Journal of Computer Science

Original Research Paper

Application of Viral System Algorithm in Load Balancing of Cloud Environment

[1]Damodar Tiwari, [2]Shailendra Singh and [3]Sanjeev Sharma

[1]Department of Computer Science and Engineering,
Bansal Institute of Science and Technology, Bhopal, India
[2]Department of Computer Engineering and Applications,
National Institute of Technical Teachers Training and Research, Bhopal, India
[3]Department of School of Information Technology,
Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India

Article history
Received: 22-05-2018
Revised: 27-06-2018
Accepted: 11-07-2018

Corresponding Author:
Damodar Tiwari
Department of Computer Science and Engineering,
Bansal Institute of Science and Technology, Bhopal, India
Email: damodarptiwari21@gmail.com

Abstract: As the cloud computing technology is gaining popularity with time, more and more users and applications are shifting towards it. This is why clouds are experiencing high load, which demands for load balancing of user tasks submitted to cloud for execution. This makes load balancing of non-preemptive tasks a key issue in cloud computing. Superior task scheduling leads to balanced loads among cloud nodes, which results in faster execution of tasks. Task scheduling in cloud environment is an instance of NP-hard optimization problem. When few nodes in a cloud are overloaded whereas other nodes are under loaded then in such situation the performance of overloaded VMs is diminished. It demands a task scheduling so that the incoming tasks can be distributed uniformly across virtual machines (VMs) for proper utilization of available resources. In this study, we propose a novel load balancing algorithm named Viral System Based Load Balancing (VSB-LB) algorithm, which is based on bio-inspired viral system algorithm that distributes the tasks uniformly among VMs. The proposed algorithm is compared with basic load balancing algorithms such as First Come First Serve (FCFS), Weighted Round Robin (WRR) as well as newer bio-inspired Load balance Aware Genetic Algorithm (LAGA) to show its effectiveness. Simulation results proved that VSBLB outperforms FCFS and WRR and LAGA for performing load balancing.

Keywords: Viral System Algorithm, Cloud Computing, Load Balancing, Task Scheduling

Introduction

Now-a-days one of the fastest emerging fields in information technology is cloud computing, also referred to as simply "the cloud," which delivers on-demand computing resources including everything from platform and applications, to data centers over internet on a pay-for-use basis. It is entirely an internet-based approach where all the resources are placed on a cloud consisting number of high speed interlinked computers for serving the incoming requests from connected clients. Under this technology, clients can use computational power, software services and platform offered by cloud service providers while paying only for duration those resources have been accessed. This forces the conventional software licensing policies to change and avoids spending of money for the facilities the client does not use in a software package (Dhinesh Babu and Venkata Krishna, 2013).

On the basis of architecture, cloud computing can be divided into three layers: Application layer, platform layer and infrastructure layer. NIST has defined three service models that a cloud service provider can provide to consumers which are *Software as a Service* (SaaS), *Platform as a Service* (PaaS) and *Infrastructure as a Service* (IaaS). SaaS makes applications running on a cloud infrastructure accessible to consumers. PaaS makes consumer created or acquired applications deployable onto cloud infrastructure. IaaS provides provision for processing, storage, networks and other fundamental computing resources where the consumer is

able to deploy and run software, which can include operating systems and other applications. On the other hand, there are three commonly-used cloud deployment models: *private*, *public* and *hybrid*. An additional, less-commonly used model is *community cloud*. A *private cloud* is meant for a single organization where software such as *VMWare*, *vCloud Director*, or *OpenStack* can be used. A *public cloud* is a set of computing resources provided by third-party organizations like *Amazon Web Services*, *Google AppEngine* and *Microsoft Azure*. A *hybrid cloud* is a mixture of computing resources provided by both *private* and *public clouds*. A *community cloud* shares computing resources across several organizations.

These days, most of us are using cloud services directly or indirectly. From email systems, social networking websites to mobile chatting apps for connecting people to each other are running on cloud. According to research and advisory consultancies including *International Data Corporation* (IDC), global SaaS market is projected to grow from $49B in 2015 to $67B in 2018, attaining a CAGR of 8.14%. It also states that global spending on IaaS is expected to reach $16.5B this year, an increase of 32.8% from 2014 and cloud applications will account for 90% of worldwide mobile data traffic by 2019, which was 81% in the end of 2014.

A client expects a high quality of service and therefore strives to find a reliable and fast cloud service that falls under the budget. In order to meet these expectations, it is essential for cloud service provider to utilize available resources optimally. For increasing efficiency and capabilities of cloud, virtualization is performed which refers to creating multiple virtual versions of resources, known as Virtual Machines (VMs) within a host. It enables same set of resources available in one or more execution environments. VMs should complete the execution of user submitted tasks as fast as possible. At the same time, one VM may experience overload whereas other may see under loaded condition. Such improper utilization of resources results into longer execution time and waiting time leaving the clients disappointed.

To avoid above situation, a scheduler (also known as load balancer) is used, which receives all the tasks incoming from clients for execution, keeps them in a queue, applies a task scheduling algorithm to determine best possible set of VMs for executing those tasks and finally assigns the tasks across VMs for execution. A scheduler works as good as its scheduling algorithm is and it can significantly improve execution time by optimizing the utilization of cloud resources. Since billions of users can be accessing a cloud at a time therefore it requires a large scale task scheduling algorithm. Since performance of a cloud depends upon how scheduling of tasks is performed therefore it makes task scheduling one of the major concern that needs to be addressed in this area.

