Int. J. Nonlinear Anal. Appl. 13 (2022) 2, 1893–1905
ISSN: 2008-6822 (electronic)
http://dx.doi.org/10.22075/ijnaa.2022.27403.3588

On optimizing scalability and availability of cloud based software services using scale rate limiting algorithm

V. L. Padma Latha[a,*], N. Sudhakar Reddy[b], A. Suresh Babu[c]

[a]*Department of Computer Science and Engineering, Sri Venkateswara College of Engineering, Tirupati, JNTUA University, Ananthapuramu, India*

[b]*Department of Computer Science and Engineerig, Sri Venkateswara College of Engineering, Tirupati, India*

[c]*Department of CSE, JNTUA College of Engineering , JNTUA University Anantapuramu, India*

(Communicated by Javad Vahidi)

Abstract

In this paper, the scale rate-limiting algorithmic approach namely the Token bucket algorithm is utilized to optimize the performance of cloud based software services. In distributed applications, higher availability and scalability have long been a critical challenge. In cloud computing, offering highly accessible services is critical for retaining client satisfaction, confidence and avoiding revenue losses. The gateway Zuul is considered as the entryway to various services in the Spring Cloud based software service must perform the rate-limiting process and make sure the service's reliability in the event of excessive scalability. The token bucket rate-limiting technique cannot ensure core service availability. To address this issue, this study developed an overload protection technique depending on a URI configuration file in conjunction with the Zuul gateway that may filter requests before obtaining tokens. The token bucket rate-limiting algorithm is implemented the traffic limitation function and ensured the cloud platform service's reliability and availability. The estimation of scalability, as well as availability, demonstrates the level of service supplied to consumers in response to their requests. The elasticity measures are used to assess the cloud based software services performance in terms of scalability. In the future, cloud computing improvements and expansion might increase cloud-based software services.

Keywords: Performance optimization, scalability, availability, Cloud based software services, Rate-limiting, Token bucket algorithm
2020 MSC: 90C31

1 Introduction

Cloud computing produces convenient, ubiquitous, on-demand network access to a shared pool of configuration-based computing resources such as networks, applications, storage, servers, and services which are rapidly provisioned and published with limited communication or service provider communication [30]. Cloud computing is rapidly growing, but it is already having a big influence in a variety of fields. More businesses are moving to the cloud, and significant technological advancements are occurring. Organizations, companies and small companies alike can benefit

*Corresponding author
Email addresses: vlpadmalatha.svit@gmail.com (V. L. Padma Latha), principal@svce.edu.in (N. Sudhakar Reddy), asureshjntu@gmail.com (A. Suresh Babu)

Received: January 2022 *Accepted:* March 2022

from cloud computing. Many businesses may save money by utilizing cloud-based infrastructure. There are three main approaches to classify cloud services namely Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) where the cloud users utilize personalized apps.

As infrastructure costs have decreased and computer resources are highly affordable and effective, cloud-based apps are fast-growing in popularity. It is critical to include scalability performance evaluation into the enhancement process to optimize the scalability and availability of any software program. It might lay a solid basis for future optimizations and also ensure that cloud services meet Service Level Agreement (SLA) requirements [28, 2]. Scalability and availability are two common performance characteristics connected with cloud based software services [4].

Some of the most pressing issues in cloud services include high availability (HA) as well as dependability. The availability of a network at time 't' is known as the likelihood that the system will be up and running appropriately at that point of time [31]. Regarding cloud services, high availability (HA) is critical for retaining customer confidence and avoiding revenue losses owing to service level agreement (SLA) breach fines [12]. Cloud computing platforms have received much interest in recent decades from worldwide businesses and government organizations for their ability to handle essential mission systems [13]. Moreover, in cloud service, scalability and high availability are increasingly becoming a big concern.

Scalability refers to the cloud layer's potential to improve the efficiency of software service provision by increasing the number of software services available. Elasticity refers to the cloud layer's ability to adjust autonomously in response to changing requests for service software. Performance is a metric that compares the number of software services offered for execution to the amount of demand for such services. Additionally, efficiency includes an economic benefit focused on methodologies for understanding the effects of key performance elements in a cloud-based service [11]. The potential to analyze the cloud based software services performance relies heavily on technical scalability measures and validation [8].

One of the primary aspects of the cloud environment that encourages cloud solutions is scalability. In the cloud, increased scalability is often accomplished through increasing hardware resources or enhancing network infrastructure. Since it is a key characteristic of cloud computing, it implies that when additional resources are provided, the cloud's capability to manage rising data volumes and processing performance improves. The key motivations for people to utilize cloud services are elasticity and scalability [5]. The software program may be scaled up in a variety of ways to achieve high flexibility.

The term rate-limiting indicates the termination of an operational frequency from reaching a certain range. Rate limiting is often employed in large-scale systems to preserve fundamental programs and services. Rate limitation is typically implemented as a protective mechanism for services. To preserve service availability and to achieve high scalability then the shared services must defend themselves against excessive use if planned or unplanned. Also, scalable networks have consumption restrictions in place at some point. Users must be developed with rate limitations in mind for the system to work properly and avoid cascade failure. For increasing throughput and decreasing end-to-end delay over huge distribution systems, rate limiting on either user or server sides is critical. The most typical purpose for rate limitation is to reduce resource hunger, which improves the availability of API applications. In most cases, rate limiting is applied before the restricted resource along with a safety buffer for prior notice. As there is a delay in loads, the margin is necessary; therefore rate limiting must be in position before serious conflict for a resource occurs. A RESTful API, for illustration, may use rate-limiting to safeguard an underlying database, and an API service that is scalable using a rate limiter may make a huge number of simultaneous requests to the database, and this database is unable to deliver unambiguous rate-limiting signals.

