International Journal of Engineering and Science Invention (IJESI)
ISSN (Online): 2319-6734, ISSN (Print): 2319-6726
www.ijesi.org ||Volume 12 Issue 9 September 2023 || PP 49-57

Google Cloud Services Based On Service Level Agreement & Quality of Services

BILAL AHMED[1], DR. PRASADU PEDDI[2]
Research Scholar, Shri Jagdishprasad Jhabarmal Tibrewala University, Jhunjhunu, Rajasthan
Professor, Dept. of CSE & IT, Shri Jagdishprasad Jhabarmal Tibrewala University, Jhunjhunu, Rajasthan

Abstract: *Cloud computing has revolutionized IT architecture with its scalable, adaptable, and affordable solutions. Google Cloud Services (GCS) offers a range of services, including computing, storage, networking, AI, and ML. To ensure optimum service delivery, Google Cloud uses strong Quality of Service (QoS) measurements and a well-defined Service Level Agreement (SLA). This article examines Google Cloud's SLA system, analyzing its main parts such as service credits, latency considerations, performance benchmarks, and uptime assurances. It also explores how quality of service factors like availability, dependability, scalability, and security are maintained. Google Cloud's SLAs are crucial for its dependability, as they outline rules for service availability, performance objectives, and compensation mechanisms in case of service failures. Customer satisfaction is ensured by service credit policies, response time assurances, and uptime agreements. Google Cloud uses rigorous QoS procedures to guarantee scalability, security, and high availability. To improve system stability, Google Cloud uses security mechanisms, load balancing, and multiple data centers. This report offers advice for businesses looking to optimize their cloud infrastructure, maximize productivity, and minimize downtime. By understanding these factors, enterprises can make informed decisions about cloud adoption and risk management, and ensure compliance with Google Cloud's SLA conditions.*

Keywords: *Cloud Computing, Edge Computing, Virtualization, Server less-based AI applications*

I. INTRODUCTION

Edge computing is a method of data processing that uses network nodes like base stations and access points instead of traditional data centers. This approach reduces network latency by placing computers in close proximity to mobile devices, such as at access points or base stations. The expansion of edge computing has great promise for businesses and systems, with applications such as object recognition, intrusion detection, augmented reality, autonomous driving, and providing storage and computing resources on a massive scale with minimal delay. Server-less edge computing offers various advantages over other network edge designs, allowing users to focus on creating and running functions instead of managing resources. This research focuses on a server-less edge computing architecture for artificial intelligence applications. Edge computing offers several benefits, including reduced latency, improved system stability, and reduced energy and bandwidth costs compared to traditional cloud computing systems. It also allows for more flexible edge computing environments, as edge nodes can come from anywhere and operate on any platform. Virtualization technologies make the service compatible with a broad variety of devices and systems, allowing applications to run in numerous edge computing environments.

Isolation is a practical choice for improving system stability, as many applications on the network's perimeter use containers, compact, standardized software units that include the packages and environments required to execute the service code. Isolation safeguards other containers on the same host in case of a single container failure, preventing the possibility of the host computer going down due to a single container failure. System software is essential for applications to access hardware resources, and virtualization is crucial for highly scalable edge computing. As virtualization becomes more prevalent, the need for server-side resource sharing to handle requests from multiple users increases. The four mainstays of edge computing server architectures are classical, container, server-less, and virtual machine designs, each catering to a different degree of virtualization and resource sharing. In virtual machine-based edge architecture, several virtual machines (VMs) may share the hardware resources of a single server, but their operating systems remain unchanged. This allows peripheral applications on the network to operate autonomously since they use their own virtual machines.

