Technique for Balanced Load Balancing in Cloud Computing Environment

Narayan A. Joshi
Department of Master of Computer Applications
Dharmsinh Desai University, Nadiad, India

***Abstract*—Resource sharing by means of load balancing in cloud computing environments helps for efficient utilization of cloud resources and higher overall throughput. However, implementation of poor load balancing algorithms may cause some virtual machines starving for additional cloud resources. Employing meagre crafted mechanism for priority-oriented load balancing may leave low-level priority virtual machines starving. We suggest an improved resource sharing mechanism for load balancing in the cloud computing environments. The suggested mechanism helps to provide efficient load balancing by avoiding starvation. In order to cater efficient load balancing, the proposed resource sharing technique takes respective virtual machines' priority levels into consideration. An implementation of the suggested load balancing algorithm in cloud environment provides reduction in waiting time of the starving virtual machines which are looking for additional resources in cloud platform. The implementation of our proposed algorithm has been deployed on a prototype cloud computing infrastructure testbed established with open source software OpenStack. The prototype cloud testbed is supported in backend by the open source CentOS Linux operating system's minimal setup. Experimental results of proposed load balancing mechanism in the prototype cloud computing infrastructure setup designate reduction in the waiting time of overloaded starving virtual machines. The proposed mechanism is beneficial to accomplish priority-oriented and starvation free resource sharing for load balancing in cloud computing environments. In future, the proposed technique can be further enhanced for implementing load balancing in collaborated cloud computing environments.**

Keywords*—*Cloud environment; resource sharing; load balancing; starvation; priority oriented resource allocation

I. INTRODUCTION

Cloud computing is one of the most important and outstanding innovations in the 21st century. Amongst several technological innovations done in the 21st century, the cloud computing has seen the expeditious adoption into not only IT sector but also IT enabled services sectors. Nowadays majority of the computing solutions which are being used in various societal service sectors directly or indirectly bank upon cloud computing-based services in the backend. Various parameters such as progressions in mobile communication infrastructure technologies, computation technologies, data storage technologies and telecommunication technologies have increased the span of cloud computing environments in several countries. Apart from that, the ongoing global COVID pandemic have influenced a vital role in cloud computing based digital transformation of numerous organizations in both IT sector and IT-enabled services sectors around the world [1]. Moreover, in the last decade, establishment and utilization of cloud computing based various service models for offering virtual classrooms, distance learning and e-learning platforms has grown exponentially across urban and rural areas in developed and developing countries [2].

Moreover, advancements in virtualization, networking and storage technologies have enabled resource sharing by means of load balancing in cloud computing environments. Cloud load balancing distributes workloads and computing infrastructure across multiple servers in order to provide advantages such as: increased scalability, redundancy, reduced downtime, increased performance, increased flexibility and higher throughput [3]. However, effective resource management plays a vital role in the overall success of utilization of computing solutions which are based on cloud computing enabled IT services. End-user satisfaction highly depends on the quality of service and adequate fulfilment of service level agreement. For attaining the objective of efficient resource management, cloud service providers extensively bank upon resource allocation solutions by means of load balancing. The mechanism of load balancing in cloud computing systems involves reorganization of workload allocation amongst other nodes in a cloud computing platform. Load balancing process encompasses continuous identification of overburdened and lightly loaded machines in cloud and then migrate the workload from overburdened machines to suitable lightly loaded machine in cloud. It helps to attain optimum utilization of cloud resources by safeguarding virtual machines from becoming not only overloaded but also underloaded or idle [4].

Several load balancing approaches have been proposed so far in literature and many of them are being practiced extensively nowadays in various public and private cloud computing environments. A novel load balancing technique has been presented in this paper. The technique presented in this paper helps to avoid starvation and reduce waiting time for overburdened machines in the cloud.

In cloud computing environments, machines work at different priority levels for example, some of the priority values could be: high priority, standard priority or low priority. Priority-based resource allocation in cloud computing environments is advantageous, however it may often cause starvation for machines which work at low priority level. A priority-oriented and starvation free resource sharing mechanism suitable for cloud-based load balancing has been suggested here. The proposed algorithm reports reduction in the waiting time. Reduction in the waiting time of overloaded

virtual machines for additional cloud resources helps not only improve quality of service and performance but also better return on investment.

The sequence of this research paper is set out as follows: our observations on the related literature survey have been discussed in the Section 2 whereas the proposed load balancing mechanism has been presented in the Section 3. The Section 4 and the Section 5 represent the implementation details, observations and outcomes respectively. Concluding remarks and further research scope have been given in the Section 6.

