs which is analyzed with the varied task arrival rate. Fig. 4 illustrates the same. Here, the capacity of each VM to hold containers is 20. From the figure, it can be observed that the increment in response time to the task arrival is quite proportional. It is analyzed, for all the given scales of VMs, there are no substantial change in the response time when the arrival rate of the task varies from 5500 tasks per second to 9500 tasks per second. As the tasks arrival rate increases from 9500 tasks/second the response time increase exponentially for all the scales of containers. The response time 0.41second is observed in the 20 VMs scenario when the arrival rate is 10,000. The proper configuration of the VM can be chosen if the minimum response time is one of the QoS targets to be achieved for SLA.

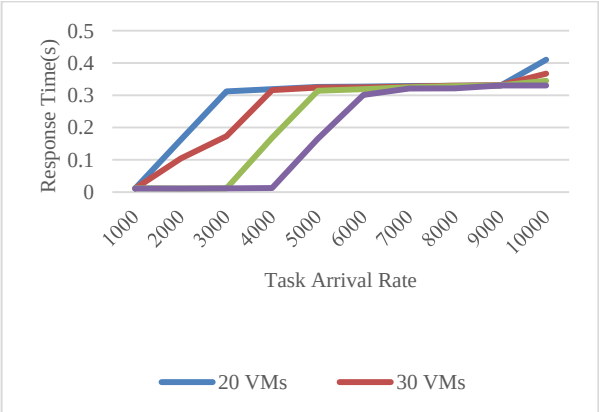

Fig. 4. Response time.

B. Drop Rate

The drop rate of the system is defined as the rate at which the tasks are dropped or rejected because of either lack of space in the LS queue or a lack of capacity in DC. Fig. 5 depicts the drop rate against the task arrival rate with the varied number of containers. Each VM has 18 containers. It can be observed from the figure that initially there is not much drop in the tasks but as the task rate increases the drop rate increases. This increase varies differently with a different configuration. In the case of 20 VMs, the loss starts after the 2000 task arrival rate is reached while in the 50 VMs case this loss starts after 5000 tasks/sec. Until the rate reaches 5000 the 50VMs configuration doesn't show any loss while the 3015 tasks/s are lost in the same task arrival rate as the 20VMs configuration. It can be deduced with the increased number of containers there is less loss of tasks.

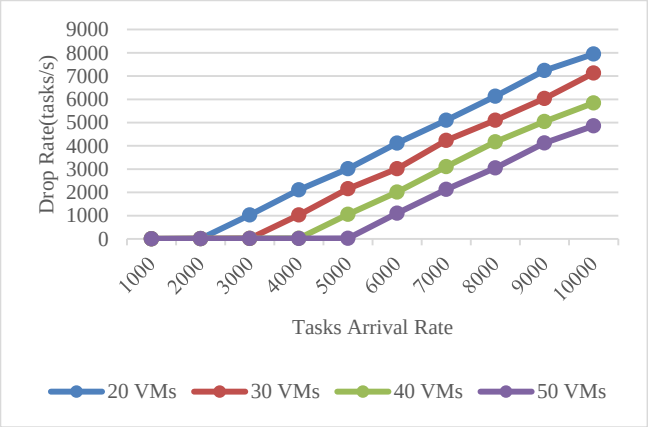

Fig. 5. Drop rate.

C. Throughput

The system throughput is being analyzed against the task arrival rate with all four configurations of the containers. Fig. 6 shows the variation of the system throughput measured in tasks per second. As it can be observed that in all four cases the system throughput is similar till 2000 tasks/sec. The impact of the different configurations of containers can be seen beyond 2000 tasks/sec. The system performs better with a large number of containers. There is not much variation that can be seen when the rate of the task reaches 2000 tasks/sec in the first case, 3000 tasks/sec in the second case, 4000 tasks/sec in the third case, and 5000 tasks/sec in the last case. After a certain threshold, the throughput has become quite fixed. The requirement of predefined system throughput in SLA can be resolved using the selection of the best configuration of containers by the service providers.

