Research Article

Bhargavi K*, Sathish Babu B, and Jeremy Pitt

Performance Modeling of Load Balancing Techniques in Cloud: Some of the Recent Competitive Swarm Artificial Intelligence-based

https://doi.org/10.1515/jisys-2019-0084
Received Apr 01, 2019; accepted Nov 20, 2019

Abstract: Cloud computing deals with voluminous heterogeneous data, and there is a need to effectively distribute the load across clusters of nodes to achieve optimal performance in terms of resource usage, throughput, response time, reliability, fault tolerance, and so on. The swarm intelligence methodologies use artificial intelligence to solve computationally challenging problems like load balancing, scheduling, and resource allocation at finite time intervals. In literature, sufficient works are being carried out to address load balancing problem in the cloud using traditional swarm intelligence techniques like ant colony optimization, particle swarm optimization, cuckoo search, bat optimization, and so on. But the traditional swarm intelligence techniques have issues with respect to convergence rate, arriving at the global optimum solution, complexity in implementation and scalability, which limits the applicability of such techniques in cloud domain. In this paper, we look into performance modeling aspects of some of the recent competitive swarm artificial intelligence based techniques like the whale, spider, dragonfly, and raven which are used for load balancing in the cloud. The results and analysis are presented over performance metrics such as total execution time, response time, resource utilization rate, and throughput achieved, and it is found that the performance of the raven roosting algorithm is high compared to other techniques.

Keywords: Swarm artificial intelligence, Load balancing, Performance, Cloud computing, Efficiency

1 Introduction

Cloud computing is a distributed computing paradigm used to store huge amount of data to provide software, platform, or infrastructure services on demand to the users with features like high tolerance, scalability, availability, robustness and so on [1]. As it deals with huge amount of data there is a need to effectively distribute the load among the nodes in the cloud by making sure that situation of the node being overloaded or underloaded does not occur. Effective load balancing increases the overall performance of the cloud applications in terms of efficient resource utilization, lowered latency, bottleneck avoidance, and increased job completion rate. However, there are several issues related to load balancing in cloud, like varying Quality of Service (QoS) requirement of the users, sudden failure of the resources, traffic variation over different geographical areas, communication overhead, frequent migration of traffic, wastage of the resources, inconsistent service abstractions, and so on [2–6].

***Corresponding Author: Bhargavi K:** Department of Computer Science and Engineering, Siddaganga Institute of Technology, Tumakuru, India; Email: bhargavi.tumkur@gmail.com
Sathish Babu B: Department of Computer Science and Engineering, R. V. College of Engineering, Bangalore, India
Jeremy Pitt: Department of Electrical and Electronic Engineering, Imperial College, London

The swarm artificial intelligence based techniques use collective intelligence among many entities to solve the Nondeterministic Polynomial time (NP) problem by properly handling the imprecise, inefficient, and uncertain input data. There are few swarm intelligence survey papers available in literature which provides theoretical description followed by algorithms of swarm intelligence techniques like bat algorithm, ant colony optimization, particle swarm optimization, and so on. Most of these algorithms are old and they are discussed from biological optimization point of view, they lack application and performance point of view description [7–9]. Swarm artificial intelligence techniques which are developed recently includes ageist spider monkey, shark smell optimization, whale optimization, lion optimization, spider, antlion optimization, jellyfish food search, eagle search, elephant, raven roosting, dragonfly, crow search and others. All these techniques are not suitable for cloud domain, this paper focuses on performance modeling of some of the efficient techniques used for cloud load balancing which includes whale optimization, spider, dragonfly, and raven roosting [10–15]. The performance modeling results demonstrate that among the considered techniques the performance of the raven roosting is high compared to the other techniques.

The rest of the paper is structured as follows, Section 2 describes the system model, Section 3 gives mathematical definition of the performance metrics used to evaluate the swarm intelligence based load balancing techniques, Section 4 provides performance modeling of whale optimization load balancing technique, Section 5 provides performance modeling of spider load balancing technique, Section 6 provides performance modeling of dragonfly load balancing technique, Section 7 provides performance modeling of raven roosting load balancing technique, Section 8 deals with results and discussion, and finally, Section 9 draws the conclusion.

2 System Model

Consider a cloud computing environment CCE comprising of several partitions of data center $P'_K s$ monitored by a main controller MC, i.e., $CCE = \{MC, \{p_1, p_2, p_3 \ldots, p_k\} \in P\}$, $P /= \varnothing$ is a universal partition set comprising of several partitions. Every p_k is a collection of several virtual machines monitored by a partition controller PC, $p_k = \{PC, \{vm_1, vm_2, \ldots, vm_m\} \in VM\}$, where $VM /= \varnothing$ is a universal set comprising of m virtual machines. The possible state of p_k is described as a virtual machine vector vm^{pk}_{ks}, where vm^{pk}_{ks} denotes a virtual machine with k job set running s type of swarm intelligence based load balancing technique on p_k.

At time ΔT, the MC receives load status message $ls_i \in LS$ from PC of every p_k i.e., $p_k(LS, \Delta T) = \{PCid, \ LS = \sum ls_i\}$, where PC_{id} is the unique partition controller identifier, $ls_i = \{i, n, o\}$ is the status of load on vm_i in p_k i.e., idle, normal, or overloaded, and $LS /= \varnothing$ is a universal load status set of the partition. Let j_k be the incoming job requests set at MC, the j_k will be sent to normal or idle partitions, during which the PC runs Load Balancing Technique (LBT) with set of functional modules $fm'_i s$ to evenly distribute the load among the virtual machines in p_k. A high-level view of the system model considered for load balancing in a typical cloud environment is shown in Figure 1.

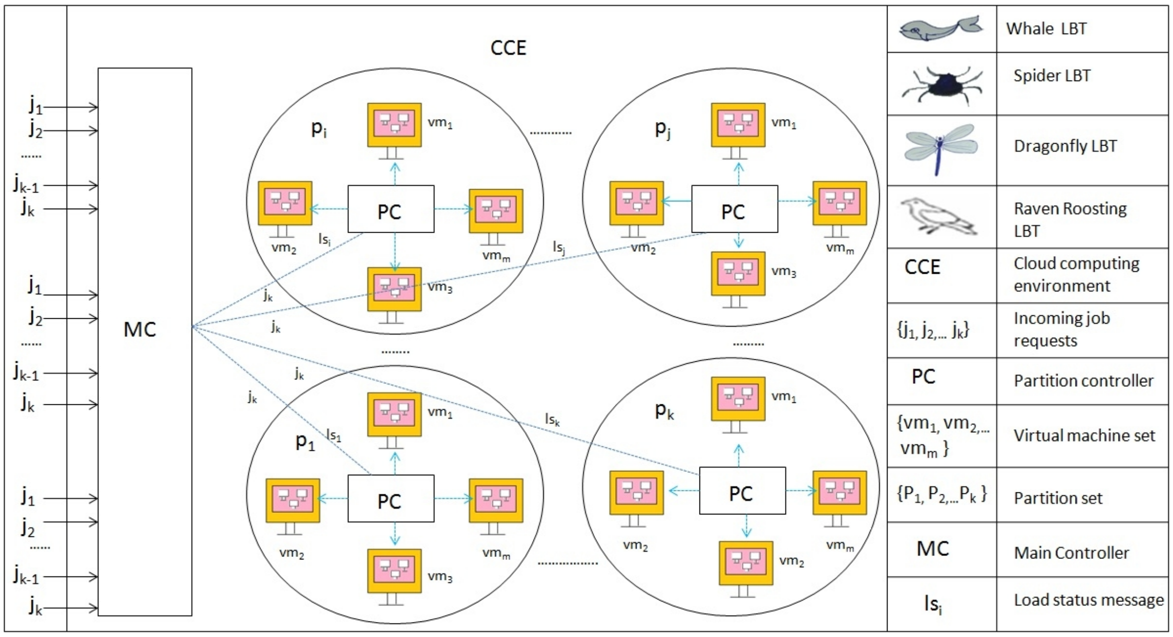

CCE
MC
PC
p_i
p_j
p_1
p_k
vm_1
vm_2
vm_3
vm_m
ls_i
ls_j
ls_1
ls_k
j_1
j_2
j_{k-1}
j_k
Whale LBT
Spider LBT
Dragonfly LBT
Raven Roosting LBT
CCE Cloud computing environment
$\{j_1, j_2, \ldots j_k\}$ Incoming job requests
PC Partition controller
$\{vm_1, vm_2, \ldots vm_m\}$ Virtual machine set
$\{P_1, P_2, \ldots P_k\}$ Partition set
MC Main Controller
ls_i Load status message

Figure 1: High level view of the system model

3 Definition

This section gives the mathematical definition of the performance metrics used in the paper to evaluate the efficiency of the selected load balancing techniques.

