DOI 10.12694/scpe.v20i2.1520
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 20, Number 2, pp. 299–315. http://www.scpe.org

STATIC LOAD BALANCING TECHNIQUE FOR GEOGRAPHICALLY PARTITIONED PUBLIC CLOUD

MANDEEP KAUR* AND RAJNI MOHANA†

Abstract. Large number of users are shifting to the cloud system for their different kind of needs. Hence the number of applications on public cloud are increasing day by day. Handling public cloud is becoming unmanageable in comparison to other counterparts. Though fog technology has reduced the load on centralized cloud resources to a remarkable extent, still load handled at cloud end is significantly high. Geographic partitioning of public cloud can resolve these issues by adding manageability and efficiency in this situation. Dividing public cloud in smaller partitions opens ways to manage resources and requests in a better way. But, partitioned clouds introduce different ends for submission and operations of tasks and virtual machines. We have tried to handle all these complexities in this paper. Proposed work is focused upon load balancing in the partitioned public cloud by combining centralized and decentralized approaches, assuming the presence of fog layer. A load balancer entity is used for decentralized load balancing at partitions and a controller entity is used for centralized level to balance the overall load at various partitions. In the proposed approach, it is assumed that jobs are segregated first. All the jobs which can be handled locally by fog resources are not forwarded to the cloud layer directly. Those are processed locally by decentralized fog resources. Selection of an appropriate Virtual Machine (VM) for filtered set of job, which are forwarded to cloud environment, is done in three steps. Initially, selecting the partition with a maximum available capacity of resources. Then finding the appropriate node with maximum available resources, within a selected partition. And finally, the VM with minimum execution time for a task is chosen. Results are compared with the results produced in this work with First Come First Serve (FCFS) and Shortest Job First (SJF) algorithms, implemented in same setup i.e. partitioned cloud. This paper compares the Waiting Time, Finish Time and Actual Run Time of tasks using these algorithms. After initial experimentation, it is found that in most of the cases, while comparing above parameters, the proposed approach is producing better results than FCFS algorithm. But results produced by SJF algorithm produce better results. To reduce the number of unhandled jobs, a new load state is introduced which checks load beyond conventional load states. Major objective of this approach is to reduce the need of runtime virtual machine migration and to reduce the wastage of resources, which may be occurring due to predefined values of threshold. The implementation is done using CloudSim.

Key words: Load Balancing, Public Cloud, Cloud Partitioning, Geographically Partitioned Cloud, Static Load Balancing, Overloaded Node Management, Fog Computing, IoT Devices, Fog stratum.

AMS subject classifications. 68M14, 68M20

1. Introduction.

Cloud Computing has the potential to affect a large part of IT industry. Nowadays, developers need not to concern about the over-provisioning or under-provisioning of resources. Elasticity of resources without spending a large amount of funds is a unique concept of its kind [2]. Basic characteristics of Cloud Computing as user friendliness, virtualization, automatic adaptation, scalability, resource optimization, pay-per-use, service SLAs, infrastructure SLAs etc. are attracting users in masses [26]. Cloud Computing is proving itself so beneficial, but there is a requirement of efforts to maintain performance and efficiency of services. Handling concurrent jobs, users and processes through such a large set of machines is a very difficult task. Load balancing is one of the serious concerns in success of cloud computing. Emerging trends like edge or fog computing are being integrated with cloud computing. These share the load on public cloud by shifting significant amount of work from centralized public resources to local fog processing units. Fog/Edge computing is the future of cloud technology. Apart from heavy load challenge, another big concern is of imbalanced load among servers. Many kinds of troubles occur due to overloaded as well as under loaded servers. Under loaded servers cause inefficient energy consumption, inefficient use of resources and add-on to the management overheads. On the other hand, overloaded servers can cause delay in response, low speed of processing, decreased throughput, increased makespan etc. To handle both these situations it is very important to adopt a load balancing mechanism which can distribute tasks evenly among all available servers [28]. Recent growth in the field of IoT has further increased the challenges. As per CISCO cloud traffic is expected to rise upto 14.1 ZB per year by [27]. Data associated IoT devices are the main cause of this growth. Similarly, Internet of Vehicles (IoV) is also an emerging concept which is a subclass of IoT. It refers to the network of heterogenous sensors

*Computer Science Engineering and Information Technology, Jaypee University of Information Technology and School of Computer Applications and Lovely Professional University, India (mandeep.13695@lpu.co.in)

†Computer Science Engineering and Information Technology, Jaypee University of Information Technology, India (rajni.mohana@juit.ac.in)

installed on vehicles [5]. In all such applications major issue is of latency. To handle these latency issues a new term fog computing has been introduced. Fog computing is capable to reduce the operational cost of cloud data centres and helps to avoid revenue loss due to WAN propagation delays [22]. Need of optimal allocation of jobs to available fog resources has led the researchers towards load balancing in fog networks [21]. In paper [12] authors have presented an in-depth analysis of principles of Green-IoT. This chapter also highlights different kind of barriers challenging implementation of Green-IoT in real life. Load balancing can either be dynamic or static. In dynamic load balancing, load status of nodes is checked while jobs are running. During the execution process if any node gets overloaded then few of the jobs are shifted to some other node, with lesser load. This process is known as virtual migration. Static load balancing, efforts are made to distribute tasks equally among VMs. That means while initially putting tasks in VMs, possible efforts are made to keep task distribution even. This method of load balancing is very efficient as it will reduce the need of VM migrations. Run time VM migrations are a kind of interrupt or overhead during the execution period of tasks [17]. In this paper, we have focused on static load balancing part. Load management becomes even more challenging when we are considering a large public cloud. A public cloud has numerous nodes, scattered around in various geographic locations. Huge number of users and large amount of outsourced cloud data makes it very difficult to achieve the performance requirements and system usability [30]. Cloud Partitioning is one of the popular techniques used for load management in clouds. In this technique, clouds are divided into small groups of nodes. Small partitions are more manageable as compared to large, public set of nodes. So handling load will become simple and efficient by using this technique. This paper is an attempt to implement decentralized load balancing through cloud partitioning at an initial state of task submission.

Contribution The main contribution of this work is to design and evaluate a decentralized load management model for public clouds and fog layer in a partitioned environment. It can fully utilize the resources available with Datacentres. Our objective is to introduce a technique for load balancing in a partitioned cloud. The proposed approach suggests that in a public cloud, presence of a substantial number of nodes can be used as a privilege. Static load balancing is done depending upon a threshold value. This value denotes the allowable and optimal amount of load in a partition at a time. Once the load in a partition has reached at that threshold value, that node will no more be considered for job allocation until a few of its jobs have been finished. Considering that we are talking about a public cloud, there must be a scope of remaining resources to handle a few of the jobs, maybe with minor resource requirements. Hence, while calculating the load state of a partition, keeping records of load at all nodes under a partition, its available resources and required resources of jobs can be very helpful. Proposed work implements static load balancing for handling load mismatches during the initial allocation of virtual machines. Static load balancing gives an overall better performance.