The scalable rate limiting is achieved using the various algorithmic approaches namely leaky bucket algorithm, token bucket algorithm, fixed window, sliding log, and sliding window. The cloud based software services make use of a token bucket algorithm for rate-limiting to achieve high effective performance via the optimization of scalability and availability for the user's request. In this paper, the token bucket algorithm is now employed for the rate-limiting function in the spring cloud based software services mostly via the Zuul gateway component. The main issues are that it is regardless of the actual request URI, the demand is directly deleted or re-forwarded if the threshold load is surpassed. This traffic management mechanism is overly simplistic, and it may cause important APIs such as payment to be ignored while non-core APIs are often used. This has an impact on the user's experience as well as the interactions among key services in the cloud software. An overload protection approach depending on URI configuration files, the token bucket algorithm with a Zuul gateway is developed to optimize the performance scalability. The fundamental notion is that if a request comes at the gateway, it locates and filters the actual URI before retrieving the token and evaluates if the demand necessitates a current-limit action. Thus, the proposed algorithmic approach optimizes the scalability and availability in terms of response time.

2 LITERATURE REVIEW

Scalability performance analysis and evaluation for cloud software applications are highlighted as important research issues and approaches in related studies [24, 14]. Some other mapping research [1] found that most cloud based software analysis studies report preliminary findings, indicating increased interest in the topic and the opportunity for far more investigation to follow up on the preliminary findings.

Cloud performance evaluations and measurements in terms of scale, flexibility, and efficacy are addressed. The majorities of publications concentrate on elasticity; note that the publications are preliminary results or first thoughts of research respondents regarding scalability. The important performance factors elasticity, scalability, and efficiency are used in this work [26]. Other latest surveys [16, 20] have focused particularly on the elasticity of cloud services.

A great deal of research [18, 6, 7, 25, 23] emphasis quantifying the cloud based service elasticity from a technological standpoint. Several essential elements are provided that may be used to measure the elasticity of cloud service concerning time magnitudes and quantity for periods when service delivery is either under or over the required service demand. The sharing times and mean time durations in under and over-provisioned provinces are elasticity measurements described by [19]. The inverse value of multiplication of the mean under-provisioned/over-provisioned period and mean an absence of resources define the up and down-elasticity measures. Other elements and methods, including functions of resource inaccuracy, reconfiguration time, and scalability, are incorporated in a later development that expanded the previous metrics.

The contribution of [22, 21] on assessing and estimating scalability from a utilization-oriented approach is noteworthy. The performance meter from a performance aspect, the manufacturing scaling metric incorporates the assessment of a quality-of-service (QoS) as well as the costs of such service. This technique is valuable from a usability standpoint, but because it is dependent on various aspects of the system with cost metrics, it is unlikely to give relevant and detailed information regarding system component contributions to scalability from a performance perspective. Software as a service (SaaS) is assessed in terms of performance and scalability from standpoint of system capacity, employing load requirements and potential as scalability metrics [15]. Some other recent study [27] involves creating a model which can be used to evaluate and assess the capacity, costs, and flexibility of various deployment configurations. The two scalability measurements: one concerning the relation among the capability of cloud-based software services and their usage of cloud resources replacing cost instead of cloud resources using the cost scalability measurement function [10]. To illustrate the performance measures, the Cloud Store apps are organized on Amazon EC2 with various configurations. A theoretical architecture for the scalability of portable multi-agent systems is presented, although it is still restricted to theory and experimental findings [33].

For decades, businesses have tried to preserve and safeguard data to secure their client's personal information. Cloud computing was created by businesses as a method to give safe storage as well as the processing of data with the capacity to businesses and people. Cloud storage is used by numerous businesses in a variety of industries [29]. Cloud computing is a type of dynamic computing type for various applications and also the storage that makes use of Internet technologies. To enable self-service, resource pooling, accessing of wide network, quick elasticity, and measurable service are five main features of cloud computing [3]. In general, there are 3 main categories of cloud computational services namely Infrastructure as a Service, Software as a Service, and Platform as a Service. Moreover, cloud computing may be used in four distinct ways namely hybrid, public, community, and private cloud.

The benefits of cloud computing include increased computational power, storage, adaptability, scalability, and lower IT equipment overhead costs [3]. Startups can benefit from migrating to the cloud by redirecting capital expenditures into operational expenditures, providing cloud computing appealing when IT resources are being slashed. The smallest businesses mostly use cloud computing, while medium-sized businesses possess lower rates, while organizations with less than 100 participants possess the lowest rates [32]. Larger companies have ample computational power within them. But still, cloud computing has several drawbacks [9], such as the need for Internet connectivity, speed, and direct access resources. As a result, businesses may find it dangerous to rely only on cloud computing services. Any disruption in cloud services might be disastrous for businesses [17]. The key benefits are cost savings, privacy, security, and dependability. Stakeholders believe that in the future, those challenges with cloud computing adoption will be minimized or removed.

3 System model

The efficient cloud-based service framework is primarily built with different forms of optimization levels including Software-as-a-Services (SaaS), Platform-as-a-Services (PaaS), and Infrastructure-as-a-Services (IaaS). The layer's objective is improving service to the users in terms of scalability, and availability optimization performance. Because of abstraction generated by the SaaS as well as virtualization idea, the types and numbers of models for predicting optimization and driving optimization performance of cloud services have been constrained. The cloud based services are shown in Figure 1.

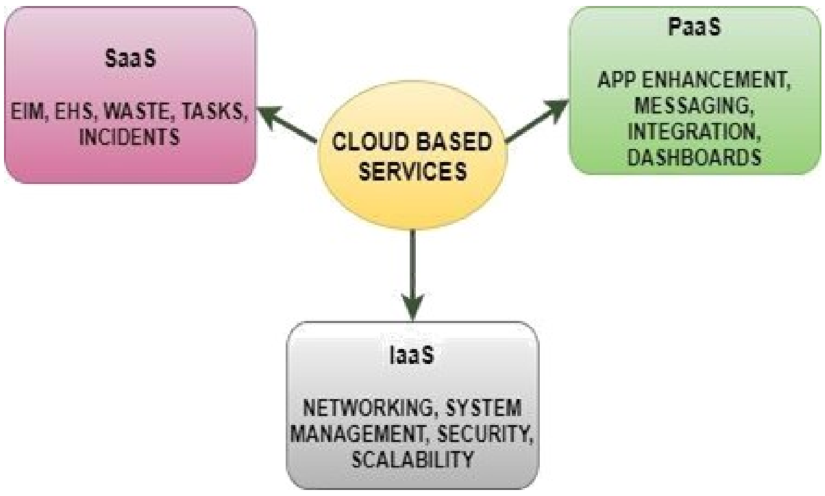
SaaS
EIM, EHS, WASTE, TASKS, INCIDENTS
PaaS
APP ENHANCEMENT, MESSAGING, INTEGRATION, DASHBOARDS
CLOUD BASED SERVICES
IaaS
NETWORKING, SYSTEM MANAGEMENT, SECURITY, SCALABILITY

Figure 1: Cloud-based services.