3.1 Total execution time

The total execution time TE of j_k on p_k with VM is defined as the total time taken to execute every functional module fm of the LBT.

$$TE(j_k, p_k, VM) = \sum_{i=1}^{i=m} \left[\sum_{j=1}^{j=J} \left[t_{fm^j} (j_k, p_k, vm_i) \right] \right] \tag{1}$$

3.2 Response time

The response time RT of j_k on p_k with VM is the total time elapsed between the execution of first functional module fm^f and last functional module fm^l of the LBT.

$$RT(j_k, p_k, VM) = \sum_{i=1}^{i=m} \left[t_{fm^f} (j_k, p_k, vm_i) - t_{fm^l} (j_k, p_k, vm_i) \right] \tag{2}$$

3.3 Resource utilization rate

The resource utilization rate RU of j_k on p_k with VM is the measure of number of virtual machine resources that are idle N_{ri}^{idle} and overutilized N_{ri}^{over} by the j_k with respect to total number of resources N_{ri} in p_k.

$$RU\,(j_k, p_k, VM) = \sum_{i=1}^{i=m} \left[\frac{N_{ri}^{idle}}{N_{ri}} \star \frac{N_{ri}^{over}}{N_{ri}} \right] \tag{3}$$

3.4 Throughput

The throughput TH of j_k on p_k with *VM* is the rate at which j_q among the allocated j_k have completed successfully by the *LBT*.

$$TH\,(j_k, p_k, VM) = \frac{j_q}{j_k} \tag{4}$$

4 Whale optimization LBT

We consider some of the methods dealing with load distribution among the nodes in the cloud using the foraging behavior of humpback whales as the reference for performance modeling. The methods considered are Whale optimization based Scheduler (W-Scheduler), Multi-objective Whale Optimization Algorithm (WOA) for task scheduling and load balancing, Levy flight trajectory-based Whale Optimization Algorithm (LWOA), and improved levy based whale optimization algorithm for virtual machine placement [16–21]. The technique uses bubble net hunting strategy of whales to distribute the load evenly among the virtual machines as shown in Figure 2. The spiral simulated hunting behavior of whales with the best search policy to chase the prey is used to select optimal virtual machines which are capable enough to execute the allocated jobs.

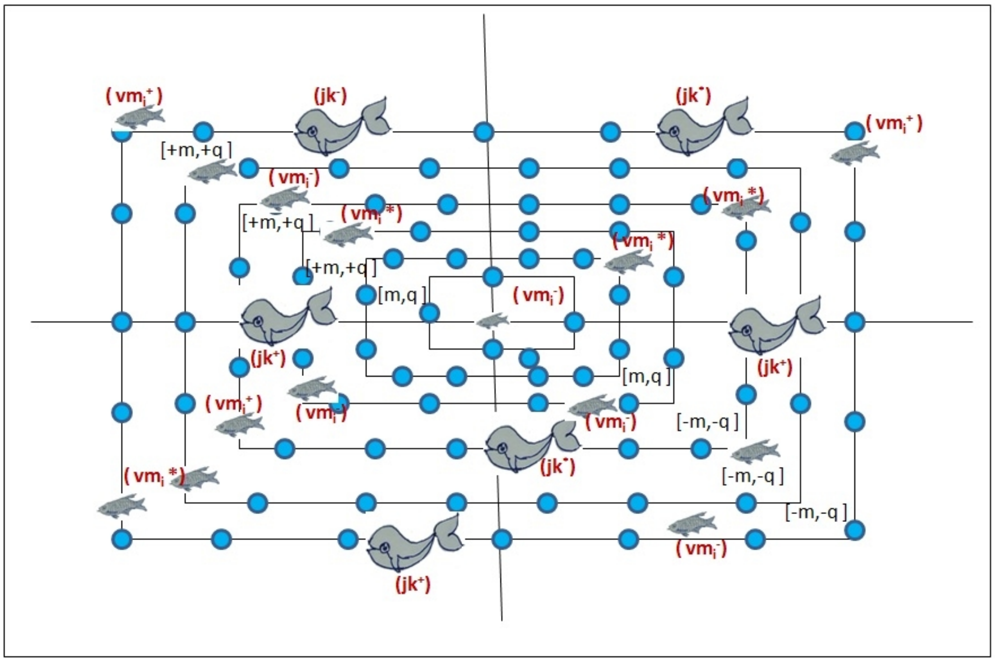


Figure 2: Bubble net hunting strategy of whale optimization load balancing technique

4.1 Performance Modeling

The virtual machines in p_k are considered as whales and the jobs as preys. The search agents population set *SA* is initialized to null and the fitness of every search agent in the search agent population set i.e., $sa_i \in SA$ is computed then the best fit search agent is selected randomly. Encircling of the prey happens by considering the current solution as best solution, because optimal solution in the search space is not known earlier and while encircling the search agent hypercube movements are performed around the neighborhood of the prey. After encircling operation, exploitation phase is initiated which tries to mimic the strategy used by the humpback whale to develop bubble net around the prey using two operations i.e., shrinking of the encircling operation and spiral updating operation. The shrinking operation determines the new position of the search agent in-between the original position of the search agent and the position of the current best agent, the spiral updating operation forms a spiral between the whale and the prey to mimic the helix movement of the humpback whale. At last in the exploration phase, the position of the search agent is updated by the random agent as the humpback whale searches its prey randomly by knowing the position of other whales to arrive at the global optimal solution.

4.2 Total execution time

The $TE(j_k, p_k, VM)$ is the total time taken to, initialize the search agent $t_I(j_k, p_k, vm_i)$, compute the fitness of search agent $t_F(j_k, p_k, vm_i)$, perform encircling operation $t_{En}(j_k, p_k, vm_i)$, perform exploitation operation $t_{Expi}(j_k, p_k, vm_i)$ and, perform exploration operation $t_{Expo}(j_k, p_k, vm_i)$.

$$TE\,(j_k, p_k, VM) = \sum_{i=1}^{i=m}\Big[t_I\,(j_k, p_k, vm_i) + t_F\,(j_k, p_k, vm_i) + t_{En}\,(j_k, p_k, vm_i) + t_{Expi}\,(j_k, p_k, vm_i) + t_{Expo}(j_k, p_k, vm_i)\Big] \tag{5}$$

$$t_I\,(j_k, p_k, vm_i) \propto SA = \{sa_i = \varnothing, \ldots, sa_m = \varnothing\} \tag{6}$$

The $t_F\,(j_k, p_k, vm_i)$ is dependent on the makespan and budget function of the incoming jobs.

$$t_F\,(j_k, p_k, vm_i) \propto \Big[M\,(j_k) + B(j_k)\Big] \tag{7}$$

Where, $M\,(j_k)$ is the makespan function which need to be lower than the deadline of jobs, $M\,(j_k) \leq \sum_k D_{j_k}$, and $B\,(j_k)$ is the budget function which need to be lower than the user budget cost of jobs, $B\,(j_k) <= \sum UBC_{j_k}$.

The $t_{En}\,(j_k, p_k, vm_i)$ involves finding the position of current best solution and updating the position of search agent to new position.

$$t_{En}\,(j_k, p_k, vm_i) \propto [P \text{ and } R] \tag{8}$$

Where, the current position $P = |Q \star R^\star - R|$, updated position $R = |R^\star - M.P|$, $R^\star$is the optimal position of the search agent, and M, Q are the coefficient vectors.

The $t_{Expi}\,(j_k, p_k, vm_i)$ involves shrinking the encircle by updating the spiral formed.

$$t_{Expi}\,(j_k, p_k, vm_i) \propto \Big[t_{Es}\,(j_k, p_k, vm_i) + t_{Su}\,(j_k, p_k, vm_i)\Big] \tag{9}$$

The $t_{Es}\,(j_k, p_k, vm_i)$ is time taken to shrink the encircle by decreasing the coefficient vectors i.e., $t_{Es}(j_k, p_k, vm_i) \to \Big[M : [-m, +m] \text{ or } Q : [-q, +q]\Big]$ and the spiral is updated as follows $t_{Su}\,(j_k, p_k, vm_i)] = P' \star \cos{(2 \star \phi \star \alpha)}{+}R^\star$, where ϕ, α are empirical constants, and $P' = |R^\star{-}R|$. The position of the search agent is updated either by encircling operation or spiral updating operation with probability p i.e., $R = \{R^\star - M \star P \ if\ p < 0.5\ else\ P^{‘} \star \cos{(2 \star \phi \star \alpha)} + R^\star\ if\ p \geq 0.5\}$.

The $t_{Expo}\,(j_k, p_k, vm_i)$ involves updating the position of the search agent by randomly chosen agent.

$$t_{Expo}\,(j_k, p_k, vm_i) \propto [P = |Q \star R_{rand} - R|\,] \tag{10}$$

Where, R_{rand} represent random position vector and $R = |R_{rand} - M \star P|$.