Allocation done with a node with a current maximum capacity of resources will be more reliable and need of VM migrations during run time can be reduced. Decentralized load calculation and management through is tremendously capable to reduce execution time to compare, search and allocation process.

Few extra efforts are involved in calculating load status of all the nodes in all partitions. Use of decentralized approach reduces this overhead. Finally, central controller will be getting details from partition balancers hence there is very less communication required across the partitions.

Paper Structure In following sections of this paper, a decentralized load management technique is discussed. Section 2 contains the literature review and existing work done in this area. In Section 3, the proposed approach is discussed in detail. An experimental, simulation-based evaluation and comparative study of our approach along with the retrieved measurements are presented and discussed in Section 4. Finally, Section 5 concludes this paper and identifies paths for future work.

2. Related Work. Many researchers have addressed the load balancing challenges in their work. In [7] a load balancing technique the cloud public cloud based upon cloud partitioning. They proposed to switch load balancing technology based upon the situation of load at a time. This algorithm applies game theory to handle the partitions with high load status. Though this algorithm needed further experiment to determine the efficient refresh rate and load degree calculation mechanism. The tradeoff was that much testing is required to guarantee system availability and efficiency.

Authors in [24] have proposed a system to perform load balancing as well as dynamic cloud partitioning. Major components of this approach are Partition Manager: responsible for assigning a job to a partition and Job Distributor: which decides the node to which this task can be allocated. To ensure effective distribution of load, partitions are created dynamically. This is done with the help of the honey bee algorithm. This paper left scope to improve the transparency and cloud division beyond geographic limits. They have used game theory to improve the efficiency in a cloud environment. According to the researcher the network is divided into cluster using a clustering algorithm. Every node is a part of a cluster. Every cluster has an Inter Cluster Communication (ICC) node. Clustering is done while initialization of the network.

In work presented by [8], authors have addressed the issues of Service availability and reliability, Lack of Service Level Agreements, Customer data security and privacy and Government compliance regulation requirements. They have proposed intelligent workload factoring service for proactive workload management with a fast and frequent data detection mechanism at the core. This mechanism helps not only in factoring requests on the volume of data, but also on contents of data. Major design goals of this workload refactoring include avoiding overloading scenario through load redirection, smoothening the workload dynamics in base zone application platform, and making flash cloud zone application platform agile through load decomposition. There are few improvements required, such as data security management for hybrid platform, handling data replication and consistency in the flash crowd zone, and load balancing schemes are implemented in two zones.

Authors in [14] have proposed a solution to calculate load degree based upon the turnaround time. This approach helps load balancer to improve load balancing strategies of a load balancing model in a public cloud. An idea of preliminary evaluation of a cloud partitioning approach is expressed in [23], which distributes task execution requests in a volunteer cloud. Validation is done through a simulation-based statistical analysis, using the Google workload data trace. Evaluations of this model are based on comparison between the results of purposed approach and the results of an unpartitioned cloud, using same random data sets. This approach has improvement scope, such as using the bio-inspired solution to add more sophistication. Adding lightweight performance monitors can help in improving performance. A workload classification mechanism can make workload management better. User Module submits the VM allocation request to Front-End Module, which is endowed with best-fit heuristic and random load balancing heuristic. It submits the VM allocation requests to the Server Manager Module. The Server Manager Module is responsible for balancing the load across hosts and performs energy-efficient server consolidation. This Module uses two threshold values, load-related migration threshold and load-related activation threshold to accomplish its job. Authors tried to remove the scheduling decision from Job's Critical Path to improving scheduling decision accuracy in [1]. For this, they have used Schedule First and Manage later approach and job replication. In their proposed model, they suggest replication of jobs and distributing these replicas to multiple servers. Whichever server will pick this job first will notify other servers having a copy of the same job. The main goal of this approach is to make task scheduling simple by removing load balancing tasks from the process of VM allocation. Though, this approach adds a signal propagation delay in the processing time.

In work expressed in [11], authors have tried to highlight multiple issues such as elasticity, energy efficiency, and high operational costs etc. For this, they have developed a combination of algorithms for initial VM placement, partial VM migration, and full VM migration. The initial VM placement algorithm can create or destroy VMs at any time, based on the current load at that VM. A Partial VM Migration algorithm is used by over utilized compute nodes to shifts few of the jobs to one or more other compute nodes. Full migration algorithm is used to shift full load of a compute node if it is underutilized. The purpose of this migration is to reduce the overall energy consumption of a Datacentre. A star-based partitioning and index algorithm for Resource Description Framework (RDF) data of robotic systems in [29] has devised a 3-step process. In first step a start structure is created by using MapReduce and HDFS to construct a coarsened weighted graph. In second step a balance partitioning algorithm is used to divide this graph. In third step a Compressed and Linked S-tree index is proposed to improve the query efficiency. In their paper [18] have discussed the need of fog computing by exploring its taxonomy, applications and various technologies involved in it. This paper has very well described the difference between cloud computing and fog computing. A mobility and heterogeneity-aware partitioning algorithm to support cross-domain resources partitioning, is presented in [9]. In this paper a service popularity-based smart resources partitioning (SPSRP) scheme is proposed. The basic architecture of

SPSRP scheme decouples the computing control layer from data processing layer.

In [6] authors have discussed the tradeoff between power consumption and transmission delay in fog cloud computing. They have proposed an optimal workload allocation between fog and cloud resources. Objective achieved is to get job done with minimal power consumption and restricted transmission delay. For improving efficiency and security [15] have proposed a model to authenticate and compare the load status of various Edge Data Centres (EDCs). SDN-based modified constrained optimization particle swarm optimization (MPSO-CO) algorithm proposed by [25] uses the reverse of mutation particles and linear decrease inertia weight to enhance the performance of constrained optimization particle swarm optimization (PSO-CO). This technique results in reduced latency and improved QoS in Software Defined Cloud/Fog Networking (SDCFN). Technique proposed by [10] decouples computing control layer from data procession layer through a service popularity-based smart resource partitioning (SPSRP) controller. Table 2.1 presents the summarized view of research work done on static load balancing until now. In [16] authors have proposed a model based upon time-based data driven approach to predict load predictions in various in different utilization sectors. This model is capable to accurately predict the energy demand in residential and commercial sector of smart device users.

Authors of [4] have discussed about the variants of Ant Colony Optimization (ACO) and its role in solving discrete problems in various areas of science and engineering discipline, including load balancing. Finding an appropriate node with sufficient resources is a crucial part of load balancing. One such method to find efficient sources is a logarithmic spiral based local search strategy, namely logarithmic spiral local search (LSLS), suggested by [20]. In [19] authors have presented a deep understanding about taxonomy of fog computing, its differences from cloud computing and edge computing technologies, applications, emerging key technologies and various challenges involved in fog technology. Authors of [3] have presented an Artificial Bee Colony (ABC) Optimization algorithm.