In this study, we propose a viral system based load balancing (VSB-LB) algorithm for scheduling independent heterogeneous tasks in cloud environment and reducing the execution time of tasks. Rest of the paper is organized as follow: Section-2 presents related works, section-3 discusses basics concepts of viral system algorithm, section-4 explains the proposed load balancing model based on viral system behavior, section-5 discusses about the behavior and performance of proposed model, section-6 finally concludes the paper.

Related Work

During the last decade, rapid increase in number of cloud users has caught the attention of researchers from all round the globe. Recently a large variety of task scheduling and load balancing algorithms have been introduced, which are briefly discussed in this section.

In the past few years, researchers around the globe have proposed a variety of solutions to perform task scheduling and load balancing in cloud. Subramanian *et al*. (2012) used dynamic priority for scheduling virtual machines. Paul and Sanyal, (2011) used credit based scheduling decision for evaluating group of task in the task queue and find the minimal completion time of all task. Zhao and Huang (2009) reduced the migration time of virtual machines through shared storage and fulfilling the zero-downtime relocation of virtual machines by transforming them as Red Hat cluster services. Li *et al*. (2011) proposed a hybrid energy-efficient scheduling algorithm using dynamic migration that not only reduces response time but also conserves energy besides achieving load balancing. Mondal *et al*. (2012) used a local optimization stochastic hill climbing approach for allocating incoming jobs to virtual machines whereas Wadhwa *et al*. (2015) proposed a scheduling that aims to improve QoS by minimizing the waiting time.

Fang *et al*. (2010) proposed a two levels task scheduling mechanism for load balancing in cloud computing. It not only meets user's requirements, but also leads to high resource utilization. The first level scheduling is performed at application layer to the virtual machine and second is from virtual machine to host resources. The performance of this technique can be improved by taking more parameters into account such as bandwidth, cost etc.

Bitam *et al*. (2012) applied population based meta heuristic Bees Life Algorithm (BLA) to solve the job scheduling problem in cloud. It improves the efficiency and the performance in terms of execution time.

Xu *et al*. (2013) proposed a cloud partitioning based load balancing conceptual framework for a large public cloud. It creates separate load balancer for each partition, all of which are controlled by a main controller. At first, the controller chooses right partition according to partition status that can be either idle, normal or overload. Once the partition is chosen, the load balancer of that partition

applies an appropriate load balancing strategy to choose best suitable node within the partition to execute the task. This framework does not consider many practical aspects and its feasibility is not yet assessed.

Dhinesh Babu and Venkata Krishna (2013) proposed an algorithm for load balancing of tasks, which is completely inspired by natural foraging behavior of honey bees, which adopt to find and reap food. In bee hives, scout bees forage for food sources and upon finding one, they come back to the beehive and advertise it by a waggle/tremble/vibration dance that gives the idea about the quality and/or quantity of food and its distance from the beehive. Forager bees then follow scout bees to that location and begin to reap it. They then return to beehive and do same before other bees in the hive giving an idea of how much food is left and hence resulting in either more exploitation or abandonment of the food source. In the same manner, removed tasks from over loaded VMs are considered analogous to honey bees. Upon submission to the under loaded VM, the task updates the number of various priority tasks and load of that particular VM to all other waiting tasks. It helps other tasks in choosing their virtual machine based on load and priorities.

Dasgupta *et al*. (2013) proposed a genetic algorithm based load balancing technique for improving the response time. The three major operations involved are selection, genetic operation and replacement. Authors claim that it can handle a vast search space, applicable to complex objective function and can avoid being trapped in local optimal solution. However the cost function includes only two parameters i.e., number of instructions in task and MIPS of VM under consideration, which can be improved by considering other valuable parameters such as load difference among VMs etc.

Zhan *et al*. (2014) tried to solve the task scheduling problem in cloud computing by using a Load balance Aware Genetic Algorithm (LAGA) with Min-min and Max-min methods. It introduced the Time Load Balance (TLB) model and provided interaction between makespan and TLB that helps the algorithm to minimize the makespan. The Min-min and Max-min methods were used to find promising individuals at the beginning of evolution leading to a noticeable improvement of evolution efficiency. The $n \times m$ task scheduling problem was represented by corresponding Resource-Task Model and the characteristics of the problem were described using matrix called Expected Time of Completion containing the completion time of each task with each resource. Another matrix Expected Scheduling to Compute (ESC) describes a solution to task scheduling problem by recording the matching of tasks and resources.

A Hybrid Artificial Bee and Ant Colony optimization (H_BAC) load balancing algorithm is proposed in (Gamal *et al*., 2017). It inherits the main behaviors of both ACO and ABC algorithms and takes into consideration monitoring the load of Virtual machines (VMs) and the decision of load balancing before scheduling tasks in VMs. The authors claim that it uses two constraints in order to select the most suitable VM and guarantee the load balancing of the system.

An evolutionary algorithm for scheduling tasks in Cloud computing is proposed in (Navimipour and Milani, 2015). It is based on the obligate brood parasitic behavior of some cuckoo species in combination with the Lévy flight behavior of some birds and fruit flies and focuses on minimizing the total waiting time of tasks. The downside of their work is that it is applicable only for homogeneous cloud infrastructure.

Domanal *et al*. (2017), authors have proposed three different Bio-Inspired algorithms for efficient scheduling and resource management in a cloud environment. The MPSO algorithm was found more efficient in scheduling the tasks as compared to other algorithms. On the other hand, the proposed HYBRID (MPSO + MCSO) approach was more effective in allocating the resources to VMs when compared to other algorithms. The proposed HYBRID algorithm not only reduced the average response time but also increased resource utilization by 12% when compared to other state-of-the-art benchmark algorithms.