The total execution time is influenced by $t_{Expo}\,(j_k, p_k, vm_i)$, the search agents of whale optimization algorithm extensively search promising solutions, this may increase its convergence rate and there is a need to adaptively vary the search vector so that the smooth transit can be achieved between exploration and exploitation in optimization. However the algorithm has only two internal parameters i.e., M and Q, tuning of these parameters become easy while arriving at optimal solutions. As a result, the overall execution time is not so high or low but keeps fluctuating always.

4.3 Response time

The $RT\,(j_k, p_k, VM)$ is the sum of the time difference between $t_I\,(j_k, p_k, vm_i)$ and $t_{Expo}\,(j_k, p_k, vm_i)$.

$$RT\,(j_k, p_k, VM) + = \sum_{i=1}^{i=m} \left[t_{Expo}\,(j_k, p_k, vm_i) - t_I\,(j_k, p_k, vm_i) \right] \tag{11}$$

The algorithm uses bubble net attacking operation of humpback whale to simulate the search for best match between jobs and virtual machines as it defines the search space in the neighborhood of best matches and the bubble net employs adaptive search strategy to update the search vector by dedicating some iterations to exploration ($|Q| \geq 1$) and remaining to exploitation ($|Q| < 1$). But the search agents abruptly changes their search policy during initial stages of the optimization, this results in gradual converge rate, as a result, the response time of the jobs increases with the increase in the number of virtual machines.

4.4 Resource utilization rate

The $RU\,(j_k, p_k, VM)$ is proportional to the augmented value of $RT\,(j_k, p_k, VM)$ and constant Φ.

$$RU\,(j_k, p_k, VM) \propto \left[RT\,(j_k, p_k, VM) * \Phi \right] \tag{12}$$

The jobs are allocated among virtual machines by mimicking the hypercube and helix movement of the humpback whale, these movements make the optimization technique as a global optimizer and it does an exhaustive search on finding best fit virtual machines by considering the QoS requirements of the jobs. The periodic evaluation of resource utilization status among the virtual machines revealed that while achieving higher accuracy in mapping there are chances of transformation from exploration to exploitation leading to inefficient utilization of resources in p_k.

4.5 Throughput

The $TH\,(j_k, p_k, VM)$ is proportional to the rate of successful execution of jobs among the virtual machines in p_k, which is influenced by $t_{Expi}\,(j_k, p_k, vm_i)$ and $t_{Expo}\,(j_k, p_k, vm_i)$.

$$TH\,(j_k, p_k, VM) \leftarrow \sum_{i=1}^{i=m} \left[t_{Expi}\,(j_k, p_k, vm_i) + t_{Expo}\,(j_k, p_k, vm_i) \right] \tag{13}$$

The whale optimization algorithm often fails during the initial iteration of the optimization while finding the best match between the jobs and virtual machines and the tendency to converge to the local optimal solution is also very high. But over iterations the technique achieves balance between exploration and exploitation phases, this reduces the chances of jobs suffering from the break down while execution but increases the migration rate of jobs, as a result, the rate of successful completion of jobs becomes static over iterations of optimization.

5 Spider LBT

The methods using foraging behavior of spiders in their social colony to balance the load across the nodes in the cloud is being considered as the reference for performance modeling. The methods considered are Social Spider Cloud Web algorithm (SSCWA), Chaotic social spider load balancing algorithm, Load Balanced Task Allocation based on Social Spider Optimization (LBTA_SSO), and spider mesh overlay [22–28]. The spiders interact with each other through vibrations and they also vary their intensity of the vibrations with respect to distance, this feature of the spider helps in properly identifying the position of the resources in the cloud and reduces the situation of load imbalance due to premature convergence to locally optimal solutions. The social interaction among the spiders within the partitions web and across the partitions web to balance the load among the widely distributed virtual machines is shown in Figure 3.

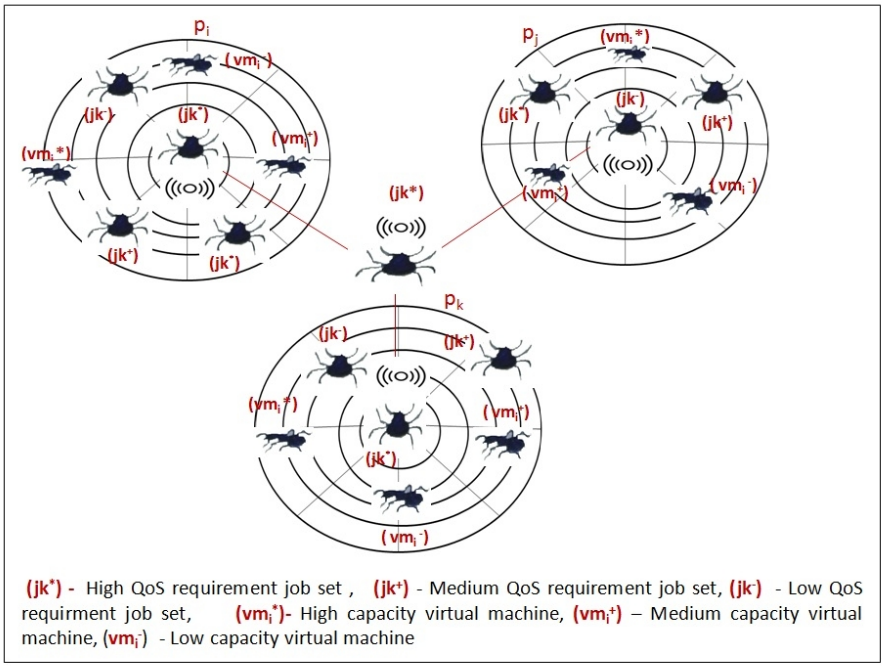

p_i
p_j
p_k
(jk*)
(jk^+)
(jk^-)
(vm_i*)
(vm_i^+)
(vm_i^-)
(jk*) - High QoS requirement job set, (jk^+) - Medium QoS requirement job set, (jk^-) - Low QoS requirment job set, (vm_i*)- High capacity virtual machine, (vm_i^+) – Medium capacity virtual machine, (vm_i^-) - Low capacity virtual machine

Figure 3: Social interaction behavior of spider load balancing technique

5.1 Performance Modeling

The jobs are represented as spiders and virtual machines are treated as preys or food sources. The population of jobs in j_k are initialized with the spider parameters like position, fitness, and vibration intensity $j_k =< p, f, vi >$. The virtual machines in p_k act as resources of the cloud and its fitness is measured in terms of its utilization capacity $f(vm_i) =< uc >$. The jobs with high QoS requirement tends to vibrate more in the colony of the web, and to prevent confusion among the jobs in acquiring the same set of virtual machines, the intensity attenuation rate is also varied over a distance. The process of finding the appropriate virtual machine to job set is carried out until a stopping condition is reached and the memory of the job is updated with the best matching virtual machine addresses.

5.2 Total execution time

The $TE\,(j_k, p_k, VM)$ is the total time taken to, initialize jobs and virtual machines with spider and prey parameters $t_I\,(j_k, p_k, vm_i)$, generate vibrations $t_V\,(j_k, p_k, vm_i)$, vary the intensity of the vibration $t_{IV}\,(j_k, p_k, vm_i)$, and allocate the jobs to virtual machines with matching vibrations $t_A\,(j_k, p_k, vm_i)$.

$$TE\,(j_k, p_k, VM) = \sum_{i=1}^{i=m} \left[t_I\,(j_k, p_k, vm_i) + t_V\,(j_k, p_k, vm_i) + t_{IV}\,(j_k, p_k, vm_i) + t_A\,(j_k, p_k, vm_i) \right] \tag{14}$$

$$t_I\,(j_k, p_k, vm_i) \propto \left[j_k < p, f, vi >= \Phi + vm_i < uc >= \Phi \right] \tag{15}$$

The $t_V(j_k, p_k, vm_i)$ is dependent on the position of the virtual machine pos_i, maximum resource utilization ratio of the virtual machine $RU_{\max}$, and minimum resource utilization ratio of the virtual machine

$$t_V(j_k, p_k, vm_i) \propto \begin{cases} \frac{1}{(RU_{\max} - f(pos_i))}, & \text{if } RU_{\max} = 1 \\ \frac{1}{(RU_{\min} - f(pos_i))}, & \text{if } RU_{\min} = 1 \end{cases} \tag{16}$$

Where $f(pos_i)$ represent the fitness of the virtual machine at pos_i.

$$t_{IV}(j_k, p_k, vm_i) \propto t_I(j_k, p_k, vm_i) * e^{\frac{-D(pos_i, pos_j)}{UP}} \tag{17}$$

Where D is the distance between virtual machine at position $i\ pos_i$ and job at position $j pos_j$, $D(pos_i, pos_j) = \sqrt{|pos_i^2| + |pos_j^2| - 2 * pos_i * pos_j}$, and UP is the parameter under reign of user.

$$t_A(j_k, p_k, vm_i) \propto j_k \rightarrow best(vm_i) \subseteq VM \tag{18}$$

Where, $best(vm_i)$ is the virtual machine with best vibration intensity.