As compared to the above approaches, our proposal suggests a combination of centralized and decentralized load management. It ensures the benefits of unlimited resources of public cloud while removing the complexities of handling the substantial number of nodes. Proposed model ensures the better utilization of resources in the partitioned public cloud environment. It also helps in retaining the individual identity of nodes in a partition to ensure the accessibility to load status information and other details of a node. Initially, the node with maximum available resource instances is chosen for job allocation. So, chances of dynamic VM migration can be reduced. The objective is to increase the efficiency of load balancing mechanism lowering the usage and management costs.

3. Proposed Approach. This section contains the presentation model, task distribution between various modules and algorithm of our proposed approach for load balancing in public clouds and fog layer. The cloud is initially partitioned statically based on geographic location. In proposing a model, the primary objective is to reduce the size of public cloud by dividing it into multiple partitions and to apply a decentralized load balancing mechanism to ensure optimal utilization of resources. It is assumed that a significant part of total load is processed at fog layer. These jobs, restricted to reach cloud layer include the latency sensitive and security sensitive jobs. Fog computing environment is required to be implemented in time critical applications. In current scenario data is generated by IoT devices and processed in cloud environment. Most operations of these IoT devices depend upon data transmission to-and-from cloud layer. This is not considered feasible while keeping in view the time and distance constraints. Similarly, the applications involving sensitive data which user is not intended to share on public cloud can be shifted from cloud to fog environment.
So, cloud environment is responsible for jobs left unprocessed by fog environment.

As shown in Fig. 3.1, major entities in this model are clients, partitions, nodes, load balancers and controllers. A node is a provider which holds physical cloud resources which a client requires to process his job. Within that partition, a load balancer is deployed to keep track of load status of the partition. This balancer collects load-related data of individual nodes, compares it with threshold value to determine in which load category this partition falls currently. Same information is conveyed to the controller for making a final decision related to task allocation to a partition and anode. Balancers are local to partitions whereas a centralized controller is an entity which can keep track about all the partitions through information received from load balancers.

The proposed model is a composition of three software modules which are Client (CM), Balancer (BM) and

TABLE 2.1
Summarized view of research work done on static load balancing

Title	Proposed Model	Limitation
A Load Balancing Model Based on Cloud Partitioning for the Public Cloud [7]	Load balance model for the public cloud based on the cloud partitioning concept with a switch mechanism to choose different strategies for different situations like high, low, normal load status. This model applies game theory to load balancing strategy to improve the efficiency in the public cloud environment.	1. Lacks detailed cloud division methodology, 2. Effectively determining the refresh period, 3. Devising a good algorithm to set Load degree, 4. Testing is required to compare different load balancing strategies, 5. Many tests are to be performed to guarantee system availability and efficiency.
A novel approach for Dynamic Cloud Partitioning and Load Balancing in Cloud Computing Environment [24]	The strategic model that performs load balancing as well as dynamic partition of the nodes of different cloud. Game theory is used to load balancing strategy to improve the efficiency in the cloud environment	1. Need to increase levels of transparency 2. Requires effective technique in updating the status report. 3. The time intervals are not very well managed, 4. Dynamic balancing technique could be made dynamic, 5. Finding alternatives to geographical cloud division methodology.
A Cluster-Based Load Balancing Algorithm in Cloud Computing (Surbhi Kapoor, 2015)	A distributed algorithm for load balancing in the master-slave architecture that outperforms the Closest Datacentre algorithm in terms of task distribution across the system and optimal system performance	To evaluate effectiveness of the proposed model in scenarios where a node belongs to more than one cluster and we believe that effective load balancing could be achieved in this case as well
Resource Allocation Issues and Challenges in Cloud Computing (S.Thamarai Selvi, 2014)	Addressed issues are: 1. Resource Provisioning, 2. Job Scheduling, 3. Resource Overbooking, 4. Scalability, 5. Pricing, 6. Load Balancing, 7. Multitier applications, 8. Availability, 9. Overheads in Network I/O Workloads, 10. QoS constraints	1. Lacks elasticity, 2. Need to minimize the costs and maximize resource utilization, 3. Need to assure high availability for long running jobs, 4. Better parallel task scheduling
Proactive Workload Management in Hybrid Cloud Computing+B17 [8]	Addressed issues are: 1. Service availability and reliability, 2. Lack of Service Level Agreements, 3. Customer data security and privacy, 4. Government compliance regulation requirements	1. Load balancing schemes are implemented in two zones, 2. Efficient data replication and consistency management in the flash crowd zone, 3. Better security management for a hybrid platform
Load Degree Calculation for the Public Cloud based on the Cloud Partitioning Model using Turnaround Time [14]	Solution to calculate Load degree of a node in the public cloud based on the Turn Around Time	Lacks efficiency of algorithms
A Workload-Based Approach to Partition the Volunteer Cloud [23]	A preliminary evaluation of a cloud partitioning approach to distribute task execution requests in volunteer cloud. Comparison between the results of the proposed model, i.e. partitioned cloud and an unpartitioned cloud which uses the same random tasks.	1. Requires more sophisticated algorithms such as bio-inspired solutions, 2. Adding lightweight performance monitoring, 3. Better workload classification mechanisms
Replication-based Load Balancing [1]	Removing the Scheduling Decision from Jobs Critical Path, Improving Scheduling Decision Accuracy. Schedule First and Manage later approach is used by implementing Job replication.	1. Presence of signal propagation delay, 2. Devise single parameter configuration that is optimal for all systems.
Decentralized and Energy-Efficient Workload Management in Enterprise Clouds [11]	An issue addressed: 1. Elasticity, 2. Scalability, 3. High operational costs, 4. Efficient energy consumption. Three algorithms are introduced which are: 1. Initial VM Placement, 2. Partial VM Migration, 3. Full VM Migration,	1. Needs decentralized workload manager in an open-source cloud operating system, such as e.g., OpenStack, 2. Can use additional parameters into load-balancing algorithms,
A Partitioning and Index Algorithm for RDF Data of Cloud-based Robotic Systems [29]	Considers the 2-hop star structure as the basis object and proposes a partitioning and index algorithm	More orientation towards data structure operations. Output is dependent upon datasets, query type and strategies for graph operations.