Software-as-a-Services (SaaS): Software as a service is a method of delivering applications as a service through the internet. The users are allowed to access the program through the internet rather than downloading it on their machine. It liberates the user from having to deal with complicated hardware and software. Hardware or software is not needed to be purchased, maintained, or upgraded by SaaS users. The only need is that the user has internet access, after which the program is very simple to use. Google Apps, Microsoft Office 365, and other similar services are examples.

Platform-as-a-Services (PaaS): PaaS provides users with an application enhancement or platform as a service, allowing them to install their customized software and code. Customers are free to build their unique apps which can operate on the cloud infrastructure. To gain the administration capability of the applications, an item is considered as the service providers produce a preset configuration of system operation and server application. Platform-as-a-Service is used in the applications such as Ruby, J2EE, and LAMP including Linux, Apache, PHP, and MySQL.

Infrastructure-as-a-Services (IaaS): IaaS provides numerous computer resources as a service, including network, storage, storage devices, functional system, and hardware. IaaS customers use a Wide Area Network (WAN) where the access services are performed through the internet facility. For instance, by logging onto the IaaS platform, the users can construct virtual computers.

Token Bucket Algorithm

Cloud tasks are fully managed service which lets users control the dispatch, execution, and delivery of many dispersed tasks. Users may asynchronously do work outside as per the request utilizing cloud asks and it guarantees services that are scalable and available to consumers. The token bucket methodology is used by cloud tasks to permit some speed improvements in how items and data are distributed within certain restrictions.

The token bucket method is part of the scale rate-limiting algorithm, which optimizes the performance of scalability and availability of cloud-based software applications. The token bucket algorithm is shown in Figure 2. The token bucket approach is used to construct scalability and availability limiting tool of class Rate Limiter in cloud based software open source Guava toolset. The token bucket technique may be used by the Zuul gateway to provide an optimal, scalable, and accessible cloud service to consumers because the configuration is already incorporated in the Spring Cloud. The token bucket method works on the following principle:

1. At the r mean rate per second, the bucket collects the dropped tokens within it.
2. The bucket is capable of collecting tokens up to a maximum limit of b tokens. The freshly put token should be removed when the bucket is overflowing.
3. This takes n tokens and passes the packet to the network whenever it receives a packet of n-byte.
4. When the tokens present in the bucket are low in comparison with n tokens demanded by the packet, then the bucket with such tokens is not removed, and the packet is marked as violating the constraint.

When allocating tokens on a requesting packet, the typical token bucket technique will not filter the demand. It simply determines if there are sufficient tokens in the bucket to distribute. When the bucket's tokens are inadequate, all demands which fulfill the criterion are ignored, including certain fundamental API calls like purchase payment requests, affecting the stability as well as the quality of service. The token bucket approach is improved using a gateway zuul. The entering requests are intercepted using gateway zuul for waiting tokens. URI configuration is introduced for the file interception method is responsible for intercepting the request before the token is blocked. All of the essential URIs are contained in the configuration file.

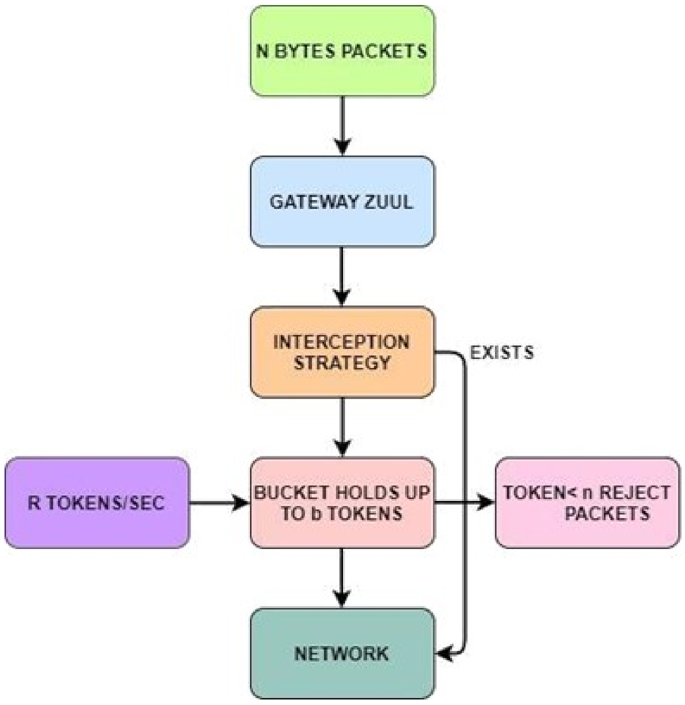

Figure 2: Token bucket algorithm.

The interception strategy checks if the demand URI is in a specific configuration file once the demand packet is received at the gateway Zuul. When it exists, this not demands a token for traffic limitation and provides direct service accessing. It must acquire the tokens when it does not available. The present limiting operation is executed once the threshold is surpassed. The token bucket technique is used to develop an interception technique depending on a URI configuration file. To access the core API, you require a customized URI configuration file. Similarly, the Zuul Filter class is applied in the Zuul gateway via the filter class that includes four generic functions specified by Zuul Filter. The generic functions of the ZuulFilter class are as follows:

Filter Type: The filter type is identified using the string return. The operational processes are limited using a limit filter the demand before it is forwarded.

Filter Order: The performance execution of the filter is depicted as an integer value. As this Limit Filter class is the very first filtering to be performed, it returns 0.

Should Filter: If the filter undergoes the execution process, then the function returns a Boolean type is involved in constructing the filter switch. If the limit filter is performed, it returns the true value.

Run: This function implements the filter's unique logic, as well as the customized code logic.