The main objective of the spider algorithm is to keep the vibration intensity value positive. After attaining the positive intensity value, a random walk of spider is carried out to locate the best matching virtual machine for the jobs. As a result, the execution time does not float much and remains constant over time.

5.3 Response time

The $RT(j_k, p_k, VM)$ is the sum of the time elapsed between the $t_I(j_k, p_k, vm_i)$ and $t_A(j_k, p_k, vm_i)$.

$$RT(j_k, p_k, VM) + = \sum_{i=1}^{i=m} \left[t_A(j_k, p_k, vm_i) - t_I(j_k, p_k, vm_i) \right] \tag{19}$$

The spider algorithm uses vibration intensity based search mechanism to map j_k to appropriate $vm_i \in VM$ but the vibration intensity exhibited by the jobs drops exponentially and becomes more restrictive when the jobs arrival rate increases, as a result, the response time goes high.

5.4 Resource utilization rate

The $RU(j_k, p_k, VM)$ is inversly proportional to augmented value of $RT(j_k, p_k, VM)$ and constant Φ.

$$RU(j_k, p_k, VM) \propto 1/\left[RT(j_k, p_k, VM) * \Phi \right] \tag{20}$$

The $RU(j_k, p_k, VM)$ is influenced by the accuracy of the mapping of j_k to appropriate $vm_i \in VM$ in p_k. The accuracy of mapping is low because the attenuation of the vibration is varied with respect to distance and random walk mechanism is followed to locate best matching virtual machine which leads to premature convergence whenever the virtual machines try to exhibit unsettled behavior, as a result, the resources remains in the substantial stage for a long time.

5.5 Throughput

The $TH(j_k, p_k, VM)$ is dependent on the vibration intensity of the spider parameter of jobs and prey parameter of the virtual machines.

$$TH(j_k, p_k, VM) \longleftarrow \sum_{i=1}^{i=m} \left[vi(j_k) + vi(vm_i) \right] \tag{21}$$

The spider algorithm automatically changes the vibration attenuation rate and many QoS parameters are taken into consideration during vibration. This feature exploits the global optimal match between j_k and $vm_i \in VM$ but if the vibration rate of several job set remains same for long time, there are chances of collision between the job set for similar type of virtual machines. As a result, the rate of successful completion of the jobs decreases with the increase in the similar type of virtual machines.

6 Dragonfly LBT

The methods using static and dynamic swarming behaviors of dragonflies for load balancing in the cloud is being considered as the reference for performance modeling. The methods considered are Deadline Aware Multi-Objective Dragonfly Optimization (DAMO), Dragonfly Optimization Algorithm (DOA), dragonfly algorithm with dragonfly parameters, and constraint measure based dragonfly optimization [29–33]. The interaction among the dragonflies while navigating, their food search procedure and even the steps followed by them to avoid enemies while swarming is used to provide best load balancing solution in crucial situations especially when the ratio of incoming jobs and available resources in the nodes are inappropriate. The dragonfly swarming behavior exhibited by jobs within partitions by forming the swarm of jobs of varying size to move towards best fit virtual machines is shown in Figure 4.

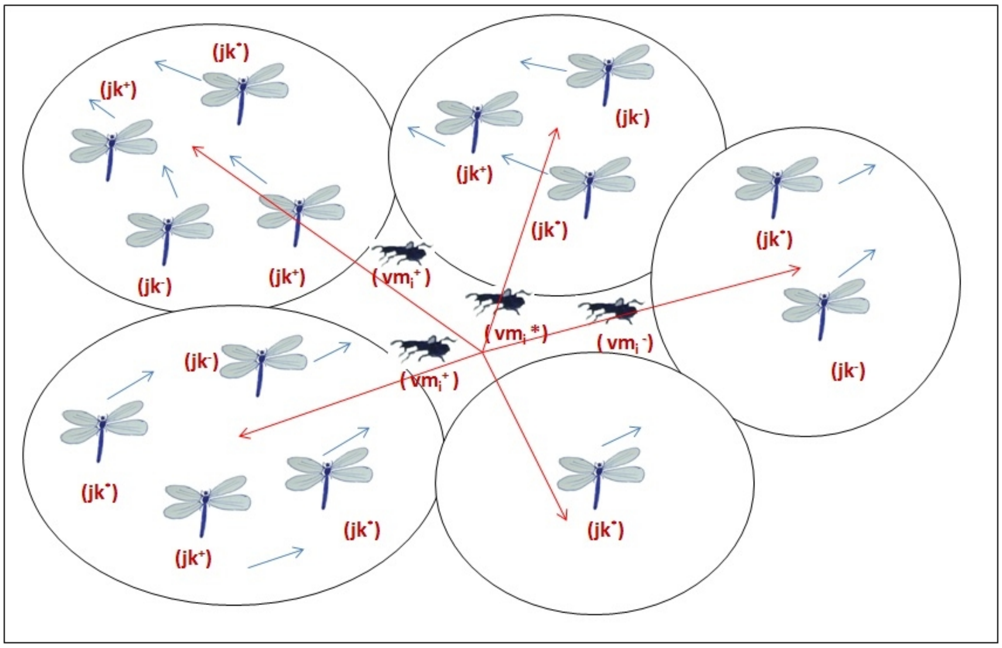


Figure 4: Swarming behavior of dragonfly load balancing technique

6.1 Performance Modeling

The jobs are represented as dragonflies and virtual machines are treated as food sources. The population of jobs in j_k are initialized with the dragonfly factors like separation, alignment, cohesion, attraction, and distraction $j_k =< s, al, c, at, d >$. Separation is the parting of one dragonfly from other dragonflies to avoid static collision in the neighborhood during food search, alignment is the matching of the one dragonfly velocity towards other dragonflies in the neighborhood, cohesion is the urge towards the center of the neighborhood, attraction is the interest towards the food source, and distraction is the disturbance caused by the movement of the enemies. The movement of the dragonflies is guided by step vector and position vector ($\Delta S,\ \Delta P$), which updates the position of the dragonflies in the large search space and navigates their movement until the stopping criteria is fulfilled.

6.2 Total execution time

The $TE\,(j_k, p_k, VM)$ is the total time taken to, initialize the dragonfly parameters of the job along with its step vector, and the position vector $t_I(j_k, p_k, VM)$, calculate separation factor of job from others jobs $t_S(j_k, p_k, VM)$, calculate alignment of job towards other jobs $t_{Al}(j_k, p_k, VM)$, calculate cohesion of jobs towards mass of neighborhood $t_C(j_k, p_k, VM)$, calculate the attraction towards the virtual machine $t_{At}(j_k, p_k, VM)$, calculate the distraction from the enemies $t_D(j_k, p_k, VM)$, and update the step vector and position vector $t_U(j_k, p_k, VM)$.

$$TE\,(j_k, p_k, VM) = \sum_{i=1}^{i=m}\Big[t_I\,(j_k, p_k, VM) + t_S\,(j_k, p_k, VM) + t_{Al}\,(j_k, p_k, VM) + t_C\,(j_k, p_k, VM) \tag{22}$$

$$+\, t_{At}\,(j_k, p_k, VM) + t_D\,(j_k, p_k, VM)\Big] + t_U(j_k, p_k, VM)]$$

$$t_I\,(j_k, p_k, VM) \propto \left[j_k < p, f, vi >= \Phi, \Delta S = \phi, \Delta P = \Phi\right] \tag{23}$$

$$t_S\,(j_k, p_k, VM) \propto \left[\sum_{i=1}^{i=N} j_k - j_k^i\right] \tag{24}$$

Where, N is the number of neighboring jobs.

$$t_{Al}\,(j_k, p_k, VM) \propto \left[\sum_{i=1}^{i=N} V(j_k^i)/N\right] \tag{25}$$

Where, V is the velocity of the i$^{th}j_k$ among the $N\ j_k^{\prime}s$ considered.

$$t_C\,(j_k, p_k, VM) \propto \left[\sum_{i=1}^{i=N} V(j_k^i/N - j_k)\right] \tag{26}$$

$$t_{At}\,(j_k, p_k, VM) \propto \left[P^{+} - P\right] \tag{27}$$

$$t_D\,(j_k, p_k, VM) \propto [P^{-} - P] \tag{28}$$

Where P is the position of the job, P^{+} is the position of the virtual machine, and P^{-} is the position of the enemy.

$$t_U\,(j_k, p_k, VM) \propto [\Delta S_{t+1} \Delta P_{t+1}] \tag{29}$$

Where, $\Delta S_{t+1} = [w * (s + al + c + at + d) + w * \Delta S_t]$, $\Delta P_{t+1} = \left[\Delta P_t + \Delta S_{t+1}\right]$, t and $t + 1$ are the iterations considered.