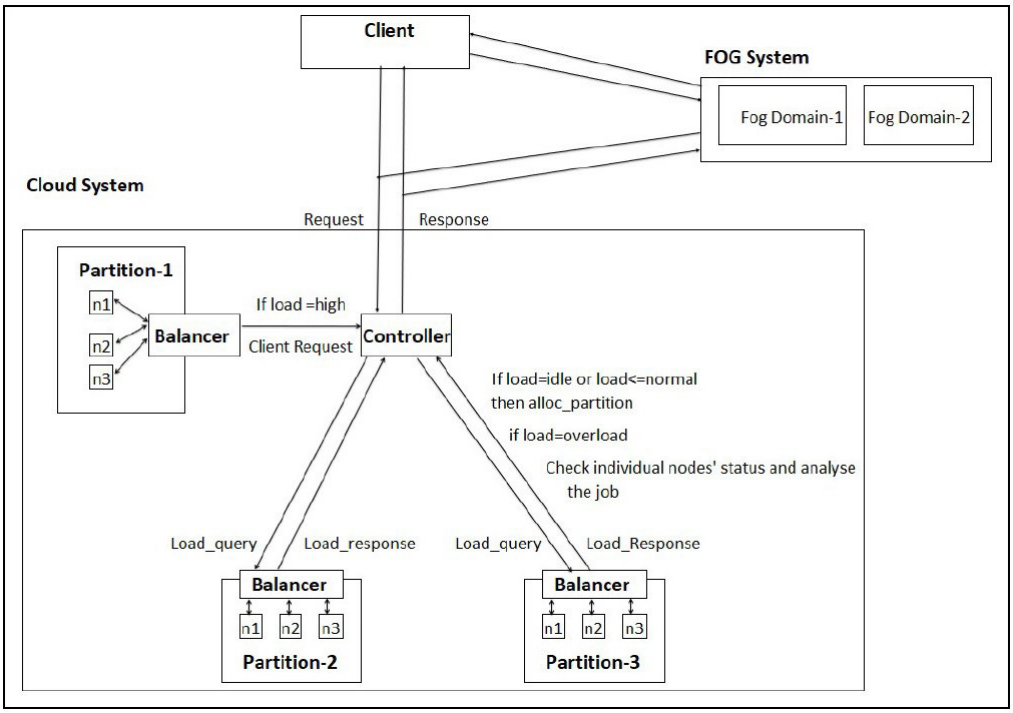

FIG. 3.1. *Architecture of Proposed Model for Load Balancing in Partitioned Cloud*

Centralized Controller (CCM). These three modules have pre-assigned roles to play during initial VM allocation to a task. These modules are capable to interact with each other wherever required. Following three modules are involved in the overall processing of the proposed approach. These are:

1. Client Module (CM) : Main responsibility of this Module is to filter the jobs to be processed by cloud environment and to submit those requests to a server node, through a broker. While submitting a request, the client is responsible to provide details about the resource requirement of that task. These details are very much important while choosing a node for final job allocation.
2. Balancer Module (BM) : Balancer module is local to partitions. They receive a notification whenever a client submits a task to a node belonging to that partition. Balancer has overall responsibility to manage load in a partition. It must keep record of load status of each node in the partition. The balancer is the accessing point of load status information of a partition, hence a controller always interacts with this module.
3. Centralized Controller Module (CCM) : This Module works like a centralized entity. It collects information from Balancer of all the partitions and makes decisions based on that information. Controller module is invoked by a Balancer whenever it finds all its local nodes are fully occupied. The controller asks all other partitions Balancer for their load status and finally directs job to the partition which fulfills following criteria.
   (a) Maximum amount of available resources.
   (b) Enough resources to cater the needs of the job.

In the proposed model, a Client Module (CM) captures the details of the service request being submitted to a node by a user. These modules prepare an estimate of resources required by the current job. Once the resource requirement details are ready, a notification is sent to the Balancer (BM) of the partition to which this node is associated with.

On receiving this notification Balancer (BM) will search resource availability status of that specific node and

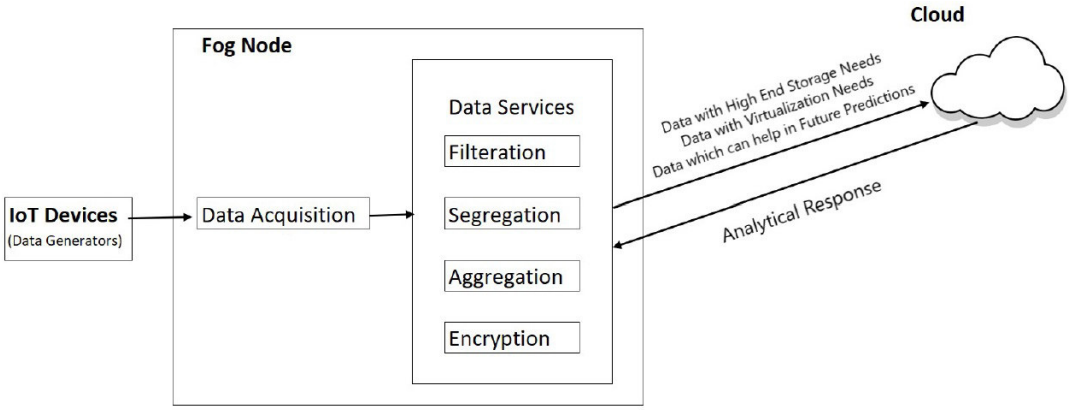

FIG. 3.2. *Distribution of Tasks between Fog and Cloud.*

will compare it with the resource requirement of the newly submitted job. If BM finds that sufficient number of resources are available, then it will allocate the job to the same node. Else it will compare jobs resource requirement with resource availability of other nodes in the same partition. If there exists any node(s) which are having sufficient resources to serve current job, then job is allocate to that node. If there is more than one such node, then the best possible allocation is done by allocating jobs to the node which has maximum availability of resources so that there are least chances of a need of VM migration.

If BM finds no node within partition which can serve job, then it will notify the Central Controller Module (CCM). CCM has the centralized access to the status of all partitions via their load balancers. Upon receiving notifications from BM, CCM will ask for fresh status information from all other BMs. On receiving this, first it will check the load status of all partitions. If it finds a single partition with normal or low load status, then the next step is to search for a node in that partition which has maximum available resources with it and the job is assigned to that node. Load status of nodes and the partition is updated in terms of resource availability. If there are multiple partitions then the one with max availability status is chosen and following same process a node is chosen in it.

There is a possibility that the load status of all the partitions is high, i.e. all the partitions have number of nodes with high load status than predefined threshold value. In such situation, resource availability of individual nodes is checked. Reason behind this step is that may be the percentage of nodes with high work load is exceeding threshold. But still, in remaining percentage there can be few nodes with low or normal load. The available resources of these nodes are usually ignored, but can be utilized in a better manner.

In rare situations, CCM would have not find any individual node with low or normal load status. In that case client module is notified about the non-availability of resources and job is kept unallocated till some of the running jobs are not complete and resources allocated to them are released. Summarized working of all the 3 modules is described in Table 3.1.

Job Distribution among cloud and fog Layer

It is a crucial decision-making point to distribute the jobs among fog and cloud, which highly depends upon their contribution in this fabricated system. Cloud services are assumed to provide services as well as data analytics required in operations of IoT devices. Fog provides various types of data services in IoT environment such as Data Filtering, segregation, Aggregation, Data Encryption, Catching etc. [13]. Hence it can be concluded that fog environment will retain all job requests which are short-span and more frequent. On the other hand, cloud environment will handle jobs which are less frequent, long term and require large amount of resources.

In the proposed model, the load balancing algorithm is being applied on the jobs submitted to cloud.