II. Literature Review

Load balancing is primarily concerned with how the workload is distributed among machines in cloud computing environments. Inadequate and poor management of cloud resources at the cloud service provider layer may turn into poor quality of service. Often, such a deterioration in quality of service may further result into termination of service level agreements also. Excerpts and our observations about the relevant literature survey in the area of load balancing has been presented here.

A mechanism for resource sharing [5] carries out migration of tasks from overburdened virtual machines to the underloaded virtual machines. The load balancing decisions are taken dynamically. However, the technique evenly operates on each virtual machine irrespective of the VM's priority level. While taking load balancing decisions, the technique does not take individual VM's priority level into consideration. Hence, the mechanism may result into starvation causing reduction in performance of the load balancing virtual machines.

A task scheduling algorithm suggested in [6] is based on ant colony optimization. The simulation-based algorithm focuses on reducing the average waiting time and works to optimize the makespan of the system. Another task scheduling technique available in [7] offers load balancing. The technique implements modified particle swarm optimization task scheduling called LBMPSO to schedule tasks in cloud computing environment. The load balancing mutation particle swarm optimization technique aims to minimize makespan and maximize utilization of cloud resources.

A machine learning based task scheduling mechanism in cloud computing environment is present in [8]. The load balancing technique takes advantage of K-means algorithm in order to create clusters of jobs and the resources available based on their operating characteristics and processing behavior.

A load balancing technique based on Dynamic Data Replication Algorithms is available in [9]. The technique works on three phase data replication algorithms. The initial two phases work on finding appropriate node for the sake of achieving load balancing. On other side, the third phase focuses on achieving better access improved load balancing by means of the dynamic duplication deployment scheme.

A novel load balancing technique available in [10] incorporate big-ip into an experimental framework. The technique incorporates secure socket layer, local traffic manager and access policy manger. The approach promises to offer high availability, redundancy and load balancing and data channel.

A load balancing technique based on Grey wolf optimization technique is available in [11]. The algorithm initially finds the unemployed or busy nodes. Then, the algorithm calculates such node's threshold and fitness function. The results obtained through the simulation indicate reduced cost and response time.

An integrated load balancing approach of Harries Hawks optimization and Pigeon inspired optimization algorithm is available in [12]. The cloudsim simulation-based load balancing technique ensures the optimal resource utilizations with task response time. A hybrid algorithm based on combining particle swarm optimization and genetic algorithm is available in [13]. The load balancing technique has a specific objective function.

A Weighted Signature based Load Balancing (WSLB) technique [14] aims on reducing users response time. The technique gathers the load assignment factor for each host and carries out mapping the virtual machines on the basis of the gathered factor value. A methodical review about load balancing techniques in software defined networks is available in [15]. The review mainly classified such techniques into deterministic and non-deterministic approaches. The paper discusses role, challenges and metric analysis in the domain of software defined networks. The study presented is based on single level classification.

In a load balancing approach based on software defined networking is available in [16]. The technique aims on optimizing traffic and data flow and reducing the delay. The resource sharing technique works on implementing FlowQoS mechanism like flow classifier and rate shaper with help of virtual queues.

An optimized resource management scheme known as MEMA is available in [17]. It splits the actual mechanism into two parts: load balancer for normal requests, and load balancer for urgent requests. The mechanism emphasizes on improving task allocation by means of providing quick services and servers to urgent requests. The technique works on lowering waiting time and maximizing fairness with help of a modified round robin technique. The technique entertains all nodes equally without discriminating their priority values.

A load balancing mechanism existing in [18] offers resource sharing suitable for cloud computing platform. A recommended load balancer software component EfficientLoadBalancer operates in coordination through various cooperating daemon threads: LoadBalancer, ManageState, OverLoadedVM and underLoadedVM. The load balancer operates differently in line with the various states of virtual machines. The technique maintains two values for representing VM's state: OVERLOAD_PASSIVE, UNDERLOAD_PASSIVE. The load balancer functions on the basis of shifting workload from the overburdened nodes to the unburdened nodes in cloud. However, the load balancer does not distinguish among the respective virtual machines' priority levels by operating all nodes equally for load balancing.

An improved technique for resource sharing has been suggested in [19]. The technique offers resource sharing in cloud computing environment and it balances workload as per the priority value of virtual machines. The load balancer module PriorityBasedLoadBalancer operates in coordination through various cooperating threads in background: PBLoadBalancer thread, ManageState thread, PBOverloaded VM thread and the thread PBUnderloadedVM thread. The priority-based load balancer module functions on transferring workload between virtual machines of the same priority level. However, in long run the load balancer may cause starvation in load balancing requests which involve low priority virtual machines.