The jobs with high alignment weight and low cohesion weight are handled during exploration stage and the jobs with low alignment weight and high cohesion weight are handled during exploitation stage, thereby the algorithm properly balances between exploration and exploitation this increases the rate of convergence, as a result, the execution time of the dragonfly algorithm is reduced.

6.3 Response time

The $RT\,(j_k, p_k, VM)$ is the sum of the time elapsed between the $t_I\,(j_k, p_k, VM)$ and $t_U\,(j_k, p_k, VM)$.

$$RT\,(j_k, p_k, VM) + = \sum_{i=1}^{i=m} \left[t_U\,(j_k, p_k, VM) - t_I\,(j_k, p_k, VM) \right] \tag{30}$$

The response time of the dragonfly algorithm is dependent on the adaptive tuning capability of the swarm factors *s, al, c, at* and *d*. As more factors need to be handled, the tuning time is high in the initial iteration, but by adaptively changing the weights of the factors the algorithm achieves smooth transition between exploration and exploitation and arrive at the globally optimal solution, as a result, the response time is found to be average.

6.4 Resource utilization

The $RU\,(j_k, p_k, VM)$ is proportional to augmented value of $RT\,(j_k, p_k, VM)$ and constant Φ.

$$RU\,(j_k, p_k, VM) \propto [RT\,(j_k, p_k, VM) * \Phi] \tag{31}$$

In dragonfly algorithm, the radii of the neighborhood are increased with the increase in the number of optimization iterations, this increases the convergence towards promising search spaces and causes divergence outwards from not so promising search spaces, as a result, the resource utilization is high.

6.5 Throughput

The $TH\,(j_k, p_k, VM)$ is dependent on the cohesion and alignment factors of the dragonfly algorithm.

$$TH\,(j_k, p_k, VM) \leftarrow \sum_{i=1}^{i=m} \left[t_C\,(j_k, p_k, VM) + t_{Al}\,(j_k, p_k, VM) \right] \tag{32}$$

The dragonfly algorithm automatically changes the attenuation rate of cohesion and alignment factors to increase their neighborhood area. They adjust their flying path to converge to the global optimum solution and uses levy flight mechanism to exhaustively search optimal solution when there are no neighboring solutions. The rate of successful completion of jobs is high when neighboring solutions do exist but in the absence of the neighboring solution, the successful completion of jobs is found to be average.

7 Raven Roosting LBT

Few methods dealing with load distribution strategies in the cloud environment using the foraging behavior and social roosting of raven birds are considered as the reference for performance modeling. The methods considered are Raven Roosting Optimization Algorithm (RROA), and Improved Raven Roosting Optimization Algorithm (IRROA) [34–39]. The roosting place of raven birds act as the information center to broadcast the data related to food sources in their environment, they usually check for the availability of sufficient food in their neighboring locations if it is found then they move towards that location in search of food else they seek for some other location. Likewise, every bird has its own private knowledge about the food sources based on their past experience, this act as deciding factor whether to move towards the food source or to find alternate food sources. These features of the raven roosting algorithm make it strong enough to handle the overload/under-load situations in larger computing domain like the cloud. The social roosting behavior of ravens to follow the leader or unfollow the leader to find the large quantity of food source is mimicked by the jobs to find the suitable virtual machines as shown in Figure 5.

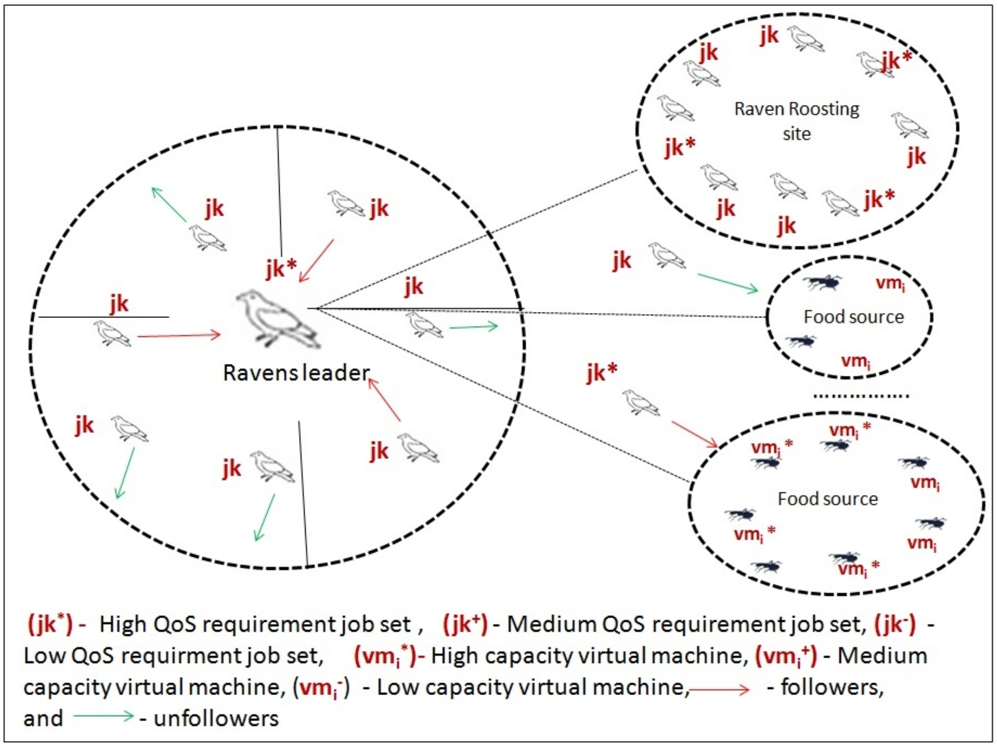

Raven Roosting site
jk
jk*
Ravens leader
Food source
vm_i
vm_i *
(jk*) - High QoS requirement job set, (jk+) - Medium QoS requirement job set, (jk-) - Low QoS requirment job set, (vm_i*)- High capacity virtual machine, (vm_i+) - Medium capacity virtual machine, (vm_i-) - Low capacity virtual machine, ⟶ - followers, and ⟶ - unfollowers

Figure 5: Social roosting behavior of raven roosting load balancing technique

7.1 Performance Modeling

The incoming jobs are considered as ravens and the virtual machines in p_k are considered as food sources, at first, the ravens are distributed among the food sources randomly. Then the fitness of every raven location is computed and the personal best location of the raven is updated and is treated as the leader. A portion of ravens is recruited to follow the leader, which starts to forage by selecting a random point within the sphere of the leader and the unrequited portion of the ravens go to their personal best and start to forage there. The process of food search is continued by updating the step length of raven movement until the highest quality food location is found.

7.2 Total execution time

The $TE\,(j_k, p_k, VM)$ is the total time taken to, assign jobs to random virtual machines $t_A\,(j_k, p_k, vm_i)$, select the leader among the jobs $t_L\,(j_k, p_k, vm_i)$, compute the step size for job movement $t_{SS}\,(j_k, p_k, vm_i)$, decide followers and unfollowers of the leader $t_{F-UF}\,(j_k, p_k, vm_i)$, and update the personal best of the job $t_{Pbest}\,(j_k, p_k, vm_i)$.

$$TE\,(j_k, p_k, VM) \sum_{i=1}^{i=m} \begin{bmatrix} t_A\,(j_k, p_k, vm_i) + \; t_L\,(j_k, p_k, vm_i) + t_{SS}\,(j_k, p_k, vm_i) \\ +t_{F-UF}\,(j_k, p_k, vm_i) + t_{Pbest}\,(j_k, p_k, vm_i) \end{bmatrix} \tag{33}$$

$$t_A\,(j_k, p_k, vm_i) \propto \sum_{i=1}^{i=M} [j_k \to vm_i] \tag{34}$$

$$t_L\,(j_k, p_k, vm_i) \propto JL = \max(f\,(j_k)) \tag{35}$$

Where, fittest job set $f(j_k) = \{f(j_1), f(j_2), \ldots, f(j_k)\}$.

$$t_{SS}(j_k, p_k, vm_i) \propto \sum_{m=1}^{m=M} \left[pos_m = pos_{m-1} + ss_m\right] \tag{36}$$

Where, pos_m is the current position of the job, pos_{m-1} is the previous position of the job, and ss_m is the randomly chosen step length of the job.

$$t_{F-UF}(j_k, p_k, vm_i) \propto \left[\frac{k}{2} \in JL \text{ and } k - \frac{k}{2} \notin JL\right] \tag{37}$$

$$t_{Pbest}(j_k, p_k, vm_i) \propto \begin{cases} neighborhood(JL), & \text{if } pbest(JL) > pbest(j_k) \\ neighborhood(j_k), & \text{if } pbest(j_k) > pbest(JL) \end{cases} \tag{38}$$

Where, *neighborhood(JL)* is the neighborhood of the leader, and *pbest(JL)* is the personal best of the leader.