Evaluating Load Status for Node Allocation

As described earlier the partitions groups certain number of data centers. Each datacenter accommodates one

TABLE 3.1
Role of various modules

Stepwise Process	**Client Module (CM)**	**Balancer Module (BM)**	**Central Controller Module (CCM)**
Step-1	Capture details of a service request being submitted by a user.		
Step-2	Based upon pre-defined categories, segregate the jobs deciding if it is to be processed locally in fog environment or cloud environment.		
Step-3	Send the job to appropriate end, based upon decision made in Step-2.		
Step-4	Prepare estimates of required resources.		
Step-5		If the current node is overloaded then notifies the Balancer Module of the current partition.	
Step-6		Notified about the new job submission. Receives resource requirement details from Client Module.	
Step-7		Traverse all the nodes in the partition and compares their resource availability status with resource requirement details.	
Step-8		Allocates job to the best suitable node based upon the resource availability. Or notifies the Central Controller Module if no node is available locally in the partition.	
Step-9			Notified by the Balancer Module, collects status information from all the balancer Modules.
Stepwise Process	**Client Module (CM)**	**Balancer Module (BM)**	**Central Controller Module (CCM)**
Step-10			Picks the best suitable partition based upon load status.
Step-11			Picks the best suitable node in that partition, based upon resource availability. Notify concerned Balancer Module.
Step-12		Allocate job to notified node. Update load status and resource availability details of partition and node respectively.	
Step-13			If all partitions have a high load status, then traverse the entire list of individual nodes and pick the one with maximum available resources.
Step-14			Compare this nodes resource with the requirement. If sufficient resources are available, Notify concerned Balancer Module.
Step-15		Allocate job to notified node. Update load status and resource availability details of partition and node respectively.	
Step-16			If none of the nodes possess sufficient resources, then notify the client Module.
Step-17	Get notification from Controller Module to wait.		
Step-18	Capture details when the job is processed and the response is sent to the client.		

or more hosts, which are physical instances of resources which a client can request. These hosts can create k number of VMs as per requirement. Task allocation is done to these Virtual machines (nodes). There is a need to evaluate the availability at two levels as given below:

1. Determining the partition Pi, to which a job can be assigned.
2. Determining the node Ni, inside partition Pi, to which a job can be assigned.

As partitions do not have any load status of their own, they depend upon the load status S of individual nodes grouped inside them. Four predefined load states are considered, which are common for partitions and nodes and are calculated as follows:

1. **Idle load state**
   (a) A node N_j falls under idle state if its available resources $R(n)_j$ exceed a threshold $t(n)_{idle}$. i.e.

   $$N_{idle} = R(n)_j > t(n)_{idle}$$

   where $t(n)_{idle}$ is the threshold at which a node is considered idle. N_{idle} shows the idle status of node N.

   (b) A Partition P_i falls under idle state if the number of idle nodes in this partition has exceeded a threshold $t(p)_{idle}$ i.e.

   $$P_{idle} = Count(N_{idle}) > t(p)_{idle}$$

   where t(p)idle is the threshold at which a partition is considered idle. Pidle shows the idle status of partition P.

2. **Normal load state**
   (a) A node N_j falls under normal state if its available resources $R(n)_j$ exceed a threshold $t(n)_{normal}$. i.e.

   $$N_{normal} = R(n)_j > t(n)_{normal}$$

   where $t(n)_{normal}$ is the threshold at which a node is considered normal. N_{normal} shows the normal status of node N.

   (b) A Partition P_i falls under normal state if the number of normal nodes in this partition has exceeded a threshold $t(p)_{normal}$ i.e.

   $$P_{normal} = Count(N_{normal}) > t(p)_{normal}$$

   where t(p)normal is the threshold at which a partition is considered normal. Pnormal shows the normal status of partition P.

3. **Overloaded load state**
   (a) A node N_j falls under ovld state if its available resources $R(n)_j$ exceed a threshold $t(n)_{ovld}$. i.e.

   $$N_{ovld} = R(n)_j > t(n)_{ovld}$$

   where $t(n)_{ovld}$ is the threshold at which a node is considered ovld. N_{ovld} shows the ovld status of node N.

   (b) A Partition P_i falls under ovld state if the number of ovld nodes in this partition has exceeded a threshold $t(p)_{ovld}$ i.e.

   $$P_{ovld} = Count(N_{ovld}) > t(p)_{ovld}$$

   where t(p)ovld is the threshold at which a partition is considered ovld. Povld shows the ovld status of partition P.

4. **Full load state** A Partition P_i falls under full state if the number of full nodes in this partition has exceeded a threshold $t(p)_{full}$ i.e.

$$P_{full} = Count(N_{full}) > t(p)_{full}$$

where t(p)full is the threshold at which a partition is considered full. Pfull shows the full status of partition P.

Task Assignment

The proposed system is focused upon static VM allocation in a partitioned cloud environment or fog environment. Clients of cloud and fog system submit their requests to nodes. If the current node is having sufficient resources to handle request, then it will process request itself. If no then a suitable node is searched in the current partition. If none of the nodes has sufficient resources, then search is continued in other partitions. The complete process is described below.

Whenever a task is submitted to a node N_j in a partition, initially the load state of same node $R(n)_j$ is evaluated. If found that the current node is capable to accept more jobs (i.e. N_{idle} or N_{normal}) with specified resource requirements, the task is assigned to it. In addition, the load status of the partition and available resources will be updated. If found that current node is not capable to accept task (i.e. N_{ovld}), then other nodes in same partition are evaluated for their load state.

Following possible results can be there after this load state evaluation:
Case-1: If a single node N_j is found with idle or normal load state, assign the task to this node, update the load status of partition $S(p)_i$ and node $R(n)_j$.
Case-2: If multiple nodes can accept the task, create a list of all such capable nodes $N_{capable}$.
Case-3: If no node in current partition is found with idle or normal load state, then let balancer handover the request to the controller. Now controller will call the balancers of other partitions and will collect the load status of all partitions by calling searchPartition () method.

Algorithm 1 Partition Selection

```
1: procedure PartitionSelection(taskId, taskLength, pesNumber, taskFileSize, taskOutputSize,
   utilizationModelCpu, utilizationModelRam, utilizationModelBw)
2:     if partition_state [ partition_id] = idle or partition_state [partition_id]= normal) then
3:         selected_partition ← partition_id
4:         if node_state [ node_id] = idle or node_state [ node_id] = normal then
5:             selected_node ← node_id
6:         else
7:             for all node_id ∈ partition_id do
8:                 if node_state[node_id] ← high then
9:                     continue
10:                else
11:                    selected_node ← node_id
12:                end if
13:            end for
14:        end if
15:        partition_state[partition_id] ++
16:    else
17:        searchPartition (partition_state [ ], node_state [ ] , node_mem, node_cpu,mem_req, cpu_req)
18:    end if
19: end procedure
```

Controller Module will traverse through all the remaining partitions by invoking their balancers. Wherever

it finds a partition with P_{normal} or P_{idle} load state, it will add that partitions id in a list of capable partitions $P_{capable}$.

While traversing all the partitions three cases are possible:

Case-1: If found single partition then assign the task to the balancer Bi of that partition and update its load status.

Case-2: If multiple partitions are capable to accept the task, create a list of all such capable partitions. Once this list is complete, we can choose the best possible partition in the context of available resources through searchPartitionnohigh ().