A load balancing framework available in [20] works on attaining better performance on the parameters makespan and cost. The suggested framework is based on hybridization of heuristic technique with metaheuristic algorithm. The framework focuses on deadline constraints and improved resource provisioning in the Cybershake and Ligo workflows execution domain.

A load balancing technique based on genetic pso algorithm is available in [21]. The technique responsible for sharing resources in cloud computing, works over heterogeneous cloud infrastructure. The hybrid genetic pso based task distribution algorithm mainly works on optimizing cost of resource allocation and makespan. The technique works on improving load balancing of the workflow application over the heterogeneous resources in the cloud environment.

A honeybee algorithm-based load balancing technique is present in [22]. The task allocation technique aims to minimization of service makespan on the cloudsim and the workflow. The dynamic resource allocation mechanism works for both dependent and independent jobs. Moreover, the technique addresses to the task priorities and not the VM priorities.

An IMLDB mechanism for resource sharing [23] is based on Improved Modified Distribution Load Balancing and it aims to yield low cost for task migration. The mechanism eyes on the two facets overloading and under loading by means of maintaining profit of the capital gain during the process of task migration in the cloud computing environment.

Intercloud load balancing techniques are present in [24, 25]. The techniques work on resource sharing in collaborated cloud computing environments. The suggested techniques eye on offering not only optimized resource utilization but also continuous availability of cloud resources to stakeholders. However, the mechanisms do not discriminate among the respective virtual machines' priority levels. They treat all virtual machines at the same level.

A modified genetic based algorithm is available in [26]. The technique works on the optimization problem and strives to determine the fittest machines which are associated with various data centers for allocation of cloudlets into appropriate virtual machines. The technique reports lesser execution time consumption by the cloudlets.

In a view of the literature study presented above so far in this section, it is felt that some of the load balancing mechanisms are restricted to specific computation platforms only. Apart from it, few load balancing mechanisms focus on relocating entire overburdened virtual machines to some another host on the cloud infrastructure. Some of the load balancing techniques are static in nature, such techniques are inappropriate for the dynamic load balancing environments. On the other side, some available simulation-based load balancing mechanisms do require prior knowledge about various time variables such as the service time and the arrival time of workload. Often, there exist some closed source load balancing solutions of which no source code is available openly. Hence, it is difficult to extend such closed source solutions on open-source cloud computing frameworks such as the OpenStack. Whereas some of the existing open-source resource sharing techniques fail in controlling starvation problems arising due to their inability to tackle priority-oriented load balancing in particular way.

Hence, virtual machines' priority oriented and starvation free mechanism for dynamic resource sharing in cloud computing environments has been presented here. The load balancing technique presented here aims to overcome the starvation problem often faced by overloaded virtual machines. Furthermore, the technique takes the priority level of virtual machines before taking load balancing decisions. The suggested mechanism has been presented in the following Section 3.

III. Materials and Method

The resource sharing mechanism suggested in [18] provides load balancing functionality in cloud computing environment. However, the technique operates every virtual machine equally without discriminating the virtual machines according to their respective priority levels.

On the other side, the resource sharing technique available in [19] offers priority-based load balancing in cloud computing environment. The technique schedules resource sharing on basis of the preassigned priority value of the virtual machines. However, the technique may cause starvation for certain low priority virtual machines. Situations may arise such that the low priority and the standard priority virtual machines which are overloaded might not get necessary attention due to higher number of resource sharing demands made by the high priority virtual machines.

Hence, the constraints of the work available in [18] and [19] motivated us to extensively work the resource sharing problem. An extended mechanism of priority-based dynamic load balancing approach [19] for load balancing in cloud computing environments has been presented here in order to give adequate attention to such starving overloaded virtual machines. The block diagram our proposed methodology has been described in the Fig. 1. Working of various components of our proposed load balancer PBImprovedLoadBalancer have been presented here:

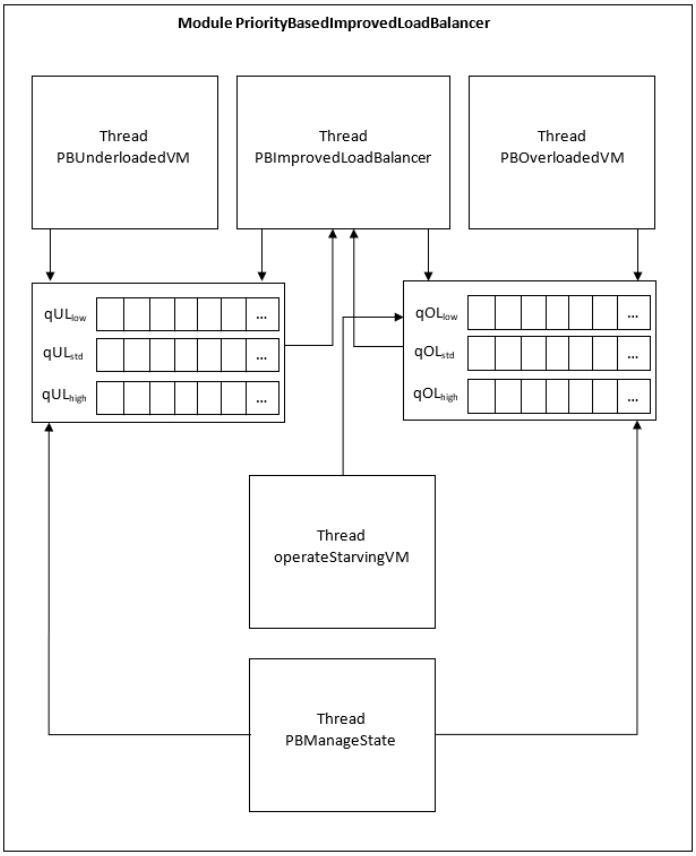

Fig. 1. Block Diagram of Proposed Load Balancing Technique Priority based Improved Load Balancer.

1) The mechanism maintains various state values for representing current state of a particular virtual machine. The AVAILABLE and UNAVAILABLE states indicate current availability and unavailability respectively of a particular virtual machine for load balancing. The UL_PASSIVE state is used to signal that a particular virtual machine was previously underloaded; but now it has already been considered for assigning additional workload and hence no more workload should be assigned to it. The OL_PASSIVE state is used to signal that a particular virtual machine was previously overloaded; but the virtual machine is currently under the process of unloading. The extended mechanism presented here offers one more state of virtual machines: OL_STARVING. The newly introduced state helps the mechanism to mark the overloaded virtual machines which are striving since considerable amount of time for availing additional resources by means of resource sharing in the cloud.

2) The three priority levels LOW, STANDARD and HIGH have been employed for representing current priority of machine. The mechanism permits resource sharing amongst the virtual machines at the same priority level.

3) Based on the current workload of the concerned virtual machines, the thread thread_PBIUnderloadedVM puts the virtual machines with the underutilized resources in one of the respective priority queues: qULhigh, qULstd or qULlow. The daemon thread keeps fetching underutilized virtual machines and drops them in the appropriate queue. For sharing the resources, the load balancer picks virtual machine from relevant priority queue of underloaded virtual machines. Empty queue indicates unavailability of underloaded virtual machine.

4) Based on the current workload of virtual machines, the thread PBIOverloadedVM puts the overloaded virtual machines in one of the appropriate priority queues: qOLhigh, qOLstd or qOLlow. The daemon thread keeps fetching overloaded virtual machines and drops them in the appropriate queue. For obtaining the cloud resources, the load balancer picks virtual machines from relevant priority queue of overloaded virtual machines.

5) The resource sharing task is carried out by the thread PBImprovedLoadBalancer. The operation takes place between the overburdened virtual machine and the under burdened virtual machines at the same priority levels. The thread runs continuously. The thread may sleep for a while if there are no additional requirements available in the relevant overload queue.

6) In priority-based cloud computing environment, the suggested technique helps stay away from starvation. The technique periodically evaluates waiting time of concerned virtual machines in the LOW priority queue. The technique treats the starving LOW priority virtual machines with help of the continuously working thread operateStarvingVM.

7) To avoid multiple consecutive resource allocations on the same virtual machine, the load balancer module switches the state of concerned virtual machines to UL_PASSIVE state. Arrangement is such that the suggested resource sharing mechanism does not consider UL_PASSIVE virtual machines eligible for new load balancing requests. However, after completion of certain buffer time, such virtual machines should be automatically toggled back to the AVAILABALE state. Only AVAILABLE state virtual machines are considered eligible for making additional resource demands and sharing of resources.

8) A daemon thread PBImanageState is assigned the task of state management of virtual machine. It periodically keeps checking if the VM is currently passive or not. If the VM is found to be passive, then the thread toggles the VM state to AVAILABLE.

9) Finally, the enhanced module for priority-based resource allocation starts by initializing all time variables and all priority queues. The variables TSu, TSo and TSt represent sleeping time of various daemon threads PBIUnderloadedVM, PBIOverloadedVM, operateStarvingVM respectively.

*10)*Then the load balancer module launches various collaborating daemon threads PBIOverloadedVM, PBI UnderloadedVM, operateStarvingVM and PBEnhancedLoad Balancer. The thread PBIManageState is responsible for VM state management. It also is launched during the starting phase of our load balancer module PBImprovedLoadBalancer.