Most existing systems consider only two resources i.e., CPU and memory, while evaluating their performance. In (Gawali and Shinde, 2018), authors proposed a heuristic algorithm that performs task scheduling and allocates resources efficiently in cloud computing environments. They used real *Cybershake* and *Epigenomics* scientific workflows as input tasks for the system and have also considered the bandwidth as a resource. Their heuristic approach gives improved results as compared to existing BATS and IDEA frameworks with respect to turnaround time and response time. On the other hand, proposed heuristic approach efficiently allocates resources with high utility. The results have shown that it achieves maximum utilization result for computing resources such as CPU, memory and bandwidth.

Viral System Behavior

Viral System Algorithm (VSA) is a relatively new bio inspired algorithm based on viral infection process. It was originally proposed in (Cortés *et al*., 2012). VSA consists of two basic operations namely *replication* and *infection*. A Viral System (VS) consists of following three components:

a) Set of viruses (V), where each virus consists of its own state (s), input (i), output (o) and a process (p):

$$V = \{Virus_1, Virus_2, Virus_3, \ldots, Virus_n\}$$

where, $Virusi = \{s_i, i_i, o_i, p_i\}$

b) An organism to be infected (O), which includes its state (S) and process (P):

$$O = \{S, P\}$$

c) Interaction (I) between the above two components

Thus a viral system can be represented using three tuples as $VS = \{V,O,I\}$. A clinical picture represents organism's health, which contains a set of infected cells. The organism may generate antibodies during infection, which resists the spreading of infection in organism. Once the viral system has been created, it works under following steps:

i. **Set coding and measuring criteria**

- Create coding representation for possible solutions
- Set criteria for measuring quality of solutions
- Set criteria about how antibodies are generated

ii. **Initialize system and let the infection spread**

- Create and initialize clinical picture
- Determine the type of infection to be applied
- Run through iterations and let the viruses interact with organism's cells. The cells get gradually infected by virus and during this process, cell may produce antibody as a result of which, they are excluded from clinical picture

iii. **Termination**

- Either the organism will die which refers to finding a good solution or
- The viruses get isolated from organism

The life cycle of a virus within organism's body can be of two types namely lytic replication and lysogenic replication. Lytic replication results in generating number of new viruses, which in turn infect other cells selectively or massively. Lysogenic replication is carried out by performing mutation in organism cells present in clinical picture. Let Y be the binomial random variable representing cells infected by the virus in the neighborhood. In selective infection, single cell from neighborhood is selected and its antigenic response is evaluated as a Bernoulli process (A). In massive infection, (Y-A) neighborhood cells are selected, infected and included into the clinical picture. If there is lack of space in clinical picture then it erases the cells randomly from the (Y-A) selected cells. In case of antigenic response, a lysogenic replication is performed.

Cortés *et al*. (2010) have applied VSA to solve the Steiner problem, which falls under NP-Complete problems. It was found that VSA was able to produce even better results as compared to GA. This motivated us to apply VSA for performing tasks scheduling in cloud, which is an NP-hard problem.

Modeling of Viral System based Load Balancing

The clinical picture contains a number of cells where each cell or genome is represented using resource task model as described in (Zhan *et al*., 2014). According to this model, if there are n tasks to be assigned among m virtual machines then i^{th} cell (C_i) is represented as shown in Fig. 1.

Where, C_{ik} is an integer value representing the index number of virtual machine on which the k^{th} task in i^{th} chromosome is scheduled to execute. The length of a cell is equal to the number of available tasks and C_i represents a possible distribution of tasks across available VMs. This whole distribution is one possible solution from numerous solutions in solution space. The cells in clinical picture repeatedly undergo infections massively or selectively according to *Viral System* algorithm.

As Fig. 2 depicts, the clinical picture contains a fixed number of cells. The i^{th} cell is represented as C_i where $1 \le i \le Size$ of clinical picture. With each cell, expected time to complete (ETC) and load difference (LD) is associated. The term ETC_i and LD_i denotes the expected time duration to complete the execution of all tasks and the load difference among all the VMs respectively in i^{th} solution. Smaller values of ETC and LD are desired i.e. smaller the values of ETC_i and LD_i better is the solution represented by cell C_i.

C_i =	C_{i1}	C_{i2}	C_{i3}	C_{i4}	C_{i5}	C_{i6}	C_{i7}	…	C_{in}

Fig. 1: Cell representation in resource task model

Clinical Picture			
S.No.	**C**	**ETC**	**LD**
1.	C_1	ETC_1	LD_1
2.	C_2	ETC_2	LD_2
.	.	.	.
.	.	.	.
Size.	C_{Size}	ETC_{Size}	LD_{Size}

Fig. 2: Clinical picture representation

The Ideal Average Load on a VM:

$$MIPS\_Share_j = \frac{\left(MIPS_{VM(j)} \times 100\right)}{\sum_{k=1}^{m} MIPS_{VM(k)}}$$

$$L_{j(ideal)} = \frac{\left(MIPS\_Share_j\right) \times \sum_{k=1}^{m} No\ of\ instructions\ in\ task\ T_i}{\sum_{k=1}^{m} MIPS_{VM(k)}}$$

where, $L_{j(ideal)}$ denotes the ideal average load on j^{th} VM (VM_j), T_i denotes the i^{th} task, n is the total number of tasks to be assigned across available m number of VMs. The value of i varies from 1 through n whereas k varies from 1 through m. $MIPS\_Share_j$ is the share of VM_j in total MIPS available through all k VMs.