Raven roosting algorithm follows a simple perception mechanism where the ravens either follow the leader or goes with their personal best choice and stochastically stops on finding the best location; this increases the convergence rate of the algorithm and meanwhile reduces its total execution time.

7.3 Response time

The $RT(j_k, p_k, VM)$ is the sum of the time difference between $t_A(j_k, p_k, vm_i)$ and $t_{Pbest}(j_k, p_k, vm_i)$.

$$RT(j_k, p_k, VM) += \sum_{i=1}^{i=m} \left[t_{Pbest}(j_k, p_k, vm_i) - t_A(j_k, p_k, vm_i)\right] \tag{39}$$

The response time of the raven roosting algorithm is high during initial iterations of the algorithm as the ravens are randomly distributed among the food locations and even the step size is also randomly chosen to navigate the movement of ravens but over iterations due to the update of the memory of the ravens to the P_{best} of the location of food source the response time gradually reduces.

7.4 Resource utilization

The $RU(j_k, p_k, VM)$ is proportional to augmented value of $RT(j_k, p_k, VM)$ and constant Φ.

$$RU(j_k, p_k, VM) \propto \left[RT(j_k, p_k, VM) * \Phi\right] \tag{40}$$

In raven roosting algorithm, every individual raven maintains personal memory which remembers the location of the best food source rather than only trusting the location of the ravens leader, as a result, the accuracy of the mapping of the raven to appropriate food sources increases which in turn leads to efficient resource utilization.

7.5 Throughput

The $TH(j_k, p_k, VM)$ depends on the successful execution of jobs, which is influenced by the $t_{SS}(j_k, p_k, vm_i)$ and $t_{F-UF}(j_k, p_k, vm_i)$.

$$TH(j_k, p_k, VM) \leftarrow \sum_{i=1}^{i=m} \left[t_{SS}(j_k, p_k, vm_i) + t_{F-UF}(j_k, p_k, vm_i)\right] \tag{41}$$

The rate of completion of jobs is lower in initial iterations as the ravens suffer from neophobia (fear of new food locations), they exhibit some sort of reluctance towards new food locations even if the location has sufficient amount of resources. However, over iteration the rate of completion of jobs increases due to the capability of the ravens to memorize the successful/unsuccessful food foraging locations.

The behavior of the swarm intelligence based techniques like a whale, spider, dragonfly, and raven roosting, towards the performance metrics like total execution time, response time, resource utilization rate, and throughput are summarized as follows. Among the four load balancing techniques considered, the whale and spider exhibits high similarity, as both of the approaches adopt the flexible searching strategy. The two techniques use either bubble-net hunting strategy or vary the vibration intensity of the spider to arrive at proper load balancing strategies with minimum search episodes. Whereas the behavior of dragonfly and raven roosting are unique compared to all four techniques considered for modeling in terms of search strategy optimization. The dragonfly vary the size of the swarms while navigating towards the solution and even make sure that it does not get distracted from enemies. This feature helps in arriving at the global optimal solution, even in the presence of several sub-optimal solutions for load balancing problem. The raven roosting does both global and local search, as every raven bird uses its private knowledge and knowledge of raven leader while taking decisions. The raven roosting and dragonfly outperforms all other as both of them utilize the resources efficiently by preventing the situation of overloading or under-loading of the resources using roosting behavior of ravens.

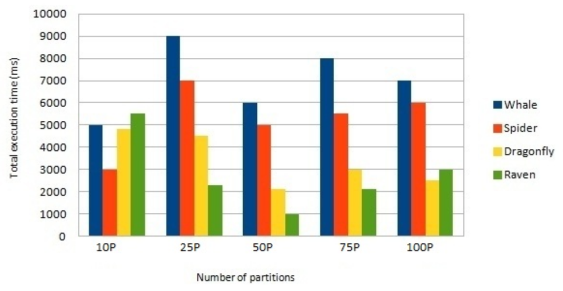
8 Results and Discussion

For simulation purpose, CloudAnalyst visual modeling tool based on CloudSim simulator is used, it consists of two major components one is the partition and the other is the job [40]. The partition constitutes set of virtual machines, memory, disk storage, and bandwidth. The job constitutes incoming requests per hour, the location of clients, and size of the requests. For experiment purpose, the number of partitions are varied from 10 to 100, the number of jobs for every partition is varied from 100000 to 2200000, virtual machines in every partition is varied from 10000 to 30000 and the jobs are also classified into five types with fixed size which includes *computing jobs, memory jobs, CPU jobs, transmission jobs, and retrieval jobs*. Here the performance achieved by the whale, spider, dragonfly, and raven are discussed with respect to performance metrics like total execution time, response time, resource utilization rate, and throughput in three different scenarios. In the first scenario, the number of incoming jobs to every partition is fixed by varying the number of partitions, in the second scenario one partition is considered by varying the jobs, and in the third scenario variety of jobs are considered towards every partition.

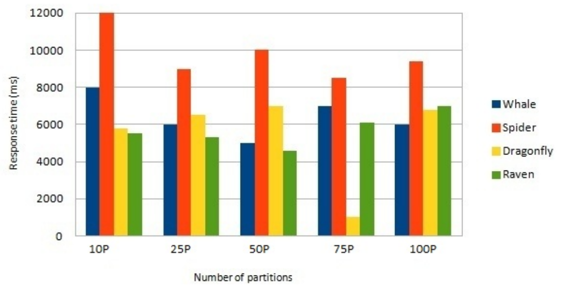
The fixed parameter scheme is used to select specific parameter values from predefined range of values for the swarm intelligence based load balancing techniques before the simulation is carried out based on the characteristics of the parameters and the same parameter range is maintained throughout the whole search space and the parameter values are chosen randomly from the mentioned range. The range of values considered for evaluating the load balancing techniques are as follows. For the whale load balancing technique, the parameters considered are number of search agents= {5, 10, 15, 20, 25,. . . ,1000}, and the search vector= {10 to 0, 20 to 0, or 40 to 0}. For the spider load balancing technique, the parameters considered are: Position= {1, 2, 3,. . . ,1000}, fitness=$\{10^{-3}, 10^{-4}, \dots, 10^{-8}\}$, and vibration intensity={1/10, 1/5, 1/4, 1/3, 1/2}. For the dragonfly load balancing technique, the parameters considered are step vector = {0.2, 0.6, 0.8, 1.0}, and position vector= {1, 3, 5, 7, 9}. For the raven roosting load balancing technique, the parameters considered are step length= {4, 8, 12, 18,. . . ,120}. The number of runs considered for all load balancing techniques are {100,200,300,400,500,600}. After certain number of evaluations, the final parameter setting based on which the results are generated are as follows. For the whale load balancing technique, the parameters set are search agents= 25, search vector=30 to 0. For the spider load balancing technique, the parameters set are position=500, fitness=10^{-6}, and vibration intensity=1/3. For the dragonfly load balancing technique, the parameters set are step vector =0.6, position vector=5. For the raven roosting load balancing technique, the parameters set is step length=25. To check the correctness of the results obtained, all load balancing techniques are executed for 500 iterations.

8.1 Scenario-1

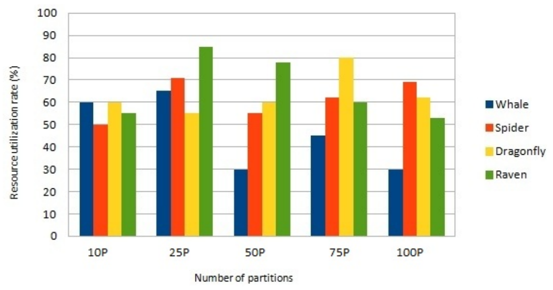
In scenario 1, the number of incoming jobs towards every partition is fixed to 2000000 and the data center partitions are varied from 1 to 100. The partitions are distributed randomly over the large geographical region and the settings related to the partitions are also fixed. The performance of the whale, spider, dragonfly, and raven techniques of load balancing under scenario 1 is depicted in Figure 6. With respect to total execution time, with the increase in partitions, the total execution time of dragonfly and raven is lower; the execution time of spider remains average, whereas the execution time of the whale is found to be higher. With respect to response time, the spider's response time is found to remain higher with the increase in partitions but the response time of whale, dragonfly, and raven remained in average range. With respect to resource utilization rate, dragonfly, spider, and raven remained above average, whereas there is a considerable drop in resource utilization rate of the whale. With respect to throughput, the raven achieved outstanding successful job completion rate, the job completion rate of the dragonfly is above average, whereas the job completion rate of whale and spider remain in an average range.