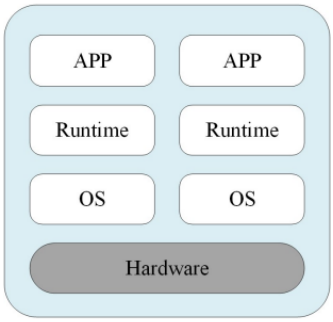

Figure 1: Level 1: Virtual machines

Via VM solutions, the relative advantages of virtualization technology—including platform freedom and isolation—are efficiently realized. Although virtual computers have made great strides, installing a whole operating system is still a major hassle when using one. The approach that uses virtual machines needs a lot of software and hardware. Poorly equipped devices will not be able to make proper use of it. One virtual machine architecture that can multitask in real-time is Gabriel, and it's available for usage in cognitive wearable apps [5]. This is an example of architecture for computing at the edge that relies on virtual machines.

Incorporating Warehouse Area Containers provide a lightweight virtualization solution that may be used in edge computing architectures. Every container shares the same set of physical components and operating system. Containers for software only include the bare minimum of OS components and application code required to run a single programmer [6]. This result in less hardware requirements compared to virtual machines. This makes the deployment of new services considerably quicker as compared to alternatives that rely on virtual machines. The Kilda architecture was an early attempt to use container-based edge computing for speech recognition [7]. It enabled faster automatic speech recognition and could accommodate several users at once.

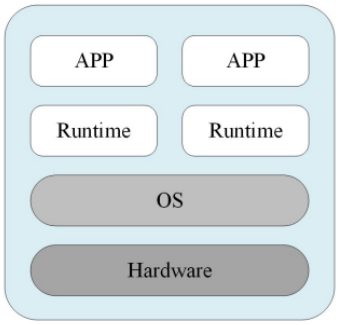

Figure 2: Level 2: Containers

Server less computing is an approach focused on the edge, allowing for shared container runtime and prioritizing client-side service applications. This platform offers a virtualization-based execution environment, with developers responsible for handling server less activities independently. The primary area of study is AI apps developed using server less edge computing.

Object detecting software is crucial in today's digital world, as devices need to take pictures or record videos before sending them to an object recognition server. Edge computing allows for distributed object identification in real-time by using the processing power and data storage of adjacent devices and infrastructures. For networks with one stage, YOLO is the best option, while faster-RCNN is the gold standard for networks with two stages. The YOLOv5-based object identification application outperforms rival RCNN algorithms on inference challenges.

Virtual reality systems can enhance user experience by superimposing digital content on the real environment, allowing users and places to engage in three-dimensional, real-time interaction. However, it takes more time and power for mobile devices to run augmented reality algorithms. Edge computing real-time access and reduced latency may radically alter this industry. Autonomous driving requires real-time, low-latency processing capability to handle massive volumes of data. Implementing stateless functions eliminates the need for servers, allowing multiple user requests to be processed in parallel. Developers are only charged per hour for computationally expensive tasks, and users are not billed for resources until the function is executed. Server less computing also allows for the execution of stateless services and resource management. The three main parts of a server less architecture are the worker, master, and edge. Worker nodes carry out tasks and code, while master

nodes handle resource management and event distribution. The server less platform is easy to use if followed instructions.

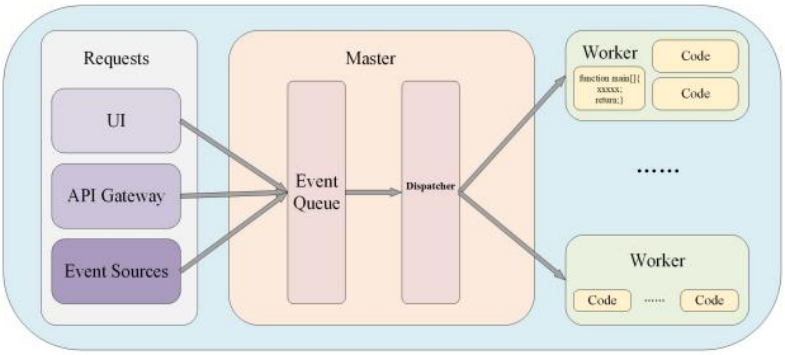

Figure 3: Server less platform architecture

Server less-based AI applications

Academics are very interested in the potential benefits of using AI into server less edge computing. The authors provided a method for developing and deploying AI apps to the server less edge in [21]. This platform streamlines the process of developing and managing AI workflow operations at the edge. According to [22], the first suggestion for the Wear Mask AI software to identify COVID-19 face masks was to use server less edge computing. This server less face mask identification programmer is compatible with standard mobile and desktop computers. The main objective of [23] was to provide server less services for processing movies in order to cut expenses and boost efficiency. For this project, our team decided to construct an object detection app using a server less architecture.

