International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-3, January 2020

Priority Based Mechanism for Resource Sharing in Cloud

Narayan A Joshi

Abstract*: Cloud computing technologies are getting matured day by day. Revolutions in underlying software, virtualization and hardware technologies related to storage, processing and computing technologies has helped cloud computing service providers to win trust of concerned stake holders. However, the exponentially increasing demand of cloud based resources has made task of resource management and utilization more and more challenging. A novel load balancing technique in cloud computing environment is presented in this paper. The virtual machines are implemented on an open source cloud computing platform on open source operating system. A virtual machines' priorities based load balancing approach presented here indicates improvement in overall waiting time for load balancing. The mechanism prioritizes load balancing on same priority level virtual machines or lower priority level virtual machines.

***Index Terms*: cloud computing, load balancing, priority based resource allocation, task migration.**

I. INTRODUCTION

Modernization in virtualization, storage and Communication Technologies have directed Cloud computing towards new heights. Moreover, the paradigms of forth generation (4G) and fifth generation (5G) mobile communication technologies have started giving momentum in utilization of cloud based services [1]. On other side, improvement in back end cloud computing technologies and reduced prices for deployment prices are encouraging more and more users for adopting cloud based service deployment. Moreover, to meet the exponential growth of cloud computing based services, nowadays cloud service providers have started coming together for joining hands for establishment of collaborated cloud computing environments [4].

However, the exponential growth in demand of cloud computing is causing challenges for service providers for offering round of the clock quality and uninterrupted services. On other side, service consumers also need to see about efficient utilization of procured cloud based resources. Situations may arise such that few procured virtual machines may be heavily loaded facing pending workload while other procured virtual machines may be idle causing wastage of resources.

Revised Manuscript Received on January 30, 2020.
***** Correspondence Author
N. A. Joshi*, Department of MCA, Dharmsinh Desai University, Nadiad, India.

Non uniform utilization of cloud based resources and virtual machines results into imbalanced return on investments and thereby violations in Service Level Agreements. Hence, appropriate resource allocation and management is required for performance centric resource utilization. Optimized resource management and utilization helps in energy saving, improved resource availability and satisfactory execution of service level agreements. An improved load balancing strategy in cloud computing environments is suggested here. A study on related load balancing work is presented in section 2. Section 3 describes the suggested novel load balancing technique. Implementation scenario is presented in section 4. Section 5 covers technique's working behavior.

II. RELATED WORK AND CHALLENGES

M. Kumar et al. have suggested a priority based approach for selection of relevant virtual machine in cloud computing environment. The scheduling algorithm works as per the tasks' priorities and selects appropriate virtual machine based on three categorization levels. The suggested cloudsim simulation based algorithm works on virtual machines in a datacenter configured for Amazon EC2 [3].

An implementation of load balancing technique is available in [7]. The load balancing technique is based on DSBP and works with cloud collaboration for logistics. The simulation based approach focuses on load balancing of different data centers for information processing at various level of logistics. The technique may be useful in inventory management and vehicles' location management.

A three phase Dynamic Data Replication algorithms have been suggested in [2]. Objective of the overall mechanism is for optimized resource deployment. For the first two phases, the mechanism determines suitable service nodes for balancing workload of service nodes. The algorithm helps in enhancing availability, access efficiency and hierarchical load balancing. An advanced load balancing technique is available in [9]. The technique is based on static variables. The technique works on improvement on the Weighted Round Robin scheduling algorithm for the sake of reducing response time and maximize fairness concept. The mechanism focuses on improvement of workload distribution through servers and quick services to urgent requests. A Capacity Based Deadline Aware Load Balancing approach is available in [6]. The heuristic based simulative approach works on identifying virtual machine with respect to deadline constraints and overall cost for running tasks on virtual machines.

The approach also works on various QoS parameters such as turnaround time, response time and utilization. A dynamic round robin algorithm based scheduling approach has been suggested in [8]. The technique works on determining dynamic time quantum for individual service request in every round. Moreover, the technique calculates the process's priority component.