The Actual Load on a VM:

$$L_{j(ideal)} = \sum Number\ of\ instructions\ in\ task\ T_i$$

where, $L_{j(actual)}$ denotes the actual load on VM_j and T_i denotes a task scheduled to run on VM_j.

The Load Difference (LD)

$$LD_i = \sum_{i=1}^{Size} \left| L_{j(actual)} - L_{j(ideal)} \right|$$

The load difference parameter LD_i is the summation of differences between actual load and ideal load of all the VMs when tasks are assigned as guided by solution C_i in clinical picture, a good load balancing algorithm should try to minimize this value.

The Expected Time to Complete (ETC) all the Tasks

$$ETC_i = \underset{1 \le j \le m}{Max} \left\{ \frac{Number\ of\ instructions\ assigned\ to\ VM_j}{MIPS_j} \right\}$$

The term ETC_i denotes the expected time to finish execution of all tasks when tasks are distributed according to solution C_i in clinical picture. The term m denotes total number of available VMs and $MIPS_j$ denotes the million instructions that can be executed per second by VM_j. The ETC_i is maximum time duration required among all the *VMs* to finish all the tasks assigned to it according to solution C_i. It is the most significant parameter in load balancing and a smaller value of *ETC* represents a better solution.

The Objective Function (OF)

$$OF_i = \frac{w_1}{ETC_i} + \frac{w_2}{LD_i}$$

The objective function for solution C_i is denoted by OF_i which is the weighted sum of *ETC* and *LD*. The terms w_1 and w_2 denote the weights associated with ETC_i and LD_i respectively.

Algorithm: ViralLoadBalancing

Let the array *ClinicalPicture* denotes clinical picture and the variable *Size* denotes the size of clinical picture.

Initialize the variables *Size,* Max_{Lytic}, $Max_{Lysogenic}$, $Range_{Lytic}$ *and* $Range_{Lysogenic}$
Set *LysogenicCount[i] = LyticCount[i] = 0* for all *i* where $1 < i < Size$
Create *ClinicalPicture[Size]*
  Do until all cells develop antibody or all cells are collapsed i.e., death of organism
    For each cell (C_i) *in ClinicalPicture do*
      Generate random number R between 0 and 1 and find $Antibody_{Ci} = Bernouli(R)$
      If $Antibody_{Ci} > Threshold_{Antibody}$ (which means this cell has produced antibody)
        Remove cell C_i from *ClinicalPicture* leaving a vacant space
      Else (which means the cell failed to produce antibody and therefore will get infected)
      Generate a random number *ReplicationType* between 0 and 1
      If value of *ReplicationType* is within $Range_{Lytic}$ (which results in lytic replication)
        ++LyticCount[i]

$Max_{Lytic(Ci)}$ = Max_{Lytic} x [(OF(Ci)-$OF(C_{Best})$/$OF(C_{Best})$)]
If(*LyticCount[i]* > $Max_{Lytic(Ci)}$)
Remove cell C_i from *ClinicalPicture*
Else
Generate a boolean value *InfectionType* that can contain either 0 or 1
If(*InfectionType==0)* (which means a massive infection)
Remove C_i from *ClinicalPicture (The cell has collapsed)*
z = BinRndmNmbr($Max_{Lytic(Ci)}$, *p)*
Create a set S containing z new viruses created by mutating z neighbors of C_i such that any cell in S is not already present in *ClinicalPicture*
If (*NumberOfVacantSpace(ClinicalPicture)* < *z)* then
Remove such z cells from *ClinicalPicture* that have lowest value of OF
Add all the viruses from S to ClinicalPicture
Else (which means a selective infection)
Infect one neighbor cell of C_i in *ClinicalPicture* by performing mutation operation on it such that newly generated cell is not present in *ClinicalPicture*
Else If value of *ReplicationType* is within $Range_{Lysogenic}$ (which results in lysogenic replication)
++ *LysogenicCount[i]*
$Max_{Lysogenic(Ci)}$ = $Max_{Lysogenic}$ x [(OF(C_i)-$OF(C_{Best})$/$OF(C_{Best})$)]
If(*LysogenicCount[i]* > *MaxLysogenic(Ci))*
C_i = Perform mutation operation on C_i such that mutated C_i is not present in *ClinicalPicture*

The algorithm continues to execute until either all the cells in clinical picture develop antibody or all the cells are collapsed resulting into death of organism. In clinical picture, the i^{th} cell is represented as C_i which represents the i^{th} possible solution. The function *Bernouli(R)* returns a bernouli random number. If the value returned by this function is greater than $Threshold_{Antibody}$ than it is assumed that the cell C_i has generated antibody against viruses. The $Range_{Lytic}$ and $Range_{Lysogenic}$ represent the range of values to determine the replication type of viruses in current cell. If the value of random variable $Replication_{Type}$ comes out between the range set by $Range_{Lytic}$ then it results into a lytic replication otherwise a lysogenic replication is initiated in C_i.

The arrays *LyticCount* and *LysogenicCount* keep track about the number of lytic and lysogenic infections respectively that have occurred in each cell. The $Max_{Lytic(Ci)}$ is updated based upon the value of objective function *OF*. The $OF(C_i)$ denotes value of *OF* for cell C_i and $OF(C_{Best})$ denotes the best (minimal) value of *OF* discovered so far. If *LyticCount* for a cell exceeds $Max_{Lytic(Ci)}$ then that cell is removed from ClinicalPicture. Otherwise selective or massive infection takes place depending upon the value of random boolean variable *InfectionType*. A *0* value of *InfectionType* means massive infection whereas *1* means selective infection.