D. Number of Containers

To study the effects of the number of containers on response time and drop rate the tasks arrival rate has been fixed at 9000 tasks/sec. The number of containers has been increased from 8 to 18 containers. Table I represents the response time. It is observed from the table as the number of containers increases the response time decreases. A dramatic decrease can be seen after the 16th container. The response time of the system is highly dependent on the volume of containers. The system drop rate is also getting highly influenced by the number of containers. It can be analyzed that as the containers increase the drop rate decreases. It shows that to keep the drop rate below 3010 tasks/sec the minimum number of VMs and containers is 50 and 8 respectively.

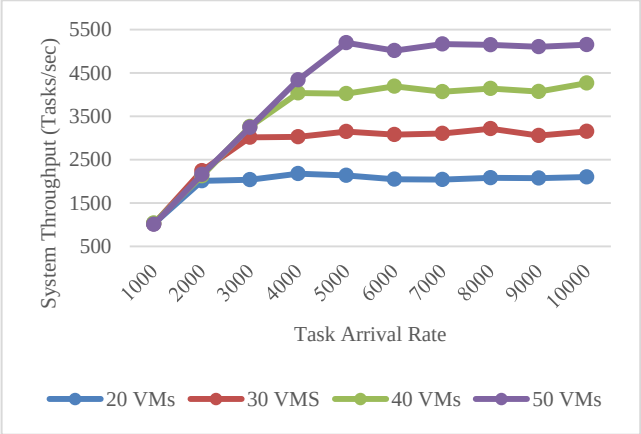

Fig. 6. System throughput.

TABLE I. SYSTEM RESPONSE TIME AND DROP RATE WITH RESPECT TO THE NUMBER OF CONTAINERS

No. of	System Response Time				System Drop Rate			
Containers	*20*	*30*	*40*	*50*	*20*	*30*	*40*	*50*
8	0.317	0.311	0.293	0.284	6135	5147	4087	3010
9	0.316	0.308	0.289	0.282	5948	5084	3942	2985
10	0.3149	0.304	0.286	0.2815	5827	4972	3875	2875
11	0.315	0.3037	0.288	0.281	5773	4864	3751	2870
12	0.3157	0.3021	0.287	0.2807	5648	4784	3617	2861
13	0.3154	0.2998	0.285	0.2794	5584	4743	3561	2756
14	0.3148	0.2994	0.283	0.279	5538	4476	3548	2641
15	0.297	0.287	0.277	0.274	5416	4507	3472	2571
16	0.283	0.278	0.264	0.258	5386	4459	3378	2468
17	0.257	0.246	0.238	0.2334	5258	4319	3307	2402
18	0.226	0.218	0.210	0.204	5156	4238	3193	2354

E. Comparison with other Algorithms

The response time and system drop rate are compared with the existing work [11] and [16] for the 50VMs with 18 containers each. From Fig. 7, it can be observed that till the 4000 tasks/sec there is not much variation among the algorithms. As the rate increases the proposed shows better results. With 10000 tasks/sec, the response rate is 0.3301sec of the proposed algorithm while that of [11] is 0.524sec and that of [11a] is 0.601 sec. From Fig. 8., it can be deduced that till the task rate is 4000 all the algorithms show the same drop rate but as the task rate increases there is an exponential increment in the drop rate. With 10000 tasks/sec, the drop rate of the proposed algorithm is 4859 tasks/sec and while that of others is 5338 and 5812 respectively. The suggested method is significantly more effective than others.

The results obtained above have demonstrated that the increment in task arrival rate affects the QoS measures depending on the containers available in the DC. So, it's very essential to scale up or scale down the container instances depending upon the rate of the incoming task. In addition to this, the number of containers available has to fulfill the SLA requirements. Besides, in the DC the workload is very dynamic and to provision, the minimum containers dynamically which can fully satisfy the SLA requisite which monitors the usage of virtual resources and modify the number of resources to be used is of utmost importance. Therefore, the main challenge that needs to be worked upon is the engagement of the minimal number of containers for fulfilling the SLA exigencies. Allocation of a greater number of containers than required may lead to the OP which increases the cost. Deploying a lesser volume of containers than expected may result in UP leading to more SLA violations. Therefore, dynamic scalability is the requirement of the time to avoid the situation of under and over-provisioning. The proposed algorithm facilitates the service provider to identify the minimal containers required while the rate of the task fluctuates helping in scaling up and down the resources and maintaining the SLA.