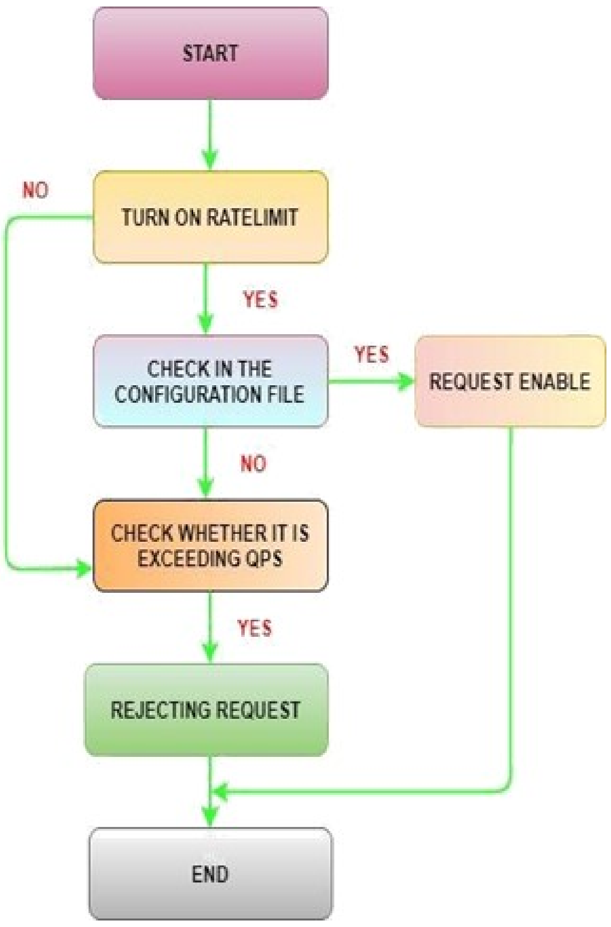

Figure 3: Flowchart of rate-limiting algorithm.

Figure 3 shows the rate-limiting algorithm. Set the starting rate value for a rate limiter instance. Get the latest request URI, retrieve the configuration file, as well as check it in the list of URI configurations. If it is found in the configuration file, it is not affected by the rate value or may be accessed directly. If users don't, you'll have to be constrained by the rate values and the rate limit is used to get the tokens. The threshold is surpassed, hence the request is dropped whether the tokens are not retrieved within the 10ms delay period and during the delay period the tokens are retrieved, and the present rate value is inside of the security range, allowing request accessing.

Users may use cloud functions to construct single and distinct functions which react to cloud activities without having to maintain a server environment. Cloud Functions are asynchronous and extremely scalable to accommodate incoming request traffic, cloud-based software service managed infrastructure launches feature instances dynamically. Functions are targeting the high rates of demands as a result of scalability behavior, and when such functions access downstream services and become a source of inadvertent DoS. Communication exhaust on datasets is one sort of DoS. When each function case makes a dataset communication to a backend, a traffic increase may cause several cases to automatically scale up, therefore depleting dataset server communication capacity. The service includes a functional max cases option to restrict functions against scaling far beyond a specific number of situations. Cloud calls the function with events payloads and contexts for idle functions. When the function breaks or not capable to handle the event—for example, as it's been rate-limited by some downstream may specify that the service to continue event delivery. While the max instances parameter can assist the scalability limit and therefore, not provide direct control for the number of times the function is called per second.

Pub/Sub is responsible for managing the contemporaneous messaging service which involves sending and receiving messages across different apps in real-time. When transporting a large volume of information over Pub/Sub items, the rates adjustment has to be made at which messages are handled at consumption users so that users operating in parallel are beneficial and don't store so much unresolved messages, causing total processing speed to increase. Numerous flow control parameters are exposed by Pub/Sub users to alter this performance. Cloud Run is a controlled computing platform for running stateless applications which can be invoked via HTTP requests. With Cloud Functions, instances can handle many requests at the same time provided the container's service stacks allow it to use a single container instance.

Open-source software such as Istio is a self-contained service mesh that offers the foundations for running a distributed cloud-based software services. Because a cloud-based software necessitates services that are protective towards rogue neighbor services and therefore, it includes rate limiter in the network mesh. Cloud gateways are an API-managed services solution that allows users to encrypt, analyze, monitor, and create quotas for their APIs utilizing a similar infrastructure as APIs. It allows you to establish your quotas, comprising rate-based rules, as a service meant to enable companies to expose services to the outside world. Cloud uses worldwide infrastructures and security technologies to defend against Distributed Denial of Service (DDoS) attacks. This contains built-in logic to safeguard users protected services from malicious surges and increased rate loads.

Scalability and availability performance:

In cloud applications, scalability refers to a software service's potential to raise the efficiency of delivering services by increasing the cloud software services as per the requirement over the duration of time. Our primary issue is that the network can achieve performance optimization in terms of scalability as demand dictates over a longer duration of service supply, based on a specific demand scenario. Users are unconcerned with the management of resources in a short-term flexible manner. Elasticity's goal is to meet service supply with the real quantity of necessary resources during any given time. Moreover, unconcerned with the performance of cloud based software services provision is often evaluated by the number of resources such as power as well as the cost is necessary to perform the intended task.

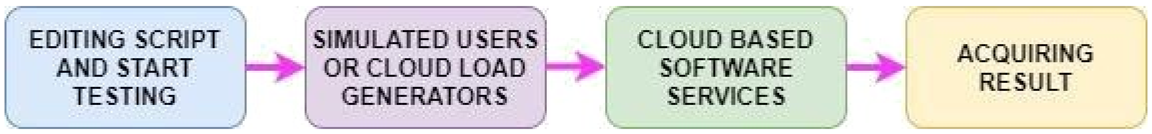
EDITING SCRIPT AND START TESTING
SIMULATED USERS OR CLOUD LOAD GENERATORS
CLOUD BASED SOFTWARE SERVICES
ACQUIRING RESULT

Figure 4: Testing of scalability performance.

Cloud availability is often increased by enhancing or lowering the service requirement volume being provided by a single instance or many instances of service software, or via the association of both methods. Assuming that by increasing scale-up service, proportionally increases cloud based software service demand will not affect performance efficiency. The service quality can be measured in terms of reaction time. In the cloud, the optimal scaling characteristic of the service system must be considered over an appropriate longer duration. If the platform is not scaled optimally then there is an increment in service volume while maintaining the same level of performance. Real systems are often anticipated to function underneath the ideal scalability rate, and the goal of scalability analysis and evaluations is to define how much real system performance varies from ideal performance. The scalability testing process is shown in Figure 4.