For massive infection, the value of *z* is calculated through function *BinRndmNmbr*($Max_{Lytic(Ci)}$, *p)* which return the random variable according to binomial distribution. This function takes two arguments $Max_{Lytic(Ci)}$ and *p* which is the single probability of one replication. In case of selective infection, only one neighboring cell of C_i is mutated. The array *LysogenicCount* maintains the number of lysogenic infections occurred in each cell in clinical picture. In case of lysogenic replication, the value of *LysogenicCount[i]* is incremented by one and $Max_{Lysogenic(Ci)}$ is updated according to value of *OF*. If *LysogenicCount[i]* exceeds $Max_{Lysogenic(Ci)}$ then mutation takes place in C_i.

A variety of mutation operators such as twors, central inverse, reverse sequence, throas, thrors, partial transfer shuffle etc. are available for performing mutation in a cell. Since the partial transfer shuffle mutation *(PSM)* is known to effective in travelling salesman problem therefore we have used *PSM* in which, a part of the order of genes gets changed. Next section presents the details of simulation environment and performance evaluation of viral system based load balancing algorithm.

Experimental Results

For evaluating the performance of VSBLB it is implemented using *CloudSim-3.0.3* proposed in (Calheiros *et al*., 2011), which is considered as a standard test bench for simulating cloud environments. Since effective load balancing can reduce the makespan considerably therefore in this section, performance of *VSBLB* is compared with *FCFS*, *WRR* and *LAGA* in terms of makespan i.e., the overall task completion time (Dhinesh Babu and Venkata Krishna, 2013). The initial values of various parameters of algorithm used in VSBLB during simulation are shown in Table 1.

Moreover, we have created a heterogeneous simulation environment where hardware configuration of each VM and length of each task is different than the other. Table 2 illustrates the makespan before and after applying load balancing using VSBLB.

Figure 3 illustrates the graphical representation of makespan before and after applying load balancing using VSBLB. The X-axis represents number of tasks and Y-axis represents the execution time. Figure 4 compares the load difference before and after applying load balancing using VSBLB.

Table 3 illustrates the load difference before and after applying load balancing using VSBLB.

The makespan and load difference were calculated when number of tasks were *100*, *200, 300* and *400* with *10* VMs. It is apparent that VSBLB improves both makespan and the load difference considerably. As the number of tasks increases, the difference in makespan becomes higher. The reduction in makespan was found to be *22%*, *37%*, *56% and 67%* when number of tasks is *100, 200, 300* and *400* respectively.

Table 1: Parameter values

Parameter	Value
Size	25
MaxLytic	5
RangeLytic	0.1-0.5
RangeLysogenic	0.6-1
ThresholdAntibody	0.5

Table 2: Makespan before and after applying VSBLB

Number of tasks	Execution time	
	Before load balancing	After load balancing
100	377	293
200	1045	650
300	2004	882
400	3150	1050

Table 3: Load difference before and after applying VSBLB

Number of tasks	Load difference	
	Before load balancing	After load balancing
100	454634	347339
200	1363635	662725
300	2727281	1692013
400	5454639	3384109

In Fig. 4, X-axis represents load difference value and Y-axis represents the number of tasks. After applying VSBLB, the load difference is reduced considerably, which reveals better load distribution among VMs.

Figure 5 shows comparison of makespan between VSBLB, first come first serve and weighted round robin algorithms. The X-axis represents the time taken for finishing the execution and Y-axis represents the number of tasks.

As compared to WRR, VSBLB improves the execution times by 13%, 51%, 34%, 48% and 46% when number of tasks is set to *100*, *200*, *300*, *400* and *500* respectively. Similarly, in comparison to FCFS, an improvement of 22%, 67%, 38%, 60% and 56% in execution time were seen for 100, 200, 300, 400 and 500 tasks respectively. It reveals that VSBLB shows significant improvement in execution time as compared to WRR and FCFS.

Figure 6 compares the makespan between VSBLB and LAGA algorithm which is another bio inspired, genetic algorithm based load balancing technique. As compared to LAGA, the VSBLB improves execution time by 8%, 38%, 9%, 13% and 14% when number of tasks is set to 100, 200, 300, 400 and 500 respectively. The VSBLB performs slightly better than LAGA and it does reveal that VSBLB seems at least equally promising as genetic algorithm, which opens a new direction for research in this field.

Figure 7 to 10 compares the execution time between VSBLB, FCFS and WRR when number of VMs is set to 12, 15, 18 and 20 respectively. It is apparent that the time taken by VMs for executing the tasks with VSBLB is always smaller as compared to FCFS and WRR irrespective of number of VMs employed.