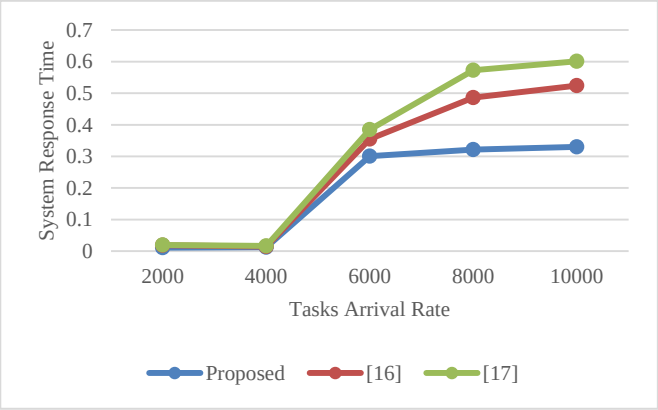

Fig. 7. Comparative analysis of system response rate.

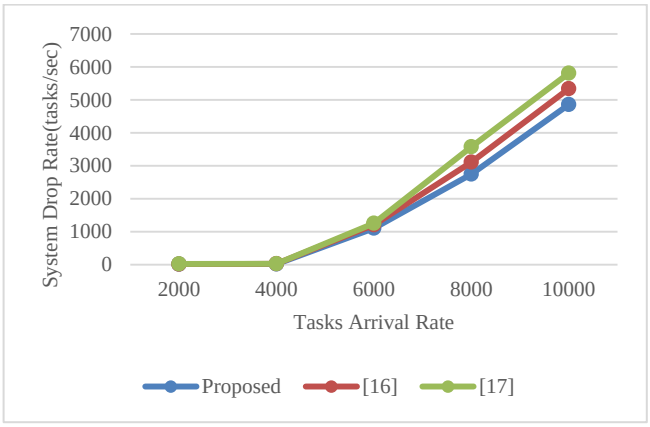

Fig. 8. Comparative analysis of system drop rate.

Further, the resources are allocated to the tasks in the order of their arrival. It may not consider the priority tasks which can be handled in further study. Since the arrival rate of the tasks is considered fixed which may differ in real life scenario as the arrival rate can vary with the state and thus making the potential customer to switch other service due to long waiting queue and thus it may affect the efficiency of the system.

V. Conclusion

This paper has proposed a queuing model for dynamic scalability in containerized clouds to analyze the workload and the effects of scaling on the QoS parameters. It also suggests the number of containers is scaled up or down for the requirement of a particular given SLA. A mathematical model is developed for identifying the key performance metrics. The model predicts and approximates the resource request for future requirements to mitigate the SLA violations and provide cost-effective solutions. The proposed methodology can also be used to scale the DC containers to guarantee the QoS parameters. The model is proficient enough in deciding the number of containers required for the provision or deprovisioned as per the given workload situation to meet up the SLA demand and QoS metrics. The proposed model is tested against some existing work and has turned out to be performing better. In future work, the model can be implemented in a real working environment and more SLA parameters can be included for the analysis. Further, clustering technique can be included and the model can have queue classified according to the requirements of the tasks like some tasks may require more processing units while some require more storage unit.

References

[1] Amini Motlagh, A., Movaghar, A., & Rahmani, A. M., "Task scheduling mechanisms in cloud computing: A systematic review," International Journal of Communication Systems, vol. 33, no. 6, pp. e4302, 2020. https://doi.org/10.1002/dac.4302.

[2] V. Eramo, F. G. Lavacca, T. Catena, and P.J. Perez Salazar, "Proposal and investigation of an artificial intelligence (AI)-based cloud resource allocation algorithm in network function virtualization architectures," Future Internet, vol. 12, no. 11, pp. 196, 2020. https://doi.org/10.3390/fi12110196.

[3] A. Bhardwaj, and C. R. Krishna, "Virtualization in cloud computing: Moving from hypervisor to containerization—a survey," Arabian Journal for Science and Engineering, vol. 46, no. 9, pp. 8585-8601, 2021. https://doi.org/10.1007/s13369-021-05553-3.