To achieve the optimal scalability performance, we anticipate that the system has to enhance the software instances count in relation to the growth in software services demand i.e., if demand doubles, the basic software instances should likewise double. Thus, the system has to sustain service quality via maintenance of similar mean response time regardless of the number of customer demands, i.e., when demand grew about 25%, then there would be no rise in mean response time. The volume of the service being considered is represented as D and D′, where D′ being greater than D. The service providing software instances are being represented as I and I′. The mean response times are represented as $\mathrm{t_r}$ and $\mathrm{t'_r}$. When the system grows optimally then the service demand level D and D′ are explained as follows

$$\frac{D\prime}{D} = \frac{I\prime}{I} \tag{3.1}$$

$$t_r = \ t_r\prime. \tag{3.2}$$

The above equations state that the number of software instances delivering the service grows in lockstep with the service demand and irrespective of the degree of service demand, the performance optimization remains unaltered in regards to mean response time. By acquiring the disparity between ideal and real system scalability performance, measuring technical scalability measurements for cloud-based software services with the ideal scalability performance is possible. Scaling is insufficient in regards to providing software situations for service delivery when the real instances count is less in comparison with ideally expected scaling instances count. To determine the degree of deficit, a demand case is chosen and begins with a reduced rate of feature demand D_0, then calculate the software instances I0. Evaluating how near the I_k values are to the ideal I_K^* values by continuing the software instances I_k correlating to the number (n) of growing demand levels D_k with a similar demand scenario. Considering equation (3.1), for ideal I_K^* values corresponding ideal scalability are as follows:

$$I_K^* = \left(\frac{D_k}{D_0}\right) I_0 \tag{3.3}$$

Let us assume the ratio among the considered area as $(D_k, I_k\,,\,)$ values in which the K$- = 0, \ldots .$n. and the system scalability η_1 service volume of (D_k, I_k^*) an area as follows

$$A^* = \sum_{k=1,\ldots.n} (D_k,\ \ D_{k-1},\ \ ) \ .\ \left(I_K^*,\ I_{K-1}^*\ \right) \ \ /\ 2 \tag{3.4}$$

$$A^* = \sum_{k=1,\ldots.n} (D_k,\ \ D_{k-1},\ \ ) \ .\ (I_k,\ ,\ I_{k-1}\ \ ) \ \ /\ 2 \tag{3.5}$$

$$\eta_1 = \frac{A}{A^*} \tag{3.6}$$

Here, A* and A are the representation of ideal and real I value of areas and η_1 is the scalability 1ηperformance parameter of the service's volume. When η_1 is near to 1, the system is near to the 1ηoptimal volume scalability. When the contrary is true as well as η_1 is near to 0, therefore the scalability system's volume is far from ideal.

The service means response times t_k proportional to the demand levels D_k are measured to acquire a high scalability performance. The system means response time is defined as the time consumed by the system to handle a request after it has been received. The ideal condition of equation (3.2) is to estimate the ideal $t0$ mean response time. When the mean response time increases then correspondingly there is a decrement in scalability quality i.e. $tk > t0$. Let us assume the ratio among the considered area as (D_k, t_k) values in which $k = 0, \ldots, n$ and considered (D_k, t_0) area ratio specifies a measure of scalability based quality system service η_t :

$$B^* = \sum_{k=1,\ldots..n} (D_k,\ \ D_{k-1},\ \ ) \ \ .\ \ \ t_0 = \ \ (D_n - D_0)\,.\ \ \ t_0 \tag{3.7}$$

$$B^* = \sum_{k=1,\ldots..n} (D_k,\ \ D_{k-1},\ \ ) \ \ .\,(t_{k,} +\ t_{k-1},\ \ )\ /\ \ 2 \tag{3.8}$$

$$\eta_t = \frac{B}{B^*}. \tag{3.9}$$

Here, the ideal and true t values of areas are denoted as B^* and B. The system performance η_t scalability is denoted as η_t. The system is considered as ideal scalability when the η_t value is nearer to 1 else for the value nearer to 0 , it is considered to be far away from the ideal system.

The scalability measurement can be used to assess the performance scalability of cloud-based software services Additional utility considerations like non-technical quality features and cost have no bearing on such measurements and enable us to focus scalability tests which attempt to identify communication protocols that have a significant influence on performance scalability correspondingly impact of any system modification. Thus, performance scalability is optimized in cloud based system services which relate to the quality and service volume metrics. Availability is also necessary to meet the requirement of the users in cloud based software services. The determination of the most essential elements can affect system high availability by defining cloud-based services component. Various controllers such as storage, node, cluster, cloud, Virtual Machine (VM), and Walrus are the essential components. The four important approaches in achieving a highly accessible, dependable cloud system and the performance of cloud computing nodes are addressed. To provide high availability for diverse demands and receiving workloads based on the specified task classification. So, when cloud services tasks are being classified based on their available resources (Memory, CPU, etc.), higher availability of cloud services tasks may be given via hosting tasks in appropriate clusters to minimize task failure because of resource constraints. A memory-intensive job will be routed to a cluster with sufficient memory resources. To provide this assurance, scalable infrastructure is required, which includes several computer clusters shared by multiple workloads with varying requirements for availability, dependability, throughput, and latency. A resource utilization task such as memory utilization, CPU cycles, as well as storage facilities is necessary to deliver needed service levels with high availability. Availability assessment is categorized as instantaneous availability, average uptime availability, steady-state availability, and operational availability.