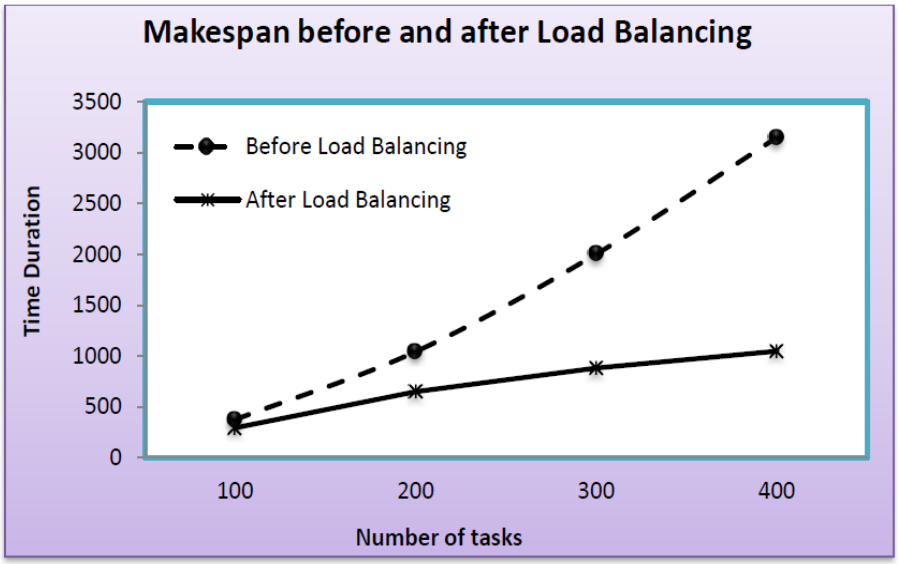

Fig. 3: Comparison of makespan before and after load balancing using VSBLB

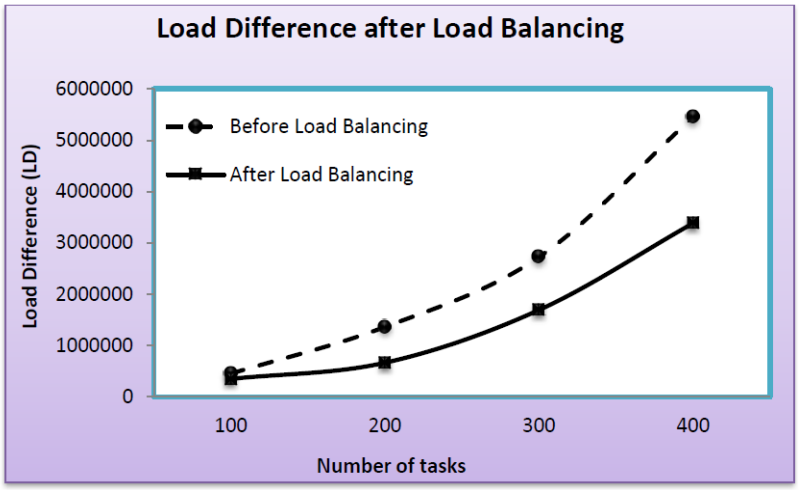

Fig. 4: Comparison of load difference before and after load balancing using VSBLB