[4] B. Varghese, R. Buyya, "Next generation cloud computing: new trends and research directions," Future Gener. Computer Syst. vol. 79, pp. 849–861, 2018. https://doi.org/10.1016/j.future.2017.09.020.

[5] H. Khazaei, C. Barna, N. Beigi-Mohammadi, M. Litoiu, "Efficiency analysis of provisioning microservices," International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, pp. 261–268, 2016. https://doi.org/10.1109/CloudCom.2016.0051.

[6] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," Journal of Systems and Software, vol. 150, pp. 77-97, 2019.

[7] Y. Saadi, S. El Kafhali, "Energy-efcient strategy for virtual machine consolidation in cloud environment," Soft. Comput, vol.24, no.19, pp. 14845-14859, 2020. https://doi.org/10.1016/j.jss.2019.01.001.

[8] I. Kabashkin, "Availability of applications in container-based cloud PaaS architecture," In International Conference on Reliability and Statistics in Transportation and Communication, Springer, pp. 241-248, 2018. https://doi.org/10.1007/978-3-030-12450-2_22.

[9] J.P. Martin, A. Kandasamy, K. Chandrasekaran, "Exploring the support for high performance applications in the container runtime environment," Human-Centric Comput. Inf. Sci., vol. 8, no.1, pp. 1–15, 2018. https://doi.org/10.1186/s13673-017-0124-3.

[10] L.Cai, Y. Qi, W. Wei, J. Li, "Improving resource usages of containers through auto-tuning container resource parameters," IEEE Access, vol. 7, pp. 108530–108541, 2019. https://doi.org/10.1109/ACCESS.2019.2927279.

[11] A. Bhardwaj, and C. R. Krishna, "Virtualization in cloud computing: Moving from hypervisor to containerization—a survey" Arabian Journal for Science and Engineering, vol. 46, no. 9, pp. 8585-8601, 2021. https://doi.org/10.1007/s13369-021-05553-3.

[12] B. Tan, H. Ma, Y. Mei, and M. Zhang, "A cooperative coevolution genetic programming hyper-heuristic approach for on-line resource allocation in container-based clouds," IEEE Transactions on Cloud Computing, vol.10, no.3, pp. 1500-1514, 2020. https://doi.org/10.1109/TCC.2020.3026338.

[13] Z. Wang, S. Zhu, J. Li, W. Jiang, K. K. Ramakrishnan, Y. Zheng, and A. X. Liu, "DeepScaling: microservices autoscaling for stable CPU utilization in large scale cloud systems," In Proceedings of the 13th Symposium on Cloud Computing, pp. 16-30, 2022. https://doi.org/10.1145/3542929.3563469.

[14] N. Nithiyanandam, M. Rajesh, R. Sitharthan, D. Shanmuga Sundar, K. Vengatesan, and K. Madurakavi, K., "Optimization of Performance and Scalability Measures across Cloud Based IoT Applications with Efficient Scheduling Approach," International Journal of Wireless Information Networks, vol. 29, no. 4, pp. 442-453, 2022. https://doi.org/10.1007/s10776-022-00568-5.

[15] S. Henning, and W. Hasselbring, "A configurable method for benchmarking scalability of cloud-native applications," Empirical Software Engineering, vol. 27, no. 6, pp. 1-42, 2022. https://doi.org/10.1007/s10664-022-10162-1.

[16] S. N. Srirama, M. Adhikari, and S. Paul, "Application deployment using containers with auto-scaling for microservices in cloud environment" Journal of Network and Computer Applications, vol.160, pp. 102629-102641, 2022. https://doi.org/10.1016/j.jnca.2020.102629.

[17] A. Al-Said Ahmad, and P. Andras, "Cloud-based software services delivery from the perspective of scalability," International Journal of Parallel, Emergent and Distributed Systems, vol. 36, no. 2, pp. 53-68, 2021. https://doi.org/10.1080/17445760.2019.1617864.

[18] C. Li, J. Liu, B. Lu, and Y. Luo, "Cost-aware automatic scaling and workload-aware replica management for edge-cloud environment," Journal of Network and Computer Applications, vol. 180, pp. 103017, 2021. https://doi.org/10.1016/j.jnca.2021.103017.