Depending on the changes, the system is operational at any specified period is defined as instantaneous availability. This definition is related to the reliability function which determines the likelihood that a system can operate at a particular time t. The service will be effective based on the instantaneous availability of the requirements are met: The desired functionality can be appropriately given with probability R(t) throughout time t since final repair period, it has been able to give the needed function, such as u, 0<u<t.

$$\int_0^t R\left(t-u\right) m\left(u\right) du \tag{3.10}$$

Here the renewal density function of a system is denoted as m(u). The expression for instantaneous availability is given as follows,

$$A\left(t\right) - R\left(t\right) + \int_0^t R\left(t-u\right)\ m\left(u\right) du \tag{3.11}$$

The percent of the period in which the system is available for usage is known as mean availability or average uptime availability. This type of availability defines the instantaneous availability's average value across a specific period.

$$A_m\left(T\right) = \frac{1}{T}\int_0^T A\left(t\right) dt \tag{3.12}$$

By evaluating the time approaches infinity limits of instantaneous availability result in steady state availability. The time approaches 4 times the MTBF of instantaneous availability is considered to be approaching the steady-state value is expressed as follows,

$$A\left(\infty\right) = \log_{r\to\infty} A(T) \tag{3.13}$$

Operational availability is an essential metric for assessing system effectiveness and performance. It assesses the software's availability that covers all types of unavailability, like diagnostic, administrative, and logistical downtime. The operational availability is determined by a formula as follows

$$A_0 = \frac{uptime}{operating\ cycle} \tag{3.14}$$

4 RESULT AND DISCUSSION

The proposed token bucket algorithm is used to optimize the performance of cloud based software services in terms of scalability and availability. To manipulate multi-threaded concurrent queries, this paper utilizes an Apache ab command. The limiting threshold for the token bucket algorithm is fixed at 10. The test case generates a similar number of concurrent requests as the services concern. The classic token bucket approach and the enhanced token bucket method are used to evaluate the multiple users. A testing procedure is done to acquire the reaction time within the interface path of various situations.

Table 1: Interface response time.

TOTAL NUMBER OF REQUESTS		**100**	**100**	**200**
CURRENT CONCURRENCY		**3**	**10**	**10**
PROPOSED TOKEN BUCKET ALGORITHM	**eurekaclient/test 1**	0.506s	2.127s	0.387s
	eurekaclient/test 2	8.789s	14.786s	21.987s

The interface functions of eureka client /test 1 as the core API in the event of a similar concurrent request and presents limitation settings, as shown in table 1. The scalability and availability are measured in terms of response time in the proposed token bucket rate-limiting algorithm are significantly faster. Eureka client/test2 interface is not the main API, the rate limit operations must be conducted using proposed methods, resulting in a quick response time. The rate-limit interception model depending on the URI configuration file processes the core interface efficiently; maintain the availability of the core service along with non-core API rate limiting operation during high concurrent requests.

The objective is to evaluate the scalability and availability of cloud-based apps using various cloud resources, configuration options, and demand situations to see how they function. In two separate cloud computing, we used the same experimental parameters for similar cloud-based systems such as OrangeHRM in EC2 and Azure. We updated the settings for Mediawiki, which operates on an AWS EC2 case. The various hardware and software configuration settings and workload generators to assess the scalability of the two cases in cloud-based software services deployed in EC2. The scaling and availability performance for the various demands size is investigated in both OrangeHRM and Mediawiki. The average number of MediaWiki and OrangeHRM users is initially evaluated to obtain the demand size of the users. Figures 5 and Figure 6 illustrate the average reaction times of OrangeHRM in EC2 and Azure and for four demand workload degrees. Figure 7 depicts MediaWiki's mean response times for four workload demands.