Studies in relevant domains include process modeling for server less architecture. There was a proposal for a consistent method of representing server less application processes in [24]. Predicting the end-to-end response time also warrants more research. The authors' recommendation to formally describe the server less approach would be a huge boon to our project. We may enhance the system model by using queuing theory, drawing on our knowledge of service process modeling in server less architecture. Built models in two distinct modeling languages for use in edge computing tasks [25]. Although server less architecture was not the main focus of the essay, the part about modeling in edge computing architecture was very useful.

There are many existing frameworks from which to choose when implementing the server less architecture. Tubeless, Apache Open Whisk, Open FaaS, and Native are four free servers fewer frameworks mentioned in the authors' evaluation [26]. You may install any of these four free frameworks using a Kubernetes cluster. The flexibility of Kubernetes, the leading edge computing platform, in creating and implementing complicated solutions has attracted developers from many walks of life. Since this architecture is server less, it also makes advantage of Kubernetes and Docker. Built on top of Open Whisk, the server less architecture described in reference [27] works with many other frameworks. Recording server less architecture's latency, throughput, and scalability was their main focus. In these particular cases, we choose to use the YOLOv5 method for object recognition. The YOLOv5 technology has been used by researchers to create a variety of object detecting applications. The authors of [28] used the YOLOv5 method to develop and distribute software that can detect cow activity over many networks. The authors' evaluation of various computing infrastructures makes the use of server less edge computing architecture in this project very evident. If you want to know how to include YOLOv5 into your project, read [29]. The authors used YOLOv5 to create a face mask recognition system.

Approaches and Methodologies

Evaluating the performance of AI applications requires the deployment of the server less edge computing architecture. Once the server less architecture is up and running, the object detection application may be started. Following the three steps of service delivery—arrival, waiting, and serving—is the next logical step. After deciding to conduct a system performance study, the next step is to administer a series of tests. The efficiency of the system and some ideas for improvements based on queuing theory make up the last part of our examination.

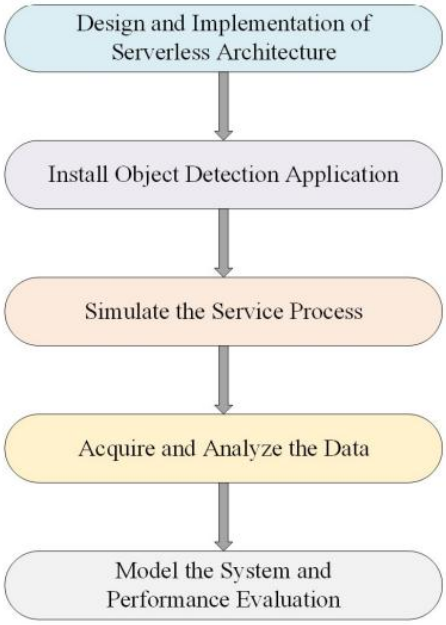

Figure 4: Research Process

The data analysis technique used in this study involves quantitative approaches for analyzing time and confidence scores, while qualitative analysis is used to evaluate item detection accuracy. Quantitative approaches involve running service simulations with different confidence score criteria to assess the effectiveness of object detection. Time analysis involves gathering critical times to model the system using queuing theory, including system, waiting, object detection, and service timestamps. Recommendations for enhancing the queuing mechanism of the server-less architecture are made based on quantitative investigation results.