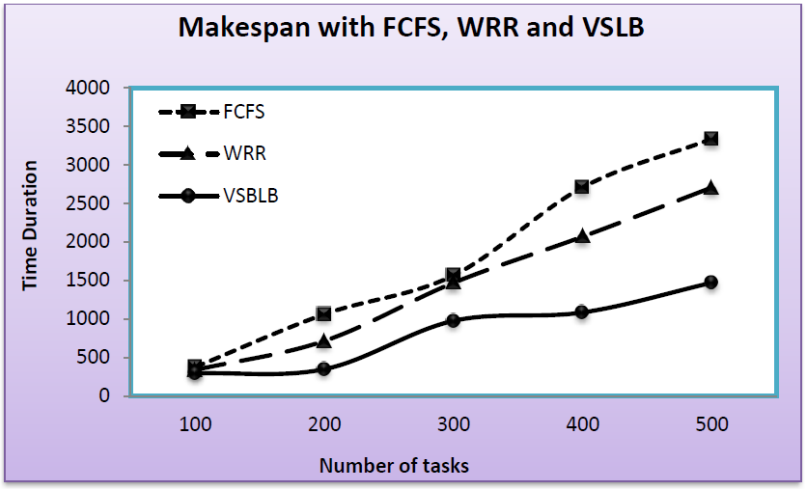

Fig. 5: Comparison of makespan between FCFS, WRR and VSBLB

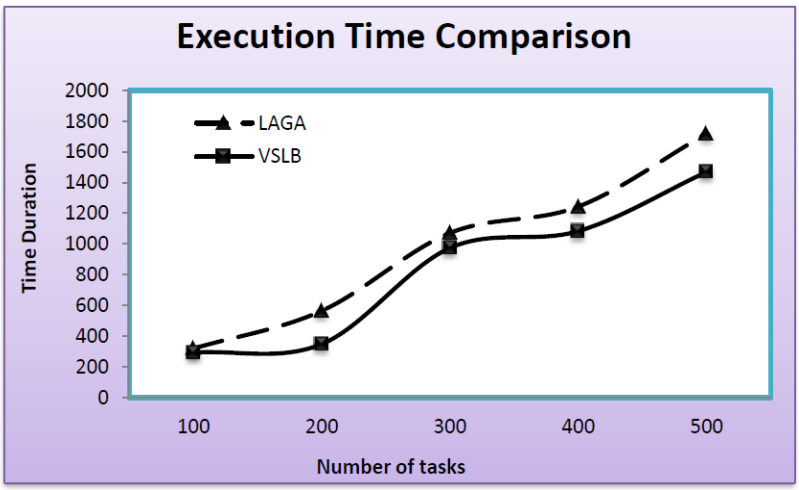

Fig. 6: Comparison of makespan between LAGA and VSBLB

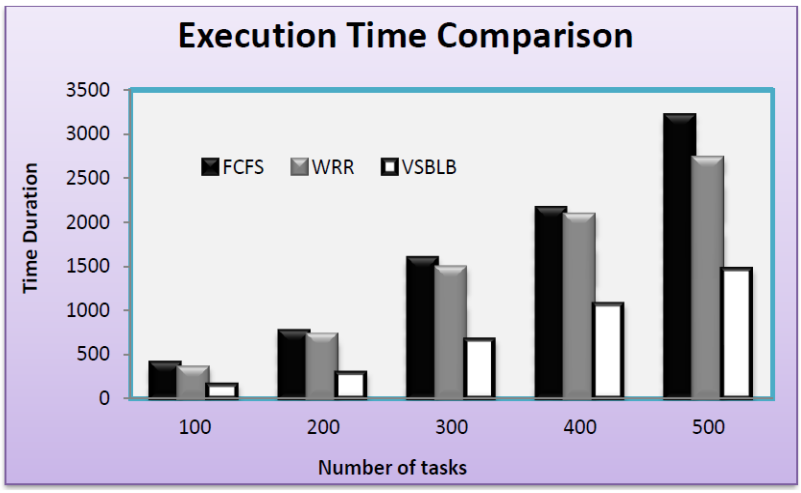

Fig. 7: Comparison of execution time between FCFS, WRR and VSBLB with 12 VMs

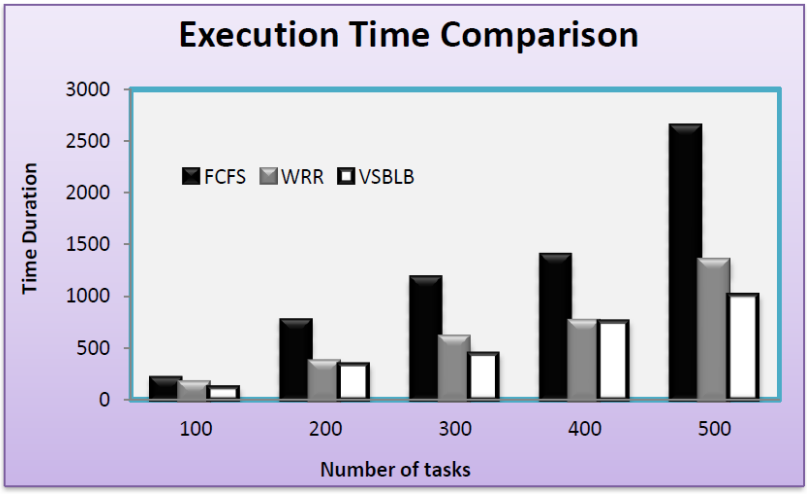

Fig. 8: Comparison of execution time between FCFS, WRR and VSBLB with 15 VMs

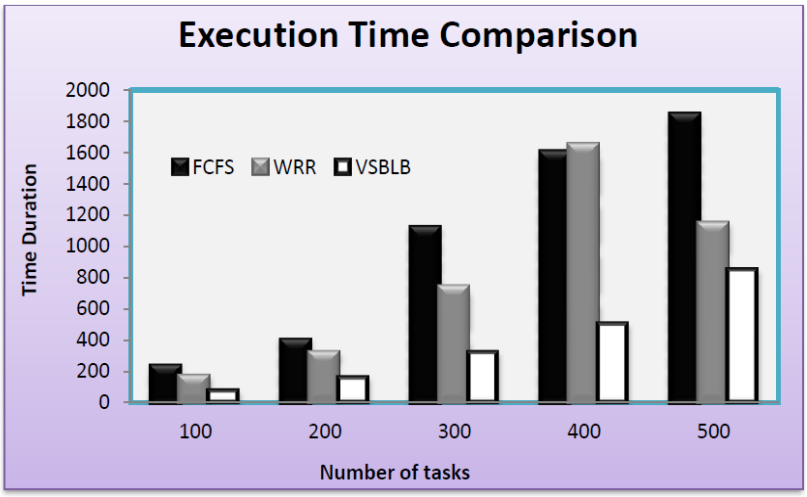

Fig. 9: Comparison of execution time between FCFS, WRR and VSBLB with 18 VMs

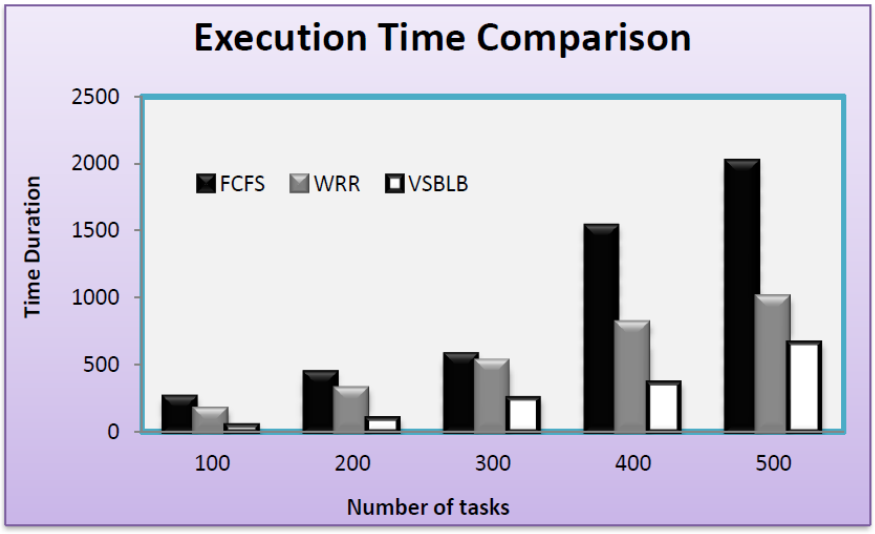

Fig. 10: Comparison of execution time between FCFS, WRR and VSBLB with 20 VMs

Conclusion

In this study a bio inspired viral system based load balancing technique for cloud computing environment is proposed. The goal of this paper is to enable the reader how viral system can be applied for performing load balancing in cloud. The algorithm distributes balanced load among the nodes and reduces execution time of user submitted tasks. Simulation results have revealed a significant improvement in distributed load and total execution time of tasks as compared to FCFS and WRR. It is also seen that viral system based load balancing,j is slightly better than genetic algorithm when applied in cloud computing environment, which may be a motivating fact for further research in this field.