[19] B. Liu, Y. Chen, "A scalable fine-grained analytic model for container cloud data centres," Int. J. Internet Technol. Secur. Trans., vol. 9, no. 4, pp. 355–389, 2019. https://doi.org/10.1504/IJITST.2019.102794.

[20] N. Nithiyanandam, M. Rajesh, R. Sitharthan, D. Shanmuga Sundar, K. Vengatesan, and K. Madurakavi, K., "Optimization of Performance and Scalability Measures across Cloud Based IoT Applications with Efficient Scheduling Approach," International Journal of Wireless Information Networks, vol. 29, no. 4, pp. 442-453, 2022. https://doi.org/10.1007/s10776-022-00568-5.

[21] M. Imdoukh, I. Ahmad, M. G. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," Neural Computing and Applications, vol. 32, no. 13, pp. 9745-9760, 2020. https://doi.org/10.1007/s00521-019-04507-z.

[22] S. Chouliaras, and S. Sotiriadis, "Auto-scaling containerized cloud applications: A workload-driven approach," Simulation Modelling Practice and Theory, vol. 121, pp. 102654, 2022. https://doi.org/10.1016/j.simpat.2022.102654.

[23] A. Wang, S. Chang, H. Tian, H. Wang, H. Yang, H. Li, and Y. Cheng, "{FaaSNet}: Scalable and Fast Provisioning of Custom Serverless Container Runtimes at Alibaba Cloud Function Compute," USENIX Annual Technical Conference (USENIX ATC 21), pp. 443-457, 2021.

[24] D. Perri, M. Simonetti, S. Tasso, F. Ragni, and O. Gervasi, "Implementing a scalable and elastic computing environment based on cloud containers," International Conference on Computational Science and Its Applications, Springer, pp. 676-689, 2021. https://doi.org/10.1007/978-3-030-86653-2_49.

[25] J. Y. Choi, M. Cho, and J. S. Kim, "Employing Vertical Elasticity for Efficient Big Data Processing in Container-Based Cloud Environments," Applied Sciences, vol. 11, no. 13, pp. 6200, 2021. https://doi.org/10.3390/app11136200.

[26] I. Kabashkin, "Availability of applications in container-based cloud PaaS architecture," International Conference on Reliability and Statistics in Transportation and Communication, Springer, pp. 241–248, 2018. https://doi.org/10.1007/978-3-030-12450-2_22.

[27] S. El Kafhali, K. Salah, "Performance modeling and analysis of internet of things enabled healthcare monitoring systems," IET Netw, vol. 8, no. 1, pp. 48–58, 2019. https://doi.org/10.1049/iet-net.2018.5067.

[28] H. Chen, D. D. Yao, "Fundamentals of Queueing Networks: Performance, Asymptotics, and Optimization," vol. 46. Springer, Berlin ,2013. https://doi.org/10.1007/978-1-4757-5301-1.

[29] R. Nelson, "Probability, Stochastic Processes, and Queueing Theory: the Mathematics of Computer Performance Modeling," Springer, Berlin 2013. https://doi.org/10.1007/978-1-4757-2426-4.

[30] K. Salah, S. El Kafhali, "Performance modeling and analysis of hypoexponential network servers," J. Telecommun. Syst., vol. 65, no. 4, pp. 717–728, 2017. https://doi.org/10.1007/s11235-016-0262-3.

[31] Burke, "P.J.: The output of a queuing system," Oper. Res., vol. 4, no. 6, pp. 699–704, 1956. https://doi.org/10.1287/opre.4.6.699.

[32] S. El Kafhal, K. Salah, S. Ben Alla, "Performance evaluation of IoT-fog-cloud deployment for healthcare services," International Conference on Cloud Computing Technologies and Applications (CloudTech'18), IEEE, pp. 1–6 ,2018. https://doi.org/10.1109/CloudTech.2018.8713355.

[33] U. N. Bhat, "An Introduction to Queueing Theory: Modeling and Analysis in Applications, Springer, New York ,2015. https://doi.org/10.1007/978-0-8176-4725-4.