The data is analyzed using MATLAB, which provides a table displaying service timings before processing and analysis. Performance test results help analyze the server-less architecture and choose the optimal queuing model. The system typically maintains a queue for users and serves them based on the first-in, first-out (FIFO) criterion. However, building a queue is not always necessary, as users can be served immediately upon log-in. Process sharing is an egalitarian approach where resources and service capabilities are distributed equally across all users or jobs. This leads to a direct proportional relationship between service time and the number of users, resulting in lower average response times. To install the server-less architecture, a virtual machine (VM) is used instead of a laptop. Oracle VM Virtual Box powers all VMs, requiring at least 40 GB of storage space, 4 GB of RAM, and 4 CPU cores. Docker and Kubernetes were used to build the server-less architecture, while YOLOv5 was used to build the object recognition software. Docker is an open-source tool that manages the process of creating containers using Docker images. The open-source Docker Hub infrastructure simplifies sharing and storing container images, making it easy to launch containers. Docker Files and Docker images allow for quick and easy container creation and launch into production mode. The master node coordinates the cluster's activities, delegating tasks and requests to worker nodes. Pods are clusters of worker nodes that containers run in, with pods sharing a connection to the same node. Sublet coordinates the flow of messages to and from worker nodes, ensuring container launch and operation. The Kubernetes command-line interface, or kubectl, provides command-line access to the cluster, allowing management by sending instructions to the master node. An internal tool is required to monitor connections to pods, such as Weave Net, which handles port mapping, connection formation, and routing within the cluster. This tool works effectively for our project, without impacting load balancing or service discovery.

System design

The server less platform is really just two Docker and Kubernetes virtual machines. The master node and the worker node are both represented by the two virtual computers. Under the supervision of the master node, the worker node's containers and pods are managed and coordinated. The worker node is now hosting the object detection programmer. Docker, Kubernetes, and Weave make up the master node.

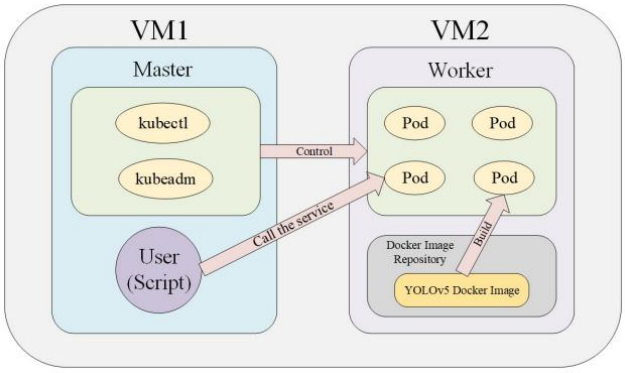

Figure 5: System design

Global Online Network. A worker node is comprised of the following software components: YOLOv5 and an automated object recognition programmer that is based on Docker. The server less architecture of the system is based on Kubernetes and Docker. Master and worker nodes are linked via Weave Net. With YOLOv5, there is an object detection tool available to users. See below for the specifics of this agreement. One cluster may be configured for the control plane of your system using Docker and Kubernetes. The Weave Net architecture is one option for connecting several nodes and pods. Once Docker and Kubernetes are ready to go, we can construct a master-worker Kubernetes cluster and verify that all of the pods and nodes inside it are able to interact with one another. After configuring the master node using kubectl, we can control the pods and containers on the worker node via the command line. In a cluster, the worker node is in charge of actually executing services and reporting its progress to the master node. You must keep this difference in mind. With server less technology in place, the worker node may be used for service development. We will implement the object detection algorithm using YOLOv5. After receiving the YOLOv5 image, the worker node may begin to build containers. Through containers, users may access server less services. The final step is to create service requests using Python scripts. Python programmers may be used to manipulate the arrival process, giving the impression that users are logging into the system when in fact they are not. Moving to a server less architecture will allow us to evaluate the object detection service's performance after the previous stages are complete.