Author's Contributions

Damodar Tiwari: Contributed in literature survey, algorithm preparation, writing manuscript.

Shailendra Singh: Contributed in writing algorithm.

Sanjeev Sharma: Contributed in result analysis.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and there are no ethical issues involved.

References

Bitam, S., 2012. Bees life algorithm for job scheduling in cloud computing. Proceedings of the ICCIT, (CIT' 12), pp: 186-191.

Calheiros, R.N., R. Ranjan, A. Beloglazov, C.A.F. De Rose and R. Buyya, 2011. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software Pract. Exp., 41: 23-50. DOI: 10.1002/spe.995

Cortés, P., J.M. García, J. Muñuzuri and J. Guadix, 2010. A viral system massive infection algorithm to solve the Steiner tree problem in graphs with medium terminal density. Int. J. Bio-Inspired Comput., 2: 71-77. DOI: 10.1504/IJBIC.2010.032123

Cortés, P., J.M. García, J. Muñuzuri and J. Guadix, 2012. Viral system algorithm: Foundations and comparison between selective and massive infections. Trans. Inst. Measurement Control, 34: 677-690. DOI: 10.1177/0142331211402897

Dasgupta, K., B. Mandal, P. Dutta, J.K. Mondal and S. Dam, 2013. A Genetic Algorithm (GA) based Load balancing strategy for cloud computing. Proc. Technol., 10: 340-347. DOI: 10.1016/j.protcy.2013.12.369

Dhinesh Babu, L.D. and P. Venkata Krishna, 2013. Honey bee behavior inspired load balancing of tasks in cloud computing environments. Applied Soft Comput., 13: 2292-2303. DOI: 10.1016/j.asoc.2013.01.025

Domanal, S., R.M.R. Guddeti and R. Buyya, 2017. A hybrid bio-inspired algorithm for scheduling and resource management in cloud environment. IEEE Trans. Services Comput. DOI: 10.1109/TSC.2017.2679738

Fang, Y., F. Wang and J. Ge, 2010. A task scheduling algorithm based on load balancing in cloud computing. Proceedings of the International Conference on Web Information Systems and Mining, Oct. 23-24, Springer, Sanya, China, pp: 271-277. DOI: 10.1007/978-3-642-16515-3_34

Gamal, M., R. Rizk, H. Mahdi and B. Elhady, 2017. Bio-inspired load balancing algorithm in cloud computing. Proceedings of the International Conference on Advanced Intelligent Systems and Informatics, (ISI' 17), Springer, Cham, pp: 579-589. DOI: 10.1007/978-3-319-64861-3_54

Gawali, M.B. and S.K. Shinde, 2018. Task scheduling and resource allocation in cloud computing using a heuristic approach. J. Cloud Comput., 7: 4-4. DOI: 10.1186/s13677-018-0105-8

Li, J., J. Peng and W. Zhang, 2011. A scheduling algorithm for private clouds. J. Convergence Inform. Technol., 6: 1-9. DOI: 10.4156/jcit.vol6.issue7.1

Mondal, B., K. Dasgupta and P. Dutta, 2012. Load balancing in cloud computing using stochastic hill climbing-a soft computing approach. Proc. Technol., 4: 783-789. DOI: 10.1016/j.protcy.2012.05.128

Navimipour, N.J. and F.S. Milani, 2015. Task scheduling in the cloud computing based on the cuckoo search algorithm. Int. J. Model. Optimiz., 5: 44-47. DOI: 10.7763/IJMO.2015.V5.434

Paul, M. and G. Sanyal, 2011. Task-scheduling in cloud computing using credit based assignment problem. IJCSE, 3: 3426-3430.

Subramanian, S., G. Nitish Krishna, M. Kiran Kumar, P. Sreesh and G.R. Karpagam, 2012. An adaptive algorithm for dynamic priority based virtual machine scheduling in cloud. IJCSI, 9: 397-402.

Wadhwa, S., M. Jain and B. Pandey, 2015. Design and implementation of scheduling algorithm for high performance cloud computing. Int. J. Web Sci. Eng., 2: 15-20. DOI: 10.21742/ijwsesd.2015.2.1.02

Xu, G., J. Pang and X. Fu, 2013. A load balancing model based on cloud partitioning for the public cloud. Tsinghua Sci. Technol., 18: 34-39. DOI: 10.1109/TST.2013.6449405

Zhan, Z.H., G.Y. Zhang, Ying-Lin, Y.J. Gong and J. Zhang, 2014. Load balance aware genetic algorithm for task scheduling in cloud computing. Proceedings of the 10th International Conference on Simulated Evolution and Learning, Dec. 15-18, Springer, Dunedin, New Zealand, pp: 644-655. DOI: 10.1007/978-3-319-13563-2_54

Zhao, Y. and W. Huang, 2009. Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud. Proceedings of the 5th International Joint Conference on INC, IMS and IDC, Aug. 25-27, IEEE Xplore Press, Seoul, South Korea, pp: 170-176. DOI: 10.1109/NCM.2009.350