*11)*The module maintains an enumerated data structure variable v_priority for maintaining the priority-level values of all virtual machines. At a time, the v_priority variable can have any one of the possible three VM priority values: LOW, STANDARD and HIGH.

*12)*The module maintains an enumerated data structure variable v_state for maintaining the current state values of all virtual machines. At a time, the v_state variable can have any one of the possible five state values: UNAVAILABLE, AVAILABLE, UL_PASSIVE, OL_PASSIVE and OL_ STARVING.

Key segments of the recommended resource sharing algorithm with necessary explanation have been presented here:

```
Module PBImprovedLoadBalancer
{
// Algorithm for starvation free and priority oriented improved
// sharing of cloud resources
    enum v_priority {LOW, STANDARD, HIGH};
    // possible VM priority values

    enum v_state {UNAVAILABLE, AVAILABLE,
    UL_PASSIVE, OL_PASSIVE, OL_STARVING};
    // Virtual machine state values: unavailable, available,
    // underload passive, overload passive, overload starving

    int TSt, TSo, TSu; //sleeping time for thread
```

```
  struct VMachine
  // Holds metadata of a VM
  {
      v_priority vm_priority;
      v_state vm_state; // current state
      // VM's resources' information
      float vm_load;
      unsigned int ncores;
      unsigned long tot_mem, free_mem;
      float bandwidth;
      // Timestamp when VM's state was set as passive
      unsigned long passive_set_time;
      unsigned long WTUnder, WTOver;
      // Threshold for waiting time in queues related to
      // overloaded and underloaded virtual machines
      char vmIP[40];
      …
      // getter and setter methods for accessing and
      // setting VMachine structure members.
      …
  }
  // Starvation time threshold
  unsigned long OLVmStarvationThresholdTime;

  // Queues for maintaining underloaded VMs
  Queue <VM*> qUL_low, qOL_high, qUL_std;
  // Queues for maintaining overburdened VMs
  Queue <VM*> qOL_std, qUL_high, qOL_low;

  // Keep detecting underloaded virtual machines and
  // populate them in respective priority queue
  Thread: thread_PBIUnderLoadedVM
  {
    VM* pVM_U = null;
    void fetch_PBIUnderloaded_VM(){
    //Thread method
    while(true) {
      pVM_U = null;
      //Find out underloaded VM
      pVM_U = determine_underloaded_VM();
      if(qUL<pVM_U -> vm_priority>.find
         (pVM_U) ||
pVM_U.passive_set_time < WTUnder)
              continue;
      if(!pVM_U) {
          // At present there is no such VM.
          // So, sleep thread for TSu sleep time.
          sleep(TSu); continue;
        }
    qUL<pVM_U->
 vm_priority>.append(pVM_U);
    // Found underloaded VM.
    // So, append it to relevant queue.
    }//while
    }//End of Thread method
  }
```

```
// Keep detecting overloaded virtual machines and
// populate them in respective priority queue.
Thread: thread_PBIOverLoadedVM
{
  VM* pVM_O = null;
  void fetch_PBIOverloaded_VM(){
  //Thread method
  while(true) {
    pVM_O = null;
    // Find out overloaded VM
    pVM_O = determine_overloaded_VM();
    if(qOL<pVM_O->vm_priority>.find(pVM_O) ||
    pVM_O.passive_set_time<WTOver)
        continue;
    if(!pVM_O) {
    // At present there is no such VM
    // So, sleep thread for TSo sleep time
      sleep(TSo); continue;
    }
    qOL<pVM->vm_priority>.append(pVM_O);
    // Found overloaded VM.
    // So, append it to relevant queue.
  }//while
  }//End of thread method
}//Thread

// Procedure for starvation free and priority oriented load
// sharing
Thread: thread_PBImprovedLoadBalancer
{
  VMachine* pVM_O, pVM_U;
  void PBImprovedLoadBalance()
  {
    Thread sleep if there are no overloaded VMs
    Thread sleep if there are no resource sharing offers
    while(true) {
    pVM_U=qUL<high,std,low>.fetch()
    if(!pVM_U)
      // VM for resource sharing is unavailable at present.
      // So, search again.
      continue;
    if(pVM_U->timeSincePassive() <
              MaxULTimeThreshold
      || !pVM_U-> isUnderloaded()
      || pVM_U->vm_state== UL_PASSIVE)
      // Found VM but it is still passive or it has no
      // sharable resources
      continue;
    pVM_O=qOL<pVM_U.priority>.fetch()
    if(!pVM_O)
    // Resource requirements unavailable at present.
    // So, search again.
      continue;
    if(pVM_O->timeSincePassive() <
              MaxOLTimeThreshold
      ||pVM_U->isOverloaded()
      ||pVM_O->vm_state==OL_PASSIVE)
      // VM is still passive or it has no resource
      // requirements.
      continue;
    if(pVM_U->vm_state == AVAILABLE &&
         pVM_O->vm_state == AVAILABLE){
      // Both source and target VMs are available.
      // So, carry out resource sharing.
      // Make both VMs passive for further load
      // balancing requests and to protect them from
      // instantaneous overburdening.
      pVM_U->vm_state=UL_PASSIVE;
      pvm_O->vm_state=OL_PASSIVE;

      set passive_set_time for pVM_O
      set passive_set_time for pVM_U
      // Remove both VMs from respective wait queues
      qUL<pVM_U.priority>.remove()
      qOL<pVM_O.priority>.remove()
      // Load balance
      balance(pVM_O,pVM_U)
      …
    }
  } // function PBImprovedLoadBalancer.
  } //while
} // thread thread_PBImprovedLoadBalancer.
// Keep detecting the starving virtual machines which are
// waiting for additional resource requirements in queue.
Thread: thread_operateStarvingVM
{
  void operateStarvingVM()
  {
    while(true){
    …
    pVM_O = qOL<low>.fetchLast()

    if(pVM_O->timeSincePassive() >
        OLVmStarvationThresholdTime) {
        //VM is starving, so operate it.
       qOL<low>.shiftToFirst(pVM_O)
        sleep(TSt);
        //Sleep thread for TSt Starvation sleep time.
        …
    }
    }// while
  }// function operateStarvingVM
}// thread thread_operateStarvingVM

// Keep managing VM state
Thread: thread_PBImanageState
{
  void vmStateManager() {
    …
    for all VMs: pVM
    if(pVM-> vm_state = UL_PASSIVE &&
      pVM->timeSincePassive()>=
      pVM->WTUnder)
      ||
```

```
        (pVM-> vm_state = OL_PASSIVE &&
        pVM->timeSincePassive()>=
        pVM->WTOver)
        // Time to remain in the passive state for a
        // particular VM is over. So, now make it
        // AVAILABLE for load balancing.
        reset pVM's passive_set_time
        // Make pVM as AVAILABLE for load balancing.
        pVM->vm_state = AVAILABLE;
        …
        }// function thread_PBIManageState
    }//thread thread_PBIManageState

    // Launch module now.
    void start()
    {
      …
      Initialize times: TSu, Tso, TSt
      Initialize time: OLVmStarvationThresholdTime

      Initialize queue: qOL<low,std,high>
      Initialize queue: qUL<low,std,high>

      Spawn thread: thread_PBIOverLoadedVM

      Spawn thread: thread_PBIUnderLoadedVM

      Spawn thread: thread_PBImprovedLoadBalancer

      Spawn thread: thread_operateStarvingVM

      Spawn thread: thread_PBImanageState

      …
    }// function start
}//module PBImprovedLoadBalancer
```