Implementation

Oracle Virtual Machine Platform The system was configured with two virtual machines (VMs) using Virtual Box. With 4 CPU cores, 4 GB of RAM, and 40 GB of storage, each virtual machine will provide enough resources for the object identification algorithms. Virtual machines need Internet connectivity before they can configure networks. The ability for the two virtual machines to converse and collaborate is of the utmost importance. As a last step, provide static IP addresses to these two virtual computers. A host-only adapter and a network address translator (NAT) are required for this to operate. Network address translation (NAT) adapters enable virtual machines to interact with the internet, as opposed to host-only adapters. In order to connect to the Internet using NAT adapters, it is essential to acquire static IP addresses. A host-only adapter is the only means of doing this. The "enp0s3" adapter manages internet connectivity and network address translation (NAT), while the "enp0s8" adapter allows us to connect more virtual machines to our desktop. Following the configuration of the two virtual PCs, all project criteria were met. The 192.168.56.3 and 192.168.56.2 internal IP addresses are shared by the virtual machines running the Master and Worker roles, respectively.

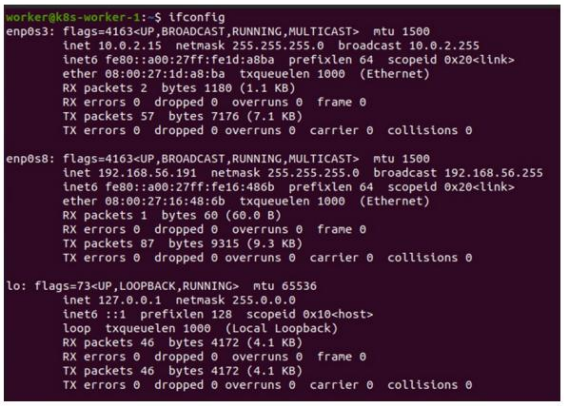

Figure 6: Network settings

The Master node should be configured with Kubernetes and Docker, along with software plug-ins like kubectl and Weave Net. The cluster's orchestration and administration are handled by these nodes. To enable sublet, deactivate swap and configure the hostname. To run the object detection service on the server-less architecture, create a Poisson arrival model for the user arrival process. Each user job must be delivered independently using a Poisson approach. To achieve this, use the Pope () function in Python. This allows child programmers to run autonomously from their parent processes. A sub process is used to mimic a user's request, and the object detection service is custom-coded to interface with each user's script. The main process's class "sub process. Pope ()" is used to for loop over each user's sub process script. The "kubectl exec" command is used to run the object detection service in the sub process script. To avoid I/O issues, manually terminate the sub process after object identification is complete. The kubectl run command is used to start the container containing the service to enable object detection.

In this YOLO object identification example, the weight is a pre-trained YOLOv5s model, which is lightweight due to its speed and object recognition capabilities. Folders containing user photos, arrival rate, confidence level, and total users are scriptable parameters.

Figure 7: Main process script

Figure 8: Sub process script

The experimental control group consists of equal photos to distinguish between them. The confidence score cutoff is set at 0.25 from the YOLOv5 object detection algorithm, and the control group is subjected to a default threshold of 0.25. The arrival rate is set at 0.2 users/s, which allows the system to take in new users at a Poisson rate. This is advantageous for investigating the connection between user load and system availability. The control group will conduct three trials with different user groups to evaluate their performance. Two crucial factors must be considered when assessing outcomes: prioritizing quality time together and applying queuing theory to the problem of system description. Time intervals for the system, object detection, pre-processing, inference, results selection, and service are all part of this process. Verifying the accuracy of the findings is the second step in assessing an object recognition algorithm. A high-quality object detection result is one that correctly identifies most items with minimal false positives. To ensure the server less architecture software works as expected, the object detection findings are displayed. Three user-submitted pictures, one with a single image, and 10 user-submitted images, all of which include multiple images, are selected at random from the coco128 collection.

Figure 9: Object detection example 1 for single-image user

Figure 10: Object detection example 2 for single-image user

See below for the results of the control group experiment for this project. All other groups are assessed in relation to this control group. With a confidence level of 0.25 and an arrival rate of 0.2 users/second, we utilize the same photographs for users of the same class here. During the request generation process, one request stands in for every user who logs into the system. It seems that five individuals are using the programmer, since we have seen five requests for user type (a). Type (a) has five users, all of whom are single-image users—that is, their projects include only a single photo. As the system is now getting logins from three distinct users, we will need to generate three requests for the "b" user type. Type (b) has three people who all utilize multiple photographs and would want to work with ten images each. For a c-type user, we generate five queries. Three of these people just have a single photo, while two of them have a lot. While users 1, 2, and 3 each utilize a single image, two of them make extensive use of photos (c). The relationships between user type, request, user, and image are shown visually in Figures 5.7 to 5.9. The preceding graphics demonstrate correlations that may be useful for other experimental groups as well.