A load balancing technique available in [5] works for tasks in cloud computing environment. The suggested `EfficientLoadBalancer` module works in association with four collaborating threads: `thread_underLoadedVM`, `thread_OverLoadedVM`, `thread_LoadBalancer` and `thread_ManageState`. The technique works on various virtual machine states such as `UNDERLOAD_PASSIVE` and `OVERLOAD_PASSIVE`. The mechanism works on transferring workload from heavily loaded virtual machines to the virtual machines which have lighter workload.

Such virtual machine states help in keeping track of current state of designated participating virtual machines for load balancing. The mechanism keeps track of time interval for putting concerned load balancing virtual machine in one of these two states. The process of keeping track of the said time interval helps in bringing the concerned virtual machines back into `ACTIVE` state for making it available for making it available in load balancing.

Couple of techniques are not available open source, hence difficult to adopt and implement by others. On other side few of them are available in the form of simulations based algorithms. Some of the load balancing techniques are based on genetic algorithms, weighted and quality of service parameters. Some resource management techniques work on reducing the overall cost and turnaround time. Few of the load balancing technique don't differentiate among virtual machines' capacities and treats all virtual machines at the same level.

III. MECHANISM

The technique available in [5] works on load balancing by means of maintaining two queues: qUnderloaded and qOverloaded. The queues help in details of available under loaded and overloaded virtual machines respectively. However, the technique does not support priority based load balancing. Situations may arise such that the low priority overloaded virtual machines might obtain more load balancing opportunities over high priority overloaded virtual machines. Such situations may cause concerned long lasting high priority virtual machines starve for uncertain duration. An optimized mechanism of the load balancing technique available in [5] is suggested here. Block diagram is shown in Figure 1.

1. The mechanism suggested here works on two more virtual machines states: UNAVAILABLE and AVAILABLE. Such states help the mechanism in maintaining the unavailable and available virtual machines for the aspect of load balancing. Moreover, the mechanism works on

Some of the foremost steps of the suggested algorithm are presented here.

priority of the virtual machines which are AVAILABLE for intra cloud load balancing.

2. The mechanism supports and facilitates three categories of priority levels: LOW, STANDARD and HIGH. The low prioritized virtual machines can utilize other virtual machines in the same category for load balancing. Accordingly, the standard overloaded virtual machines can balance workload with other virtual machines having standard and low prioritized virtual machines.
3. On other side, the technique allows the highly prioritized overloaded virtual machines to utilize the power of other virtual machines' in the cloud environment.
4. The priority based thread PBUnderloadVM keeps on determining under loaded virtual machines and filling information in the respective queues: qUL_{low}, qUL_{std}, qUL_{high}.
5. Likewise, the priority based thread PBOverloadVM keeps on determining overloaded VMs and filling information in the respective queues: qOL_{low}, qOL_{std}, qOL_{high}.
6. The thread PBLoadBalancer carries out the load balancing task according to the priority levels of concerned overloaded and underload virtual machines.