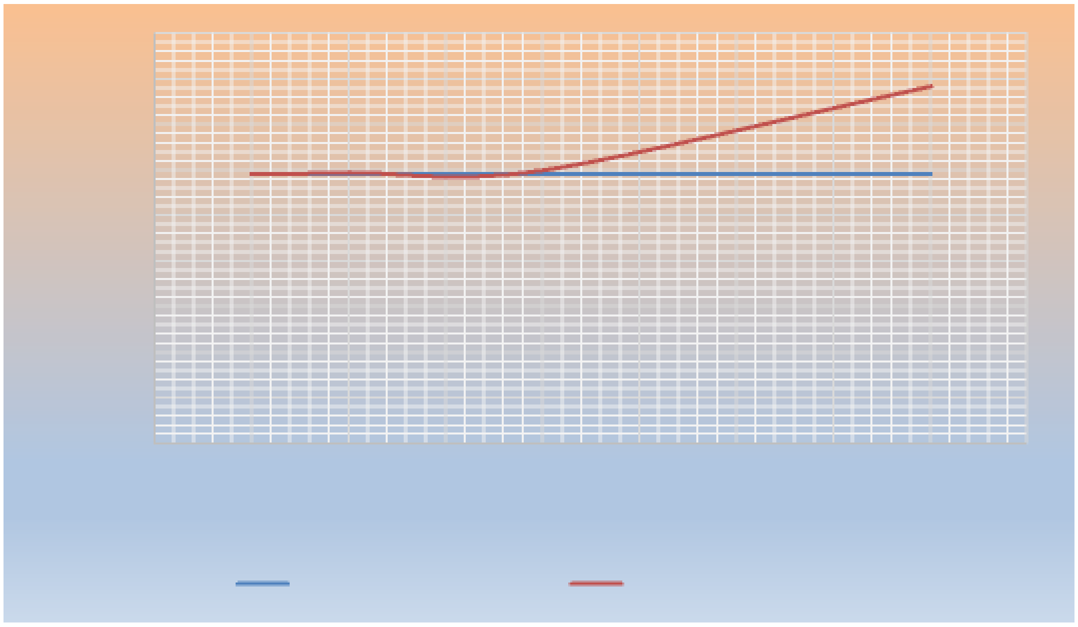

Figure 5: Average response time of OrangeHRM in EC2 for various demand sizes

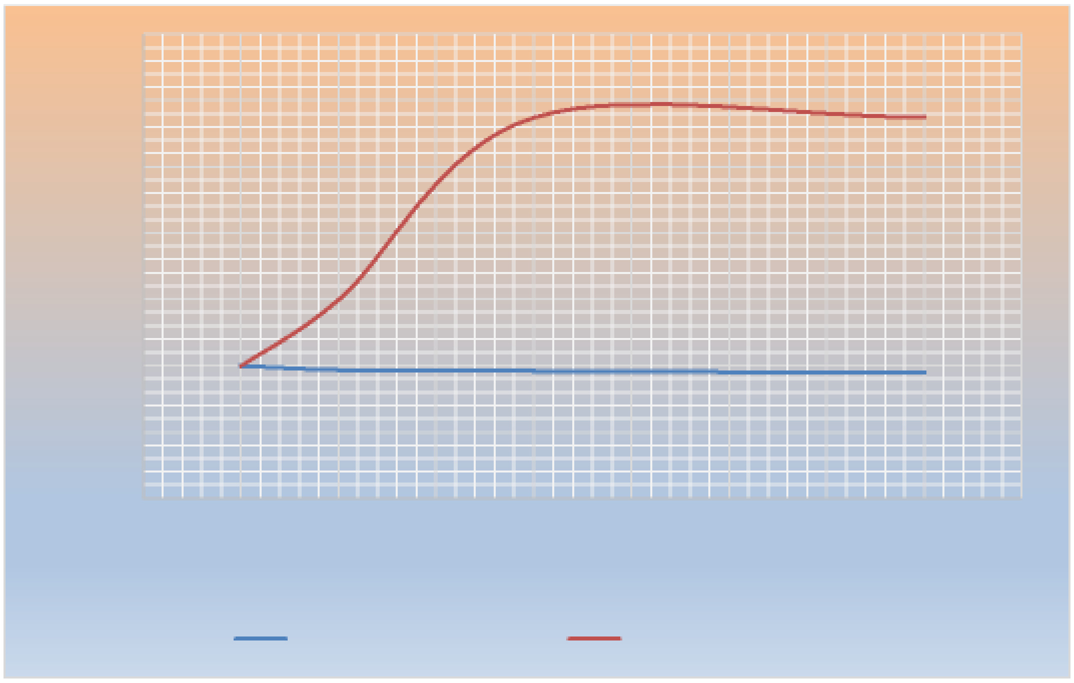

Figure 6: Average response time of OrangeHRM in Azure for various demand sizes

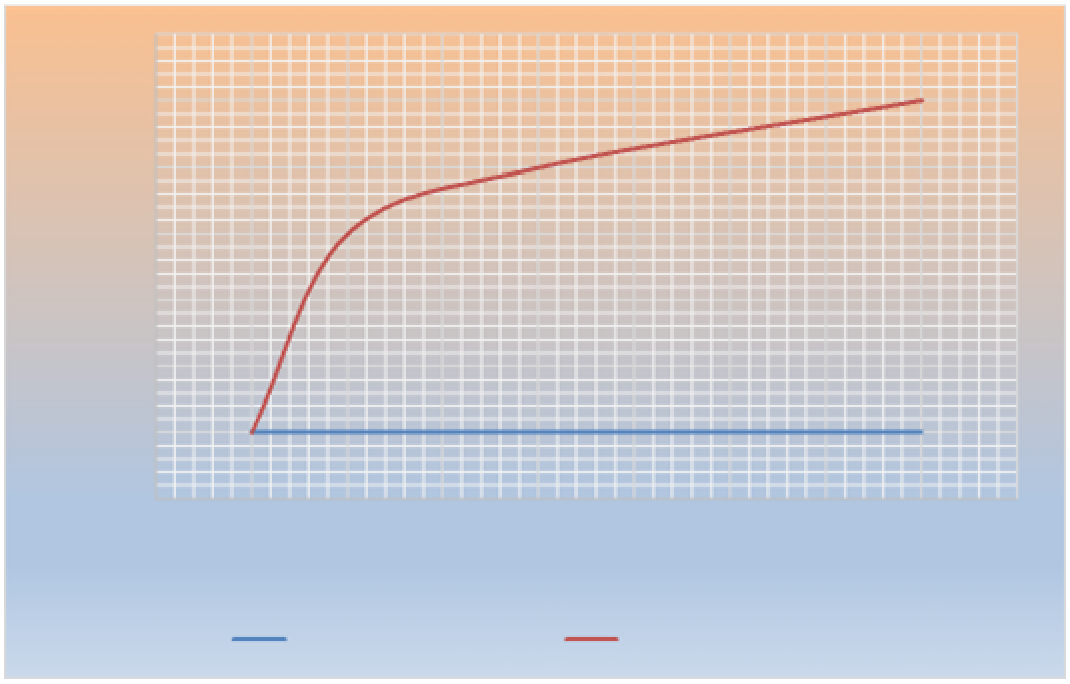

Figure 7: Average response time of MediaWiki in EC2 for various demand sizes

5 CONCLUSION

In this paper, the proposed token bucket rate-limiting algorithm uses the Zuul gateway to filtering the packet depending on the URI configuration file. The experimental results display that the proposed token bucket algorithm achieves a high scalability and availability performance during the core interface thereby providing the service to the users using the rate-limiting process for the non-core API. It's critical to stay away from unusual traffic and harmful attempts. We also observe the scalability and availability performance of the proposed algorithm in cloud based software services operating on similar and various cloud platforms using technical scalability measurements. The measurement's basic concepts are theoretically extremely basic, and they cover quality scaling performance, with discrepancies among the ideal and real scaling carves. The cloud-based software services namely MediaWiki and OrangeHRM are examined in terms of response time for the various demand sizes. Thus, the proposed token bucket algorithm optimizes the scalability and availability of cloud services.

References

[1] A. Al-Said Ahmad, P. Brereton and P. Andras, *A systematic mapping study of empirical studies on software cloud testing methods*, IEEE Int. Conf. Software Quality, Reliability and Security Companion (QRS-C), IEEE, 2017, pp. 555–562.

[2] T. Atmaca, T. Begin, A. Brandwajn and H. Castel-Taleb, *Performance evaluation of cloud computing centers with general arrivals and service*, IEEE Trans. Parallel Distrib. Syst. **27** (2016), 2341—2348.

[3] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski and M. Zaharia, *A view of cloud computing*, Commun. ACM **53** (2010), no. 4, 50—58.

[4] M. Becker, S. Lehrig and S. Becker, *Systematically deriving quality metrics for cloud computing systems*, Proc. 6th ACM/SPEC Int. Conf. Perform. Engin. ICPE '15. ACM, New York, 2015, pp. 169–1-74.