Here are the results for the control group. The first step is to create a table that categorizes the time results according to the user type. Here we shall see mathematical representations of the link between the number of users and the length of the service. To clarify, the image is used for timing purposes throughout pre-process, inference, NMS, and object detection. People with several images tend to have more efficient use of their time than those with only one. We only use one measure for persons with one picture when it comes to profile images. If you're utilizing several photos, the tables will show you how long it takes to analyze all 10. User behavior is used to determine system and service timeframes instead of visual data. Because of this, averaging system or service time is unnecessary. We allot one system time and one service time for each user. These readings are foundational to every experiment in this project.

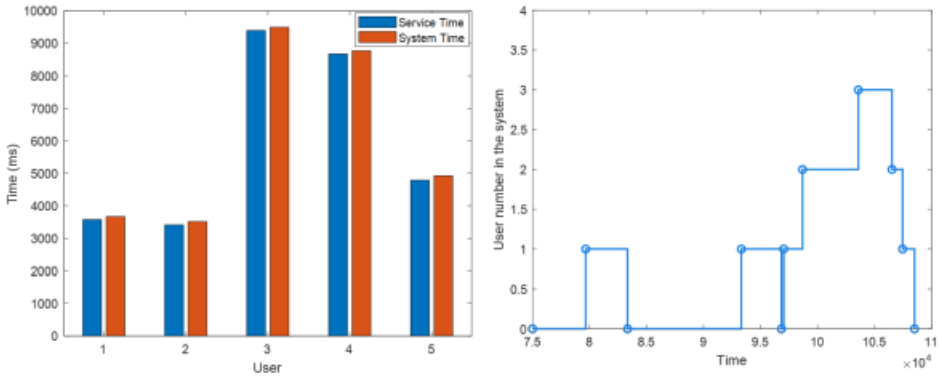

Figure 11: Service time and user number of user type (a) in control group

The time data, together with the correlations between service time and user number, may be used to extrapolate results for various experimental groups and user categories. A separate analysis is provided for every finding. With a little comparison, we can observe that the two scenarios have very similar system and service times. There is almost no waiting time in the system since service time makes up the great majority of system time—roughly 97% to 99%. Users are catered to instantly upon accessing the system; queues are not necessary. That being the case, it can't be that easy to simulate the server less architecture using mathematical models and the queue. We identify a strong association between the number of users and the duration of service time available to users, even when we maintain the quantity of photos per user constant. To evaluate the correlation between the number of users and the amount of time spent on the system, all trials should employ user type (a). When comparing the system with various users, user type (a) is ideal since the service time is much shorter and the quantity of images per user remains same. When this kind of user finishes using the system, they log out.

II. Conclusion

This project presents an object detection application using a server-less architecture and a mathematical model. The project aims to build practical models of server-less computing architectures by verifying that simple models, such as single-server or multi-server queues, can accurately mimic real-world system actions. The project involved conducting literature reviews, designing and implementing the system, and testing its functionality. Results showed that the number of users significantly affects service availability, making mathematical models unsuitable due to the lack of a queue. The study postulated that this server-less architecture may be best described by the processor sharing paradigm. The experiment provided a comprehensive description of the application of an item identification approach to a server-less architecture, and an analysis of the performance of server-less object detection software based on YOLOv5. The project was successful due to these outcomes and improvements. However, the virtual machine (VM) has restricted capabilities for processing and memory, which directly impacts service latency and speed, which in turn impacts object recognition performance. Large organizations often use resource-rich server-less platforms to launch their services, unlike the circumstances in this experiment. The testing focused on a single server configuration due to time and material restrictions. Future research could explore the possibility of installing multiple servers. User type (c) is related to two types of users: those with many photos and those with only one. After appropriate adjustments, the arrival rate will be the same for both types of users, or they may use a separate queuing system if their arrival rates differ.