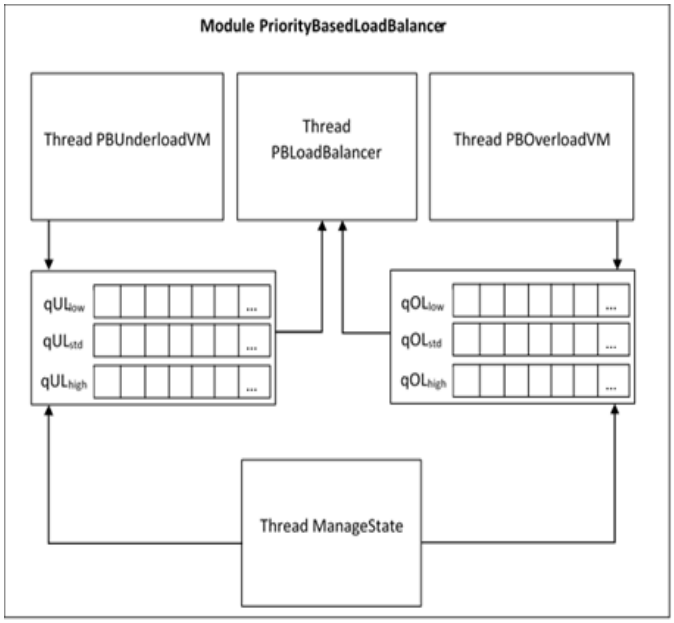

Figure 1: Priority Based Load Balancer – Block diagram

```
Module PriorityBasedLoadBalancer      // Algorithm for priority base Load balancing
{
  enum priority {LOW=0, STANDARD=1, HIGH=2};  //virtual machine's priority

  //virtual machine's state
  enum state {UNAVAILABLE=0, AVAILABLE=1, UNDERLOAD_PASSIVE=2, OVERLOAD_PASSIVE=3};

  int TSu, TSo; //Thread sleeping time

  struct VM // Data structure to hold virtual machine's recent information
  {
    priority vm_priority;
    float vm_load; unsigned short ncores;
    unsigned long passive_set_time, tot_mem;
    unsigned long free_mem, bandwidth;
    /* Waiting time threshold values */
    unsigned long WTOver, WTUnder;
    state vm_state; // vitual machine's current state
    char vmIP[40]; // address of virtual machine
    …
  };
  // Various priority queues to hold respective virtual machines
  Queue<VM*> qUL_low, qOL_low, qUL_std, qOL_std, qUL_high, qOL_high;

  // Algorithm to detect priority based underloaded virtual machines
  Thread: thread_PBUnderLoadedVM(qUL<low,std,high>)
  {
    VM* pVM_U;
    void fetch_PBUnderloaded_VM(qUL<low,std,high>) { //thread method
      while(true) {
        pVM_U = null;
        pVM_U =determine_underloaded_VM(qUL<low,std,high>);
        if(qUL<low,std,high>.find(pVM_U)|| pVM_U->load > WTUnder)
          continue;
        if(!pVM_U) { // Concerned VM not found
          sleep(TSu); continue;
        }
        qUL<low,std,high>.append(pVM_U); //append VM to appropriate underloaded queue
      }
    };
  }; //end of thread_PBOvderLoadedVM

  // Algorithm to detect priority based overloaded virtual machines
  Thread: thread_PBOverLoadedVM(qOL<low,std,high>)
  {
    VM* pVM_O;
    void fetch_PBOverloaded_VM(qOL<low,std,high>) { //thread method
      while(true) {
        pVM_O = null;
        pVM_O = determine_overloaded_VM(qOL<low,std,high>);
        if(qOL<low,std,high>.find(pVM_O) || pVM_O->load < WTOver)
          continue;
        if(!pVM_O) { // Concerned VM not found
          sleep(TSo); continue;
        }
        // append VM to appropriate overloaded queue
```

```
      qOL<low,std,high>.append(pVM_O);
    }
  };
}; //end of thread_PBOverLoadedVM(qOL<low,std,high>)

// Algorithm for Priority Based Load Balancing
Thread: thread_PBLoadBalancer {
  VM* pVM_O, pVM_U;
  while(true) {
    pVM_O = qOL<low,std,high>.fetchAndRemove();

    if(!pVM_O) continue;

    if(pVM_O.timeSincePassive()>MaxOverloadThresholdTime ||
       !pVM_O.isOverloaded() || pVM_O.state == OVERLOAD_PASSIVE)
      continue;

    pVM_U = qUL<low,std,high>.fetchAndRemove();
    if(!pVM_U) continue;

    if(pVM_U.timeSincePassive()>MaxUnderloadThresholdTime ||
       !pVM_U.isUnderloaded() || pVM_U.state == UNDERLOAD_PASSIVE)
      continue;

    if(pVM_O.priority == pVM_U.priority   ||
       pVM_U.getPriorityQueue().length()<=1 &&
       pVM_U->state != UNAVAILABLE &&
       pVM_O->state != UNAVAILABLE) {
      pVM_U->vm_state = UNDERLOAD_PASSIVE;
      pVM_O->vm_state = OVERLOAD_PASSIVE;
      set passive_set_time for Pvm_U and pVM_O
      remove pVM_O from respective priority OL-queue
      remove PVM_U from respective priority UL-queue
      balance(pVM_O,pVM_U);
      …
    }
  }
}; //end of thread_PBLoadBalancer

// Algorithm for managing current state of virtual machine
Thread: thread_ManageState {
  …
  if(pVM->passive_set_time >= TSu  &&
     pVM->state == UNDERLOAD_PASSIVE) {
    reset passive_set_time for Pvm
    pVM->state = AVAILABLE; // Make VM available for load balancing
  }
  else if(pVM->passive_set_time >= Tso &&
          pVM->state == OVERLOAD_PASSIVE) {
    reset passive_set_time for pVM
    pVM->state = AVAILABLE; // Make VM available for load balancing
    …
  }
}
void start() { //initiate load balancing environment
  …
  Initialize time intervals WTOver and WTUnder
  Initialize time intervals TSu and TSo
  Initialize various queues qOL_low, qOL_std and qOL_high
  Launch thread thread_PBOverLoadedVM (qOL<low,std,high>)
```

```
Initialize various queues qUL_low, qUL_std and qUL_high
    Launch thread thread_PBUnderLoadedVM(qUL<low,std,high>)
    Launch thread thread_PBLoadBalancer(qOL<low,std,high>)
    Launch thread thread_ManageState
    …
  }
};  //end of module PriorityBasedLoadBalancer
```