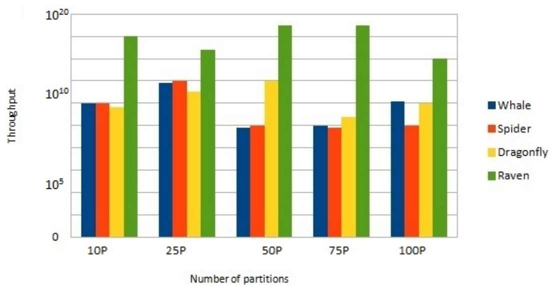
(a) Total execution time



(b) Response time



(c) Resource utilization rate

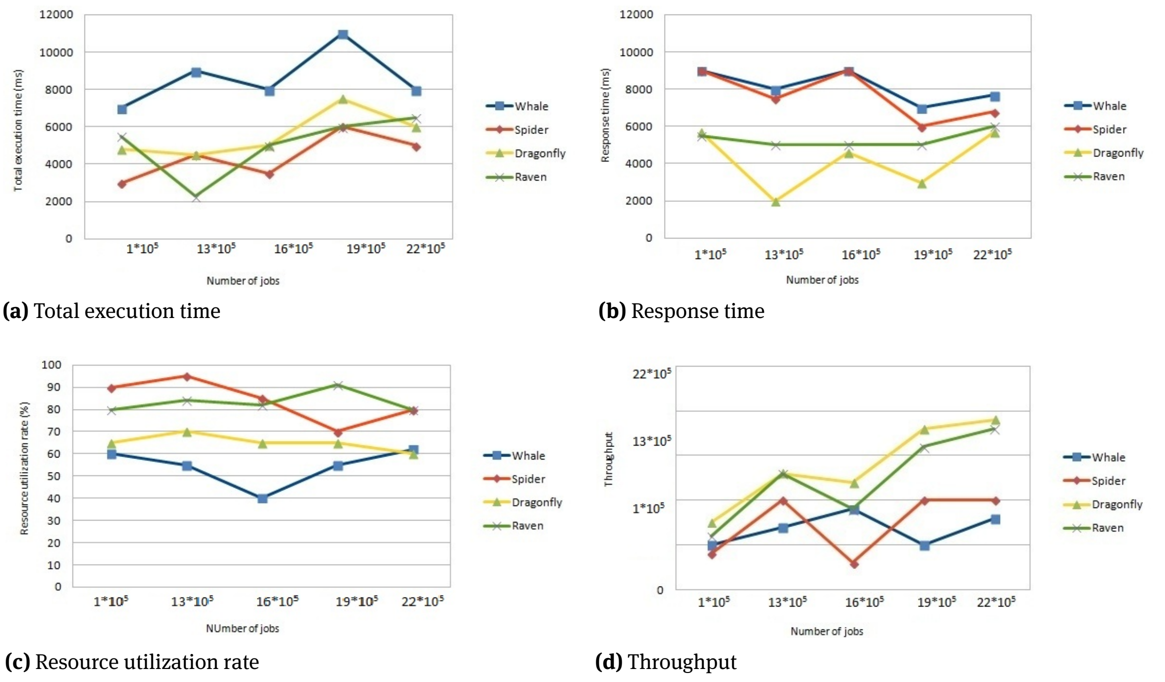


(d) Throughput

Figure 6: Performance under scenario-1

8.2 Scenario-2

In scenario 2, one partition is being considered and the incoming jobs are varied from 100000 to 2200000. The performance of the whale, spider, dragonfly, and raven techniques of load balancing under scenario 2 is depicted in Figure 7. With respect to total execution time; with the increase in jobs, the total execution time of whale is found to be high whereas the execution time of spider, dragonfly, and raven remains average. With



(a) Total execution time

(b) Response time

(c) Resource utilization rate

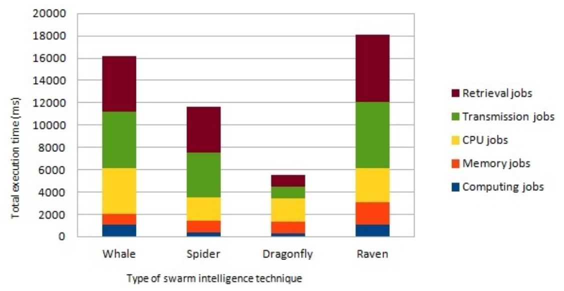
(d) Throughput

Figure 7: Performance under scenario-2

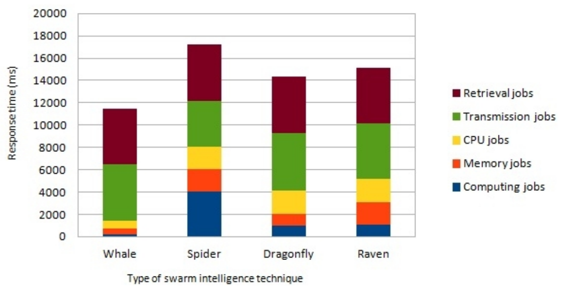
respect to response time; the response time of whale and spider is found to be higher; the response time of raven is in the average range whereas the response time of dragonfly is lower especially when more jobs are being considered. With respect to resource utilization rate; the resource utilization rate of the raven and the spider is found to be higher than whale and dragonfly. With respect to throughput; the job completion rate of the dragonfly, and raven consistently kept higher with the increase in the jobs but an uneven pattern in job completion rate of spider and the whale is observed.

8.3 Scenario-3

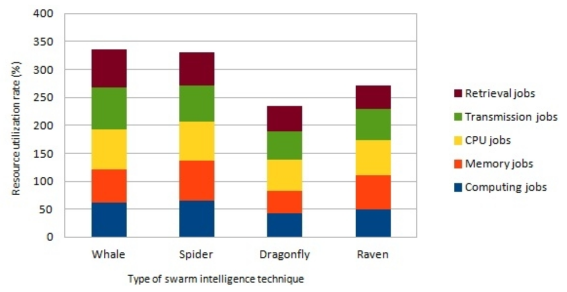
In scenario 3, varieties of incoming jobs (*computing jobs, memory jobs, CPU jobs, transmission jobs, and retrieval jobs*) in every partition are being considered. The performance of the whale, spider, dragonfly, and raven techniques of load balancing under scenario 3 is depicted in Figure 8. With respect to *computing jobs*; the total execution time of dragonfly and spider is lower whereas the execution time of whale and raven is higher, the response time of whale is found to be lower, the response time of raven and dragonfly is average but the response time of the spider is found to be higher, the resource utilization rate of whale, spider and raven is higher compared to dragonfly, the rate of successful completion of jobs is found to be average for whale, spider, dragonfly, and raven. With respect to *memory jobs*; the total execution time of whale, spider, and dragonfly is lower compared to raven, the response time of whale and dragonfly is lower compared to spider and raven, the resource utilization rate of dragonfly is lower compared to whale, spider, and raven, the rate of completion of jobs is higher for whale, dragonfly, and raven compared to spider. With respect to *CPU jobs*; the total execution time of all algorithms i.e., whale, dragonfly, raven, and spider is found to be average, the response time of whale and dragonfly is lower compared to spider and raven, the resource utilization rate of all algorithms i.e., whale, dragonfly, raven, and spider is found to be average, the rate of successful completion of jobs is found to be higher for whale, dragonfly, and raven compared to spider. With respect to transmission jobs, the total execution time of dragonfly is lower compared to whale, spider, and raven, the response time of all algorithms i.e., raven, dragonfly, spider, and whale is found to be below average, the resource utilization rate of all algorithms i.e., raven, dragonfly, spider, and whale is found to be high, the rate



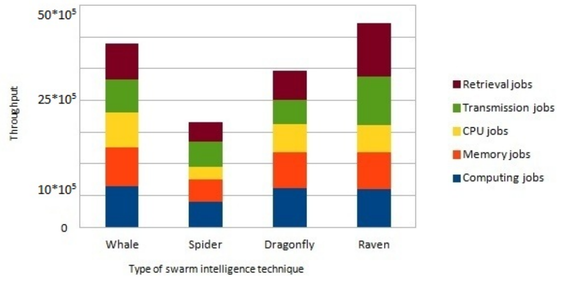
(a) Total execution time



(b) Response time



(c) Resource utilization rate



(d) Throughput

Figure 8: Performance under scenario-3

of successful completion of jobs is higher for raven compared to whale, spider, and dragonfly. With respect to *retrieval jobs*, the total execution time of dragonfly and spider is lower compared to whale and raven, the response time of all algorithms i.e., whale, spider, dragonfly, and raven fall in average range, the resource utilization rate of spider and whale is higher compared to dragonfly and raven, the rate of successful completion of jobs is higher for raven compared to whale, spider, and dragonfly.