Case-3: If no partition is found with idle or normal load state, try to find individual nodes in overloaded partitions P_{ovld}. Call searchPartitionhigh () method.

Algorithm 2 Determining Availability of Capable Nodes

```
    procedure SearchPartition(partition_state [ ] , node_State [ ], node_mem, node_cpu, mem_req, cpu_req)
2:      for all partition_id ∈ partition_list[]  do
            if partition_state[partition_id] = idle or partition_state[partition_id] = normal then
4:              selected_partition ← partition_id
                if node_state[node_id] = idle or node_state[node_id] = normal then
6:                  capable_i ← partition_id
                    i++
8:              else
                    continue
10:             end if
            end if
12:         if capable_size > 1 then
                searchpartitionNoHigh ()
14:         else
                searchpartitionHigh ()
16:         end if
        end for
18: end procedure
```

Algorithm 3 Partition Selection if Available with Normal State

```
    procedure partitionHigh(capable[ ], partition_state [ ], node_state [ ], node_mem [ ], node_cpu [ ], req_mem,
    req_cpu)
        for all partition_id ∈ capable[ ] do
3:          if max_resource[i] < capacityatpartition_resource[i]() then
                max_resource[i] = capacityatpartition_resource[i]()maxnode_resource[i] = capable[k]
            end if
6:          if max_resource[i+1] < capacityatpartition_resource[i+1]() then
                max_resource[i+1] = capacityatpartition_resource[i+1]()
                maxnode_resource[i+1] = capable[k]
9:          end if
        end for
    end procedure
```

Algorithm 4 Partition Selection if Not Available with Normal State

procedure PARTITIONNOHIGH($capable$ [ ], $partition_{state}$ [ ] , $node_{state}$ [ ], $node_{mem}$ [ ], $node_{cpu}$ [ ], req_{mem}, req_{cpu})
  for all $partition_{id} \in capable[]$ **do**
    if $max_{mem} < capacityatnode_{mem}$ **then**
4: $max_{mem} \leftarrow capacityatnode_{mem}$
      $max_node_{mem} \leftarrow capable[i]$
    end if
    if $max_{cpu} < capacityatnode_{cpu}$ **then**
8: $max_{cpu} \leftarrow capacityatnode_{cpu}$
      $maxnode_{cpu} \leftarrow capable[i]$
    end if
    allocation( )
12: **end for**
end procedure

Algorithm 5 Calculation of Capacity of a Node

procedure CAPACITYATNODERESOURCE($partitionid$, $nodestate$[ ] , $noderesource$[ ])
  for all $node_{id} \in partition_{id}$ **do**
    $capacity_{resource}[node_{id}] = node_{resource}[node_{id}] - node_{state}[node_{id}] * consumption$
  end for
5: return $capacity_{resource}$
end procedure

All the nodes are given an amount of various resources. Which node is in an idle, normal or overload state, is entirely calculated based upon the available (free) amount of these resources. This amount is being said the capacity of a node for that resource. To calculate the capacity of nodes the amount of resources under use is deducted from total allocation of that resource in the node.

Algorithm 6 Calculation of Capacity of a Partition

procedure CAPACITYATPARTITION($partition_{id}$,$node_state$[ ],$node_resource$[ ])
  for all $node_{id} \in partition_{id}$ **do**
    $capacity_{resource}[node_{id}] \leftarrow node_{resource}[node_{id}] - node_state[node_{id}] * consumption$
  end for
  return $capacity_{resource}$
6: **end procedure**

All the nodes are given an amount of distinct resources. Which partition is in an idle, normal, overload state, is calculated based on the number of nodes in that partition, falling within a category based on a threshold value for various load states. This load state is being considered as a capacity of a partition for that resource. To calculate the capacity of partitions, capacity of resources at each node for each resource is used as an input.

Once partitions with maximum capacity of various resources i.e. $Max(S(p)_j)resourcex)$ have been calculated and inside those partition, nodes with maximum capacity i.e. $Max(R(n)_j)resourcex)$ have been identified, we must select an appropriate node which can cater the requirement of all resources. For example, we can start with the node, which has maximum available memory. We must check if this node has enough processing units also. If yes, the task is allocated to this node. If no, then a node with maximum capacity of available CPU resources will be chosen and will be checked to see if it has enough memory available. A node allocation is possible only if it has sufficient instances of each resource available as per requirement of tasks. If no such node is found, then the task allocation is delayed until some partition is not capable to cater all resources requirement;

TABLE 3.2
Time and Space Complexity of Algorithms and its Modules

Sr. No.	Step	Time Complexity	Space Complexity
1.	Testing the current node for availability of resources	O (1)	O (1)
2.	Testing other nodes in the current partition	O(Count($P_{current}$(N)))	O (1)
3.	Testing other partitions		
	a) Each partition has same no. of nodes	O(Count(P)*Count(N)) where N represents the multiplication of partition count and node count	O(Count(P_i(N))) where N represents no. of nodes in a partition under consideration.
	b) Partitions have different no. of nodes	O(Sum(Count(P_i(N)))) where N represents sum of all the nodes in all partitions	O(Count(P_i(N))) where N represents no. of nodes in a partition under consideration
4.	Testing among multiple capable partitions .	O(Count($P_{capable}$)) where N represents no. of partitions	O(Count(P_i(N))) where N represents no. of nodes in partition under consideration
5.	Testing among multiple capable nodes	O(Count($P_{capable(i)}$(N))) where N represents no. of nodes in partition under consideration	O(Count(P_i(N))) where N represents no. of nodes in partition under consideration

allocation () function is called for this purpose.

Again in this situation, the decision is made based upon following criteria:

1. Find out the node with maximum availability of $resource_x$ i.e. $Max(R(n)_j)resource_x)$
2. Check if this node has sufficient availability of all other resources which can be calculated by comparing the availability with the requirement. i.e.
   $Max(R(n)_j)resource_x) > requirement_{resource(x+1,x+2,..,k)}$
3. If node N_j satisfies the above said criteria, then select this node and allocate the task. Else evaluate the node $Max(R(n)_{j)resource_x+1})$ and repeat these steps.
4. If none of the nodes satisfy above criteria, then balancer notifies the controller to search in other partitions.

Algorithm 7 Node allocation to a task

procedure ALLOCATION($capacityatnode_{mem}[\ ]$,$capacityatnode_{cpu}[\ ]$,req_{mem},req_{cpu}, $max_partition_{mem}$,$max_partition_{cpu}$)
  if $req_{mem} <= capacityatpartition_{mem}(minpartition_{mem}, capacityatnode_{mem}$ **then** AND $req_{cpu} <= capacityatpartition_{cpu}(maxpartition_{mem} capacityatnode_{cpu}$
    $selected_partition \leftarrow maxpartition_{mem}$
  else if $req_{mem} <= capacityatpartition_{mem}(maxpartition_{cpu}, capacityatnode_{mem}$ **then** AND $req_{cpu} <= capacityatpartition_{cpu}(maxpartition_{cpu} capacityatnode_{cpu}$
    $selected_partition \leftarrow maxpartition_{cpu}$
  else
7: statement: No allocation possible currently
  end if
end procedure

Time and Space Complexity During the entire process, time and space complexity varies as per entities under consideration. Initially, when the job was submitted at that time only one node availability status is evaluated. Hence, the complexity at this step is O (1). In all the other cases complexity class O(N) is applicable and value of N keeps on changing as per the case. Following is the summary of This entire process involves following steps along with their corresponding complexities.