The start() function in the suggested module PriorityBasedImprovedLoadBalancer starts with initializing various time intervals which are meant for making the thread sleep for certain time intervals. The module also sets the starvation time threshold in waiting queue for the overloaded virtual machines. Finally, the start() function launches the collaboratively working threads. The parallelly running threads thread_PBIOverLoadedVM and thread_PBIUnderLoadedVM keep finding those virtual machines which are in shortage of resources and the virtual machines which have excessive unutilized resources go wasted respectively. The priority based improved load balancer thread thread_PBImprovedLoad Balancer executes resource sharing. The collaboratively working thread thread_operateStarvingVM takes care of starving virtual machines which are waiting in queue.

IV. Implementation

The prototype cloud environment testbed was established on physical server and the entire setup was made run as suggested in the [19]. The private cloud computing environment was setup over a minimal setup of the open-source CentOS Linux operating system platform. In order to facilitate the Infrastructure as a Service (IaaS) by means of offering cloud-based instances which are available on demand, the open-source cloud computing environment was established by installing OpenStack on top of the CentOS Linux platform.

V. Results and Discussion

The mechanism begins with setting and initializing values of various data structures such as sleeping time of various threads, respective queues for maintaining information about overloaded and underloaded virtual machines. After initializing the data structures and queues, the algorithm spawns the collaborating daemon threads: PBImanageState, PBIOver LoadedVM, PBImprovedLoadBalancer PBIUnderLoadedVM and operateStarvingVM.