9 Conclusion

This paper presents performance modeling of some of the recent competitive swarm artificial intelligence based load balancing techniques like the whale, spider, dragonfly, and the raven with respect to performance metrics like total execution time, response time, resource utilization rate, and throughput. The performance of the techniques are tested using CloudSim simulator under three different scenarios i.e., data center partitions variation, jobs variation, and considering the variety of jobs. The overall performance of dragonfly and raven is found to be better as it efficiently distributes the load among virtual machines with reduced execution time, reduced response time, efficient resource utilization rate, and increased rate of job completion rate. The performance of the spider is found to be average as it shortened the response time, efficiently utilized the resources but increased the execution time by achieving the average rate of successful completion of jobs. However, the performance of whale is found to be below average as its effectiveness is only towards one or two metrics out of the five performance metrics being considered for evaluation.

References

[1] A. D. Josep, R. Katz, A. KonWinSKi, L. E. E. Gunho, D. Patterson and A. Rabkin, A view of cloud computing. *Communications of the ACM*, 53(2010), 130-143.

[2] R. Z. Khan and M. O. Ahmad, Load balancing challenges in cloud computing: a survey, *in: Proceedings of the International Conference on Signal, Networks, Computing, and Systems*. pp. 25-32, 2016.

[3] A. S. Kumar and P. Tripathi, Various issues and challenges of load balancing over cloud: a survey, *International Journal Of Engineering And Computer Science*, 5(2016), 17517-17524.

[4] V. K. Reddy, K. D. Surya, M. S. Praveen, B. Lokesh, A. Vishal and K. Akhil, Performance analysis of Load Balancing Algorithms in cloud computing environment, *Indian Journal of Science and Technology*, *9*(2016), 1-7.

[5] R. Kumar and T. Prashar, Performance analysis of load balancing algorithms in cloud computing, *International Journal of Computer Applications*, 120(2015), 19-27.

[6] M. Awan and M. A. Shah, A survey on task scheduling algorithms in cloud computing environment, *International Journal of Computer and Information Technology*, 4(2015), 441-448.

[7] I.. Fister Jr, X. S. Yang, I. Fister, J. Brest and D. Fister, A brief review of nature-inspired algorithms for optimization, *arXiv preprint arXiv:1307.4186*, 80(2013), 1-7.

[8] M. Dixit, N. Upadhyay and S. Silakari, An exhaustive survey on nature inspired optimization algorithms, *International Journal of Software Engineering and Its Applications*, *9*(2015), 91-104.

[9] E. Elbeltagi, T. Hegazy and D. Grierson, Comparison among five evolutionary-based optimization algorithms, *Advanced engineering informatics*, 19(2005), 43-53.

[10] X. S. Yang, Recent advances in swarm intelligence and evolutionary computation*, Berlin: Springer* , 585(2015).

[11] T. O. Ting, X. S. Yang, S. Cheng and K.Huang, Hybrid metaheuristic algorithms: past, present, and future. In *Recent advances in swarm intelligence and evolutionary computation,* pp. 71-83, 2015.

[12] A. Slowik and H. Kwasnicka, Nature inspired methods and their industry applications-Swarm intelligence algorithms, *IEEE Transactions on Industrial Informatics*, 14(2018), 1004-1015.

[13] M. Kalra and S. Singh, A review of metaheuristic scheduling techniques in cloud computing. *Egyptian informatics journal*, 16(2015), 275-295.

[14] H. A. Akkar and F. R. Mahdi, Evolutionary algorithms performance comparison for optimizing unimodal and multimodal test functions, *International Journal of Scientific & Technology Research*, 4(2015), 38-45.

[15] P. P. Prajapati and M. V. Shah, Performance estimation of differential evolution, particle swarm optimization and cuckoo search algorithms, *International Journal of Intelligent Systems and Applications*, 10(2018), 59-67.

[16] S. Mirjalili and A. Lewis, The whale optimization algorithm, *Advances in engineering software*, 95(2016), 51-67.

[17] K. Sreenu and M. Sreelatha, W-Scheduler: whale optimization for task scheduling in cloud computing, *Cluster Computing*, (2017), 1-12.

[18] I. N. Trivedi, J. Pradeep, J. Narottam, K. Arvind, K. and L. Dilip, Novel adaptive whale optimization algorithm for global optimization. *Indian Journal of Science and Technology*, 9(2016), 319-326.

[19] H. Hu, Y. Bai and T. Xu, T, A whale optimization algorithm with inertia weight, *WSEAS Trans. Comput*, 15(2016), 319-326.

[20] H. Hu, Y. Bai, Y and T. Xu, Improved whale optimization algorithms based on inertia weights and theirs applications, *Int. J. Circuits, Systems Signal Process*, 11(2017*)*, 12-26.

[21] G. Kaur and S. Arora, Chaotic whale optimization algorithm, *Journal of Computational Design and Engineering*, 5(2018), 275-284.

[22] P. Abrol, S. Gupta, S and K. Kaur, Social spider cloud web algorithm (SSCWA): a new meta-heuristic for avoiding premature convergence in cloud, *International Journal of Innovative Research in Computer and Communication Engineering*, 3(2015), 5698-5704.

[23] E. Cuevas and M. Cienfuegos, A new algorithm inspired in the behavior of the social-spider for constrained optimization, *Expert Systems with Applications*, *41*(2014), 412-425.

[24] E. Cuevas, M. Cienfuegos, R. Rojas and A. Padilla, A computational intelligence optimization algorithm based on the behavior of the social-spider. *In Computational Intelligence Applications in Modeling and Control*, pp. 123-146, 2015.

[25] B. Shanmugapriya and S. Meera, S, A survey of parallel social spider optimization algorithm based on swarm intelligence for high dimensional datasets, *International Journal of Computational Intelligence Research*, 13(2017), 2259-2265.

[26] C. Erredir, M. L. Riabi, E. Bouarroudj and H. Ammari, Swarm optimization algorithm inspired in the behavior of the social-spider for microwave filters optimization. *In 7th African Conference on Non Destructive Testing ACNDT 2016 & the 5th International Conference on NDT and Materials Industry and Alloys (IC-WNDT-MI)*, pp. 6374-6384, 2016.

[27] B. Gunnarsson and K. Wiklander, K, Foraging mode of spiders affects risk of predation by birds, *Biological journal of the Linnean Society*, 115(2015), 58-68.

[28] J. Q. James and V. O. Li, A social spider algorithm for global optimization, *Applied Soft Computing*, 30(2015), 614-627.

[29] Z. Amini, M. Maeen and M. R. Jahangir, Providing a load balancing method based on dragonfly optimization algorithm for resource allocation in cloud computing, *International Journal of Networked and Distributed Computing*, 6(2017), 35-42.

[30] V. Polepally and K. S. Chatrapati, Dragonfly optimization and constraint measure-based load balancing in cloud computing, *Cluster Computing*, pp. 1-13, 2017.

[31] S. Mirjalili, Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems, *Neural Computing and Applications*, *27*(2016), 1053-1073.

[32] O. Bozorg-Haddad, Advanced optimization by nature-inspired algorithms, *Singapore: Springer*, 2018.

[33] D. Nanda and A. Chhabra, Deadline aware multi-objective dragonfly optimization technique for scheduling jobs in multi-cluster environment, *International Journal of Applied Engineering Research*, 13(2018), 10286-10292.

[34] E. Rani and H. Kaur, Efficient Load Balancing Task Scheduling in Cloud Computing using Raven Roosting Optimization Algorithm. *International Journal of Advanced Research in Computer Science*, *8*(2017), 2419-2424.

[35] A. Brabazon, W. Cui and M. O'Neill, The raven roosting optimisation algorithm, *Soft Computing*, *20*(2016), 525-545.

[36] S. Torabi and F. Safi-Esfahani, Improved Raven Roosting Optimization algorithm (IRRO), *Swarm and Evolutionary Computation*, 40(2018), 144-154.

[37] S. Torabi and F. Safi-Esfahani, A dynamic task scheduling framework based on chicken swarm and improved raven roosting optimization methods in cloud computing, *The Journal of Supercomputing*, 74(2018), 2581-2626.

[38] A. Braun and T. Bugnyar, Social bonds and rank acquisition in raven nonbreeder aggregations. *Animal behaviour*, 84(2012), 1507-1515.

[39] T. Bugnyar, Social cognition in ravens. *Comparative cognition & behavior reviews*, *8*(2013), 1.

[40] http://www.cloudbus.org/cloudsim/