In Table 3.2, P stands for Partition and N stand for a Node. Overall timIn the above table, P stands for Partition and N stand for a Node. Overall time and space complexity will be affected by the count of partitions

TABLE 4.1
Implementation Setup Parameters

Sr. No.	Parameter Name	Parameter
1.	No. of Partitions	2
2.	No. of Brokers	2
3.	No. of Datacenters	4 (2 Datacenters with each partition)
4.	No. of Hosts	2 Hosts each Datacenter
5.	No. of VMs	6 VMs in each Datacenter
6.	No. of tasks	40 tasks with each broker

	Max Finish Time (millisec)	Max Waiting Time (millisec)	Max Actual Run Time (millisec)
Proposed Algorithm	3920.38	2952.54	926.20
FCFS	5889.90	4799.47	945.03
SJF	1834.20	975.89	960.25

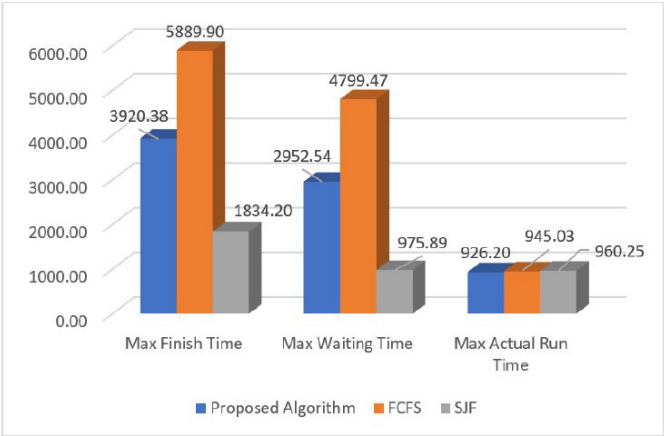

FIG. 4.1. *Average Max Execution Time Comparison*

and nodes in partitions. In all the cases linear alternatives are chosen to keep the complexities low.

4. Implementation and Results. This section shows the results of the evaluating the proposed algorithm. The algorithm is implemented in CloudSim by following setup details. The implementation is done on this small set of attributes for sampling purposes with certain assumptions.

For evaluating the performance of the proposed algorithm, the results have been compared with FCFS and SJF algorithms within same setup. The experiment is conducted by executing each algorithm 5 times. The VMs and tasks are heterogenous in their sizes. And the comparison results are shown below.

The results are compared for average of Finish time of tasks, waiting time of tasks and actual run time of tasks. Further, for all these three evaluation parameters are measured for total, average and maximum time of tasks. According to generated results proposed algorithm is producing better results when compared to FCFS algorithm, but the performance of SJF algorithm is better in all the cases. Fig. 4.1 shows the comparison of Maximum Execution time of a task during simulation. During most of the execution instances, it is found that the results of the proposed algorithm are better in case of Maximum Actual Run time of tasks. In Fig. 4.2 comparison is made between average total waiting time, average total finish time and average maximum actual time of all the tasks. It concludes that the proposed algorithm is capable to perform better than the FCFS algorithm in the same setup. Fig. 4.3 compares the actual run time of the tasks. Like other cases, it too shows that the proposed algorithm produces better results in comparison of FCFS but SJF algorithm is still better than this.

	Average Finish Time (millisec)	Average Waiting Time (millisec)	Average Actual Run Time (millisec)
Proposed Algorithm	1839.89	1110.11	668.08
FCFS	2352.28	1605.14	695.51
SJF	979.57	276.00	706.38

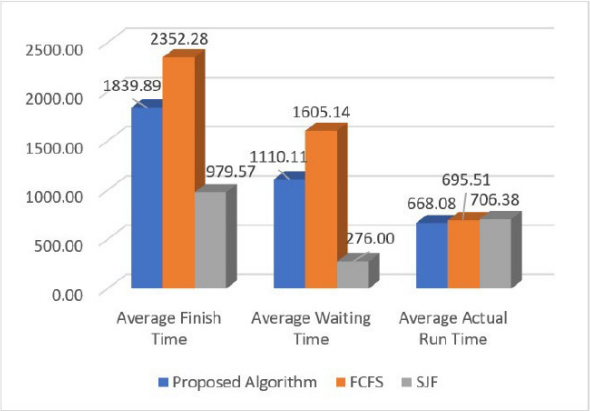

FIG. 4.2. *Average Time Comparison*

	Total Finish Time (millisec)	Total Waiting Time (millisec)	Total Actual Run Time (millisec)
Proposed Algorithm	93834.15	56615.47	34072.09
FCFS	160563.31	107551.58	46196.64
SJF	39182.74	11040.08	28255.36

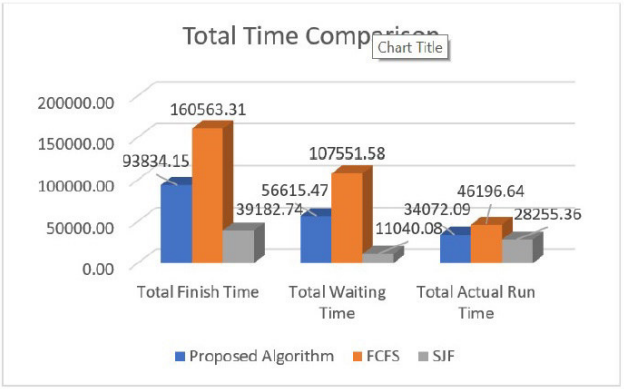

FIG. 4.3. *Total Time Comparison*

From the experimental results above, it can be seen that in most of the cases the output from SJF algorithm is better than proposed model. It looks impressive when talking about small scale local applications. But when we are focusing upon public cloud, the number of submitted jobs can be huge and their length unpredictable. In such environment, chances of resource starvation increase in manifolds. The consequences can be in terms of delayed response, longer waiting time for larger jobs, violated SLAs and so many others. In short, SJF algorithm focuses on jobs on the basis of a single parameter i.e. job length. This approach cannot be considered very beneficial for huge systems such as public cloud.