IV. IMPLEMENTATION AND RESULTS

The prototype testbed for the suggested mechanism was deployed. Cloud host setup was setup with the open source CentOS Linux operating system, on which an Infrastructure as a Service cloud environment was established with help of the open source cloud computing software OpenStack [10].

Many times, it is required to allocate and share cloud based resources according to the priority level of virtual machines. In absence of which, it may happen that the virtual machines which are at lesser priority level, may eventually get more and more cloud resources due to load balancing. The suggested algorithm has been implemented with help of separate priority queues for virtual machines having varying priority levels. Resulting work load observations and relevant waiting times are shown in the Table 1.

For obtaining the load balancing decision, the average waiting time turns around to be 9.62 ms for all virtual machines with varying priority levels. Moreover, the Table 1 clearly indicates reduction in the workload of after load balancing took place on the previously overloaded virtual machines.

The state manager thread `thread_ManageState` works in coordination with mainly three different threads: `thread_PBLoadBalancer` for priority based load balancing, `thread_OverLoadedVM (qOL<low,std,high>)` for determining overloaded VMs and `thread_UnderLoadedVM (qUL<low,std,high>)` for determining lightly loaded VMs. The thread `thread_PBLoadBalancer` ensures not to allot higher end virtual machine resources to lower end virtual machines unless there is no more present requirement for higher end VM resources.

V. RESULTS AND DISCUSSION

The testbed cloud host setup was setup with the open source CentOS Linux operating system, on which an Infrastructure as a Service cloud environment was established with help of the open source cloud computing software OpenStack [10].

Many times, it is required to allocate and share cloud based resources according to the priority level of virtual machines. In absence of which, it may happen that the virtual machines which are at lesser priority level, may eventually get more and more cloud resources due to load balancing. The suggested algorithm has been implemented with help of separate priority queues for virtual machines having varying priority levels. Resulting work load observations and relevant waiting times are shown in the Table 1.