[5] H. Ballani, P. Costa, T. Karagiannis and A. Rowstron, *Towards predictable datacenter networks*, Proceedings of the ACM SIGCOMM 2011 Conf., 2011, pp. 242–253.

[6] A. Bauer, N. Herbst and S. Kounev, *Design and evaluation of a proactive, application-aware auto-scaler*, Proc. 8th ACM/SPEC Int. Conf. Perform. Engin., New York, 2017, pp. 425—428.

[7] M. Beltran, *Defining elasticity metric for cloud computing environments*, Proc. 9th EAI Int. Conf. Perform. Eva. Methodol. Tools, ICST, Brussels, 2016, pp. 172-–179.

[8] K. Blokland, J. Mengerink and M. Pol, *Testing cloud services: how to test SaaS, PaaS & IaaS*, Rocky Nook, Inc., 2013.

[9] N. Bloom and N. Pierri, *Cloud computing is helping smaller, newer firms compete*, Harvard Bus. Rev. **94** (2018), no. 4.

[10] G. Brataas, N. Herbst, S. Ivansek and J. Polutnik, *Scalability analysis of cloud software services*, Proc. IEEE Int. Conf. Autonomic Comput. ICAC, 2017, pp. 285—292.

[11] R. Buyya, R. Ranjan and R.M. Calheiros, *Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services*, International Conference on Algorithms and Architectures for Parallel Processing. Springer, Berlin, Heidelberg, 2010.

[12] J. Dantas, R. Matos, J. Araujo and P. Maciel, *Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud*, Comput. **97** (2015), 1121-–1140.

[13] M. Gagnaire, F. Diaz, C. Coti, C. Cerin, K. Shiozaki, Y. Xu, P. Delort, J.P. Smets, J. Le Lous, S. Lubiarz and P. Leclerc, *Downtime statistics of current cloud solutions*, International Working Group on Cloud Computing Resiliency, Tech. Rep. 2012, pp. 176—189.

[14] J. Gao, X. Bai, W.T. Tsai and T. Uehara, *SaaS testing on clouds - issues, challenges, and needs*, Proc. IEEE 7th Int. Symp. Service-Oriented Syst. Engin., SOSE, 2013, pp. 409-–415.

[15] J. Gao, P. Pattabhiraman, X. Bai and W.T. Tsai, *SaaS performance and scalability evaluation in clouds*, Proc. 6th IEEE Int. Symp. Service-Oriented Syst. Engin. SOSE 2011, IEEE, Irvine, 2011, pp. 61—71.

[16] N. Geetha and M.S. Anbarasi, *Ontology in cloud computing: A survey*, Int. J. Appl. Eng. Res. **10** (2015), no. 23, 43373—43377.

[17] K. Grigoriou, G. Retana and F.T. Rothaermel, *IBM (in 2010) and the emerging cloud-computing industry*, Harvard Bus. Rev. 2012.

[18] N.R. Herbst, S.Kounev, A. Weber and H. Groenda, *BUNGEE: an elasticity benchmark for self-adaptive IaaS cloud environments*, IEEE/ACM 10th Int. Symp. Software Engin. Adaptive and Self-Managing Syst., IEEE, 2015, pp. 46—56.

[19] N.R. Herbst, S. Kounev and R. Reussner, *Elasticity in cloud computing: what it is, and what it is not*, 10th Int. Conf. Autonomic Comput. (ICAC 13), San Jose, 2013, pp. 23—27.

[20] Y. Hu, B. Deng, F. Peng, B. Hong, Y. Zhang and D. Wang, *A survey on evaluating elasticity of cloud computing platform*, World Automation Congress (WAC). IEEE, 2016, pp. 1—4.

[21] K. Hwang, X. Bai, Y. Shi, M. Li, W.G. Chen and Y. Wu, *Cloud performance modeling with benchmark evaluation of elastic scaling strategies*, IEEE Trans. Parallel Distrib. Syst. 27 (2015), no. 1, 130—143.

[22] K. Hwang, Y. Shi and X. Bai, *Scale-out vs. scale-up techniques for cloud performance and productivity*, IEEE 6th Int. Conf. Cloud Comput. Technol. Sci., IEEE, 2014, pp. 763—768.

[23] A. Ilyushkin, A. Ali-Eldin, N. Herbst, A.V. Papadopoulos, B. Ghit, D. Epema and A. Iosup, *An experimental performance evaluation of autoscaling policies for complex workflows*, Proc. 8th ACM/SPEC Int. Conf. Perform. Engin., New York, 2017, pp. 75–86.

[24] B. Jennings and R. Stadler, *Resource Management in Clouds: survey and research challenges*, J. Network Syst. Manag. **23** (2015), no. 3, 567–619.

[25] J. Kuhlenkamp, M. Klems and O. Röss, *Benchmarking scalability and elasticity of distributed database systems*, Proc. VLDB Endow **7** (2014), 1219—1230.

[26] S. Lehrig, H. Eikerling and S. Becker, *Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics*, Proc. 11th Int. ACM SIGSOFT Conf. Quality of Software Architectures, 2015, pp. 83-–92.

[27] S. Lehrig, R. Sanders, G. Brataas, M. Cecowski, S. Ivanšek and J. Polutnik, *CloudStore—towards scalability,*

elasticity, and efficiency benchmarking and analysis in cloud computing, Future Gen. Comput. Syst. **78** (2018), 115—126.

[28] H.H. Liu, *Software performance and scalability: A quantitative approach*, Wiley, Hoboken, 2011.

[29] J. Mei, K. Li and K. Li, *Customer-satisfaction-aware optimal multiserver configuration for profit maximization in cloud computing*, IEEE Trans. Sustain. Comput. **2** (2017), no. 1, 17–29.

[30] P. Mell and T. Grance, *The NIST definition of cloud computing*, NIST Special Publication, 2011.

[31] M.L. Shooman, *Reliability of computer systems and networks*, Wiley, Hoboken, 2002.

[32] V. Rajaraman, *Cloud computing*, Resonance **19** (2014), 242-–258.

[33] M. Woodside, *Scalability metrics and analysis of Mobile agent systems*, Workshop Infrast. Scalable Multi-Agent Syst. Int. Conf. Autonomous Agents, Springer, Berlin, Heidelberg, 2001, pp. 234—245.