5. Conclusion. In the light of results produced till now the proposed algorithm works better in many of the cases. Choosing virtual machines on the basis of minimum execution time is helpful in reducing the waiting time of tasks. Also, adding one more load state will certainly reduce the number of unhandled tasks. But further work can be done by finding techniques to improve the Makespan, Failure/non-served instances of tasks and better ways to share the tasks load among available VMs. Better evaluation parameters can be associated and results can be improved to out-perform more advanced and complicated algorithms. Better techniques are required for heterogenous tasks and VMs which are a reason to add non-predictive load allocation requirements. Introduction of fog stratum in this system can further share the load by shifting latency sensitive jobs to the fog devices. Locally processing a share of jobs at fog layer will prove helpful in reducing bandwidth congestion, traffic flow over the network and wide area network propagation delays. Fog layer also allows a user to choose the optimal options in terms of compatibility, minimal geographical distance and decentralized/local communication and processing. Combining public cloud partitioning and fog computing can produce much more efficient results. Finally, partitioning technique can be very much useful to handle public clouds.

REFERENCES

[1] Ariel Orda Amir Nahir and Danny Raz. Replication-based load balancing. Transactions on Parallel and Distributed Systems, pages 1–15, 2015.

[2] Danny Raz Amir Nahir, Ariel Orda. Replication-based load balancing. IEEE Transactions on Parallel and Distributed Systems, 27:494–507, feb 2016.

[3] G. Suseendran Anand Nayyar, Vikram Puri. Logarithmic spiral based local search in artificial bee colony algorithm. Balas V., Sharma N., Chakrabarti A. (eds) Data Management, Analytics and Innovation. Advances in Intelligent Systems and Computing, 839:513–525, 2018.

[4] Rajeshwar Singh Anand Nayyar. Ant colony optimization computational swarm intelligence technique. 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016.

[5] Rajkumar Buyya Anton Beloglazov, Jemal Abawajy. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Elsevier, 5:20068 – 20082, 2017.

[6] Ruilong Deng, Rongxing Lu, Chengzhe Lai, Tom H. Luan, and Hao Liang. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet of Things Journal, 3:1171–1181, 2016.

[7] Junjie Pang Gaochao Xu and Xiaodong Fu. A load balancing model based on cloud partitioning for the public cloud. TSINGHUA SCIENCE AND TECHNOLOGY, pages 34–39, 2013.

[8] Kenji Yoshihira Hui Zhang, Guofei Jiang and Haifeng Chen. Proactive workload management in hybrid cloud computing+b17. IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, 11:90–100, 2014.

[9] Gaolei Li, Jun Wu, Jianhua Li, Kuan Wang, and Tianpeng Ye. Service popularity-based smart resources partitioning for fog computing-enabled industrial internet of things. IEEE Transactions on Industrial Informatics, pages 4702–4711, 2018.

[10] Gaolei Li, Jun Wu, Jianhua Li, Kuan Wang, and Tianpeng Ye. Service popularity-based smart resources partitioning for fog computing-enabled industrial internet of things. IEEE Transactions on Industrial Informatics, pages 4702–4711, 2018.

[11] Gavriil Tzortzakis Michael Pantazoglou and Alex Delis. Decentralized and energy-efficient workload management in enterprise clouds. IEEE TRANSACTIONS ON CLOUD COMPUTING, 10:1–14, 2017.

[12] Ahmed Hamza Nour Mostafa, Ismaeel Al Ridhawi. An intelligent dynamic replica selection model within grid systems. Proceedings of the 8th IEEE GCC Conference and Exhibition, Muscat, Oman, pages 1 – 6, 2015.

[13] Aditya Brahmachari Pijush Kanti Dutta Pramanik, Saurabh Pal and Prasenjit Choudhury. Processing iot data: From cloud to fogits time to be down to earth. ResearchGate, pages 124–148, 2018.

[14] Pankaj Sharma Priti Singh. Load degree calculation for the public cloud based on cloud partitioning model using turnaround time. International Journal of Computer Science and Information Technologies, 2015.

[15] Deepak Puthal, Mohammad S. Obaidat, Priyadarsi Nanda, Mukesh Prasad, Saraju P. Mohanty, and Albert Y. Zomaya. Secure and sustainable load balancing of edge data centers in fog computing. IEEE Communications Magazine, 56:60–65, 2018.

[16] Sudeep Tanwar Shriya Kaneriya, Anand Nayyar, Jai Prakash Verma, Sudhanshu Tyagi, Neeraj Kumar, M. S. Obaidat, and Joel J P C Rodrigues. Data consumption-aware load forecasting scheme for smart grid systems. 2018 IEEE Globecom Workshops (GC Wkshps), 2019.

[17] Rimmy Yadav ; Avtar Singh Sidhu. Fault tolerant algorithm for replication management in distributed cloud system. 2015 IEEE 3rd International Conference on MOOCs, Innovation and Technology in Education (MITE), pages 78–83, 2015.

[18] Rajesh Kumar Simar Preet Singh, Anand Nayyar and Anju Sharma. Fog computing: from architecture to edge computing and big data processing. The Journal of Supercomputing, pages 1–36, 2018.

[19] Simar Preet Singh, Anand Nayyar, and Rajesh Kumar Anju Sharma. Fog computing: from architecture to edge computing and big data processing. The Journal of Supercomputing, pages 1–36, 2018.

[20] Anand Nayyar Sonal Sharma, Sandeep Kumar. Logarithmic spiral based local search in artificial bee colony algorithm. Duong T., Vo NS. (eds) Industrial Networks and Intelligent Systems. INISCOM 2018, 257:15–27, 2019.

[21] Stavros Souravlas and Angelo Sifaleras. Trends in data replication strategies: a survey. International Journal of Parallel, Emergent and Distributed Systems, pages 1 – 19, 2017.
[22] Angelo Sifaleras Stavros Souravlas. Binary-tree based estimation of file requests for efficient data replication. IEEE Transactions on Parallel and Distributed Systems, 28:1839 – 1852, 2017.
[23] Antonio Scala Stefano Sebastio. A workload-based approach to partition the volunteer cloud. IEEE Conference on Collaboration and Internet Computing, pages 2010–2018, 2015.
[24] Divya Mohandass Suguna R and Ranjani R. A novel approach for dynamic cloud partitioning and load balancing in cloud computing environment. Journal of Theoretical and Applied Information Technology, 62:662–667, 2014.
[25] Chenhua Shi Xiuli He, Zhiyuan Ren and Jian Fang. A novel load balancing strategy of software-defined cloud/fog networking in the internet of vehicles. China Communications, 13:140–149, 2016.
[26] Jui-Pin Yang. Elastic load balancing using self-adaptive replication management. IEEE Access, 5:7495–7504, 2017.
[27] Jui Pin Yang. On minimizing energy cost in internet-scale systems with dynamic data. IEEE Access, 5:20068 – 20082, 2017.
[28] Jui-Pin Yang. Intelligent offload detection for achieving approximately optimal load balancing. IEEE Access, 6:2169–3536, 2018.
[29] Hongmin Wang Yonglin Leng, Zhikui Chen and Fangming Zhong. A partitioning and index algorithm for rdf data. IEEE Access, pages 29836 – 29845, 2018.
[30] Juan F. P'erez Zhan Qiu. Evaluating replication for parallel jobs:an efficient approach. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS), pages 2288–2302, 2016.

Edited by: Anand Nayyar
Received: Mar 9, 2019
Accepted: Apr 3, 2019