The daemon thread thread_operateStarvingVM keeps watching and operating the starving virtual machines. The Table I shows comparison of results obtained using our proposed method with the results of existing technique. For the low priority overloaded virtual machines, the experimental results have been presented in a Table I. The Table I represents the waiting times of various starving virtual machines in absence and presence of our suggested thread thread_operateStarving VM respectively. The function of the daemon thread thread_operateStarvingVM is to efficiently manage the starving overloaded virtual machines with a motive to reduce their waiting time.

The column A in the Table I represents the observed waiting time value of the overloaded virtual machines in absence of our suggested daemon thread thread_operate StarvingVM. The average of the waiting times mentioned in the column A is 9.62 milliseconds for such overloaded virtual machines [19].

The column B in the Table I represents the observed value of waiting time of the overloaded virtual machines in presence of our suggested daemon thread thread_operateStarvingVM. However, in presence of our suggested collaborative daemon thread thread_operateStarvingVM, the average value of the observed waiting times mentioned in the column B is found to be reduced to 7.1 milliseconds, which clearly designates performance improvement by means of reduction in waiting time of overloaded virtual machines in waiting queue.

TABLE I. Table Showing Waiting Times of Virtual Machines in the Waiting Queue in Absence and Presence of the Thread Thread_operateStarvingVM Respectively

Virtual Machine (IP address)	**Waiting time (milli seconds)**	
	A. In absence of the thread_ operateStarvingVM	***B. In presence of the thread_ operateStarvingVM***
192.168.10.2	8.9 ms	6.4 ms
192.168.10.3	10.2 ms	7.6 ms
192.168.10.4	10.4 ms	7.9 ms
192.168.10.5	9.4 ms	6.9 ms
192.168.10.7	9.2 ms	6.7 ms

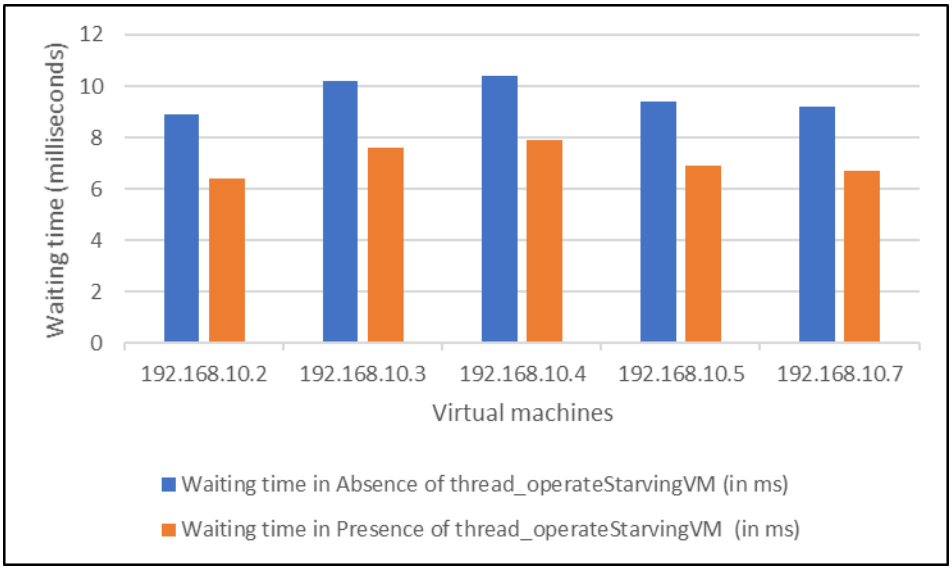

Fig. 2. Chart of Comparison of Waiting Times of Virtual Machines in Absence and Presence of the Thread thread_operateStarvingVM.

Moreover, a chart shown in a Fig. 2 also designates significant reduction in the waiting time of all concerned overloaded virtual machines for availing additional cloud resources for load balancing.

Such a reduction in the average waiting times is extremely beneficial to the low priority overloaded virtual machines in the cloud. The dropout in waiting time of a concerned starving overloaded virtual machine in a respective priority queue helps it to attain the required additional resources from cloud in lesser time. Hence, faster availability of additional cloud resources will cause faster execution of tasks. Quicker execution of tasks results into increased overall system performance and optimized utilization of cloud resources.

VI. Conclusion

An enhanced mechanism for priority-oriented resource sharing for cloud computing platform has been suggested in this paper. The mechanism prevents resource requirements on low priority virtual machines from starvation. Implementation of proposed algorithm clearly designates reduction in waiting time of concerned virtual machines. The technique is helpful in attaining efficient resource utilization and improved performance. The obtained results undoubtedly reveal reduction in the average waiting time of overloaded virtual machines in the waiting queue. Hence the technique helps overcome starvation. Thereby, the proposed technique helps achieving improved resource sharing for low priority virtual machines in cloud computing environment.