REFERENCES

[1]. Abbadi, IM & Ruan, A (2013), "Towards trustworthy resource scheduling in clouds", Vol. 8, Issue. 6, pp.973-984.
[2]. Yan, Zhenget al., (2011), "Advances in Mobile Cloud Computing and Big Data", Springer Computational Intelligence and Complexity, ISBN 978- 3-319-45143-5.
[3]. PengLi; Song Guo; Toshiaki Miyazaki; Miao Xie; Jiankun Hu; WeihuaZhuang (2016), "Privacy- Preserving Access to Big Data in the Cloud", IEEE Cloud Computing, Vol.3, No.5, pp.34 -42.
[4]. Yue-Qin, Fan, (2017), "Security and Privacy Challenges in Mobile Cloud Computing: Survey and Way Ahead".
[5]. Lin, Guoyuanet,(2014),"Cloud service selection: State-of-the-art and future research directions", Elsevier, pp.134-150.
[6]. Li, Xiaoyong, and Junpinget, (2013), "Info-Trust: A Multi-Criteria and Adaptive Trustworthiness Calculation Mechanism for Information Sources", IEEE Access, 2013
[7]. Marcos Assunção.D, Rodrigo Calheiros.N, Silvia Bianchi, Rajkumar Buyya., (2015)," Big data computing and clouds: Trends and future directions‖, Journal of Parallel and Distributed Computing", Vol. 79–80, pp. 3-15.
[8]. Saurabh Singh, Young Sik Jeong, Jong HyukPark, (2016) "A survey on cloud comput ing security: Issues, threats, and solutions", Journal of Network and Computer Applications, Vol. 75, pp.200-222.
[9]. Fan, Wenjuan, and Harry Perroset, (2017), "Performance Analysis of the Reserve Capacity Policy for Dynamic VM allocation in a SaaS environment".
[10]. Sukhpal Singh, (2016), "A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges", Journal of Grid Computing.

[11]. Zhang, Ruiet, (2016), "Key Technology competencies of Progressive in Cloud".
[12]. Zhang, Raiart, (2015), "Cloud mobility over the bigdata", IEEE Simulation Modelling Practice.
[13]. Yongkui Liu, (2014), "Key Issues of Cloud Manufacturing Applied to Agricultural Production".
[14]. Sukhpal Singh, (2016), "A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges".
[15]. Seokho Son, "A SLA-based Cloud Computing Framework: Workload and Location Aware Resource Allocation to Distributed Data Centers in a Cloud", IEEE Journal of Cloud Computing, 2015
[16]. Prasadu Peddi (2017) "Design of Simulators for Job Group Resource Allocation Scheduling In Grid and Cloud Computing Environments", ISSN: 2319- 8753 volume 6 issue 8 pp: 17805-17811.
[17]. PengLi; Song Guo; Toshiaki Miyazaki; Miao Xie; Jiankun Hu; Weihua Zhuang (2016), "Privacy- Preserving Access to Big Data in the Cloud" , Vol.3,No.5, pp.34 -42.
[18]. Gaetano Anastasi, Emanuele Carlini, Massimo Coppola, and Patrizio Dazzi., (2017), "QoS-aware genetic Cloud Brokering Future Generation Computer Systems", Vol. 75, pp. 1-13.
[19]. Prasadu Peddi (2016), Comparative study on cloud optimized resource and prediction using machine learning algorithm, ISSN: 2455-6300, volume 1, issue 3, pp: 88-94.
[20]. Halabi, Talal, and Martine Bellaiche.,(2017), "Towards quantification and evaluation of security of Cloud Service Providers".
[21]. Armbrust. M, A. Fox, Griffith. R, Joseph A. D, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica., "A view of cloud computing", vol. 53, no. 4, pp. 50-58.
[22]. Md Whaiduzzaman, Mehdi Sookhak, Abdullah Gani, Rajkumar Buyy, "A survey on vehicular cloud computing", Journal of Network and Computer Applications,