For obtaining the load balancing decision, the average waiting time turns around to be 9.62 ms for all virtual machines with varying priority levels. Experimental results are shown in Table 1. Moreover, the Table 1 clearly indicates reduction in the workload of after load balancing took place on the previously overloaded virtual machines.

The state manager thread `thread_ManageState` works in coordination with mainly three different threads: `thread_PBLoadBalancer` for priority based load balancing, `thread_OverLoadedVM (qOL<low,std,high>)` for determining overloaded VMs and `thread_UnderLoadedVM (qUL<low,std,high>)` for determining lightly loaded VMs. The thread `thread_PBLoadBalancer` ensures not to allot higher-end virtual machine resources to lower-end virtual machines unless there is no more present requirement for higher-end VM resources.

Table 1: Waiting time and Workload on VMs before and after load balancing

Sr.	Workload before load balancing		Workload after load balancing		Waiting time (ms)
	Source VM: Workload	Destination VM: WorkLoad	Source VM: WorkLoad	Destination VM: WorkLoad	
1.	192.168.10.2: 84%	192.168.10.9: 36%	192.168.10.2: 66%	192.168.10.9: 58%	8.9
2.	192.168.10.3: 92%	192.168.10.1: 39%	192.168.10.3: 65%	192.168.10.1: 60%	10.2
3.	192.168.10.4: 85%	192.168.10.8: 31%	192.168.10.4: 60%	192.168.10.8: 47%	10.4
4.	192.168.10.5: 81%	192.168.10.6: 33%	192.168.10.5: 63%	192.168.10.6: 49%	9.4
5.	192.168.10.7: 93%	192.168.10.11: 19%	192.168.10.7: 69%	192.168.10.11: 42%	9.2

VI. CONCLUSION

A novel load balancing approach in cloud computing environment has been presented here. The technique suggested here is based on priority levels of virtual machines. The suggested algorithm and its implementation indicates significantly optimized waiting time for load balancing mechanism. In future, the technique here may be further evaluated for solving load balancing issues in hybrid cloud computing environment.

REFERENCES

1. A. Gupta and R. K. Jha, A Survey of 5G network: Architecture and Emerging Technologies, IEEE Access, Vol. 3, 2015.
2. H. Hsieh and M. Ching, The Incremental Load Balancer Cloud Algorithm by Using Dynamic Data Deployment, Journal of Grid Computing, 17 (3), 2019
3. M. Kumar and Suman, Priority Based Virtual Machine Selection Algorithm in Cloud Computing, International Journal of Recent Trends and Engineering, 8 (3), 2019
4. N. Joshi, Performance-Centric Cloud-Based e-Learning, The IUP Journal of Information Technology, 10(2), 2014
5. N. A. Joshi, Efficient Load Balancing in Cloud Computing, Research Review International Journal of Multidisciplinary, 4(6), 2019
6. R. Haidri, C. Padmanabh Katti and P. Saxena, Capacity Based Deadline Aware Dynamic Load Balancing Model in Cloud Computing Environment, International Journal of Computers and Applications, 2019
7. S. Dubey, M. Dahiya and S. Jain, Implementation of Load Balancing Algorithm with Cloud Computing for Logistics, Journal of Engineering and Applied Sciences, 14(2), 2019
8. S. Ghosh and C. Banerjee, Dynamic Time Quantum Priority Based Round Robin for Load Balancing in Cloud Environment, International Conference on Research in Computational Intelligence and Communication Networks, 2019
9. S. Manasser, M. Alzghoul and M. Mohmad, An Advanced Algorithm for Load Balancing in Cloud Computing using MEMA Technique, International Journal of Innovative Technology and Exploring Engineering, 8 (3), 2019
10. The OpenStack Cloud Computing Software, https://openstack.org

AUTHOR'S PROFILE

Dr Narayan A Joshi is working as Professor & Head at Dharmsinh Desai University, Nadiad. He was felicitated with prestigious 'Drona Award' by IBM – India.