In future, this technique can be extended further on collaborated cloud computing environments for attaining improved resource utilization and getting better return on investment. Moreover, the suggested technique can be explored further for studying security aspects.

References

[1] Alashhab Z, Anbar M, Singh M, Leau Y, Al-Sai Z, Alhayja S. Impact of coronavirus pandemic crisis on technologies and cloud computing applications, Journal of electronic science and technology. 2021; 19.

[2] Joshi N. Performance-centric cloud-based e-learning, The IUP Journal of information technology. 2014; 10(2).

[3] Afzal S, Kavitha G. Load balancing in cloud computing – A hierarchical taxonomical classification, Journal of cloud computing. 2019; 8.

[4] Mishra SK, Sahoo B, Parida PP. Load balancing in cloud computing: A big picture. Journal of king saud university – computer and information sciences. 2020; 32(2).

[5] Joshi N, Choksi DB, Kotecha K, Pandya S. Implementation of novel load balancing technique in cloud computing environment. International conference on computer communication and informatics. 2018.

[6] Amit D, Dinesh R. Design a novel technique for load balancing in cloud computing environment. Solid State Technology. 2021. 64(2).

[7] Arabinda P, Sukanata Kishoro B. A novel load balancing technique for cloud computing platform based on PSO. Journal of king saud university – computer and information sciences. 2020; In Press.

[8] Shivaprasada K, Sangameshwar, Rajesh S, Swasthika T. Machine learning aided scheduling on cloud computing, International journal of emerging trends in engineering research. 2020; 8(9).

[9] Hsieh HC, Ching M. The incremental load balance cloud algorithm by using dynamic data deployment. Journal of grid computing. 2019; 17.

[10] Bholanath M, Rajesh B, Sandip R. A novel approach to load balancing and cloud computing security using SSL in IaaS environment. International journal of advanced trends in computer science and engineering. 2020; 9(2).

[11] Sefati S, Mousavinasab M, Zareh Farkhady R. Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: performance evaluation. Journal of supercomputing. 2021; In press.

[12] Poornima G, Radhamani A. A hybrid meta-heuristic for optimal load balancing in cloud computing. Journal of grid computing. 2021; 19.

[13] Dhiraj K, Vijay D. Performance evaluation of new hybrid approach of load balancing in cloud computing. Design engineering. 2021; 2021(5).

[14] Ajit M, Vidya G. VM level load balancing in cloud environment. International conference on computing, communications and networking technologies. 2013.

[15] Neghabi AA, Navimipour NJ, Hosseinzadeh M, Rezaee A. Load balancing mechanisms in the software defined networks: a systematic and comprehensive review of the literature. IEEE access. 2018; 6:14159–14178.

[16] Gokilabharathi R, Deepalakshmi P. Efficient load balancing to enhance the quality of service (QOS) in Software Defined Networking (SDN). International conference on trends in electronics and informatics. 2018.

[17] Manasser S, Alzghoul M, Mohmad M. An advanced algorithm for load balancing in cloud computing using MEMA technique. International journal of innovative technology and exploring engineering. 2019; 8 (3).

[18] Joshi N. Efficient load balancing in cloud computing. Research review international journal of multidisciplinary. 2019; 4(6).

[19] Joshi N. Priority based mechanism for resource sharing in cloud. International journal of innovative technology and exploring engineering. 2020; 9(3).

[20] Kaur A, Kaur B. Load balancing optimization based on hybrid heuristic-metaheuristic techniques in cloud environment, Journal of king saud university – Computer and Information Sciences. 2019.

[21] Manasrah A, Ali H. Workflow scheduling using hybrid GA-PSO algorithm in cloud computing, Wireless communications and mobile computing. 2018; 2018.

[22] Gopinath G, Vasudevan SK. A novel improved honey bee based load balancing technique in cloud computing environment. Asian journal of information technology. 2016; 15(9).

[23] Afzal S, Kavitha G. Optimization of task migration cost in infrastructure cloud computing using IMDLB algorithm. International conference on circuits and systems in digital enterprise technology. 2018.

[24] Joshi NA. Optimized Mechanism for Resource Sharing in Cloud. International journal of engineering and advanced technology. 2019. 9(2).

[25] Joshi N. Performance centric model for resource sharing in cloud. Research review international journal of multidisciplinary. 2018; 3(5).

[26] Swarnakar S, Kumar N, Kumar A, Banerjee C. Modified genetic based algorithm for load balancing in cloud computing. International conference for convergence in engineering. 2020.