Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2022 SCPE. Volume 23, Issues 4, pp. 321–338, DOI 10.12694/scpe.v23i4.2037

FAULT TOLERANT LOAD BALANCING WITH QUADRUPLE OSMOTIC HYBRID CLASSIFIER AND WHALE OPTIMIZATION FOR CLOUD COMPUTING

SOUNDARARAJAN ANURADHA [*] AND P. KANMANI [†]

Abstract. Cloud Computing (CC) environment is developing as a recently discovered caliber for computing applications over the network. Fault tolerance is one of the foremost issues in CC environment. Since the negligence in resource have a profound effect on job execution, throughput, response time and performance of the entire network. In this work, in order to address the issue, Quadruple Osmotic Hybrid Classification and Whale Optimization (QOHC-WO) is introduced to fault-tolerance under the requirement of different user request tasks. Initially, Quadruple Fault Tolerance Level is applied to allocate the fault tolerance level. Followed by, Hybrid Vector Classifier is used to categorize the user request tasks (task) and cloud server nodes (node). Then, the Osmotic function is employed for performing the migration among virtual machines with lesser response time. This helps to solve the maximum level of fault issue. Finally, Improved Whale Optimization Algorithm is applied to find the optimal allocation of tasks with the corresponding node. In addition, the Bandit function and Whale optimization are used to address the trade-off between exploitation and exploration. Experimental setup of the proposed QOHC-WO method and existing methods are carried out with different factors such as task response time, the number of VM migrations, and percentage of fault detected rate with respect to a number of tasks. The analyzed results validate that the proposed QOHC-WO method achieves a higher fault detection rate with minimum response time as well as task migration than the state-of-the-art methods.

Key words: Cloud Computing, Quadruple, Quadruple Fault Tolerance Degree, Osmotic, Hybrid Vector Classifier, Bandit, Whale Optimization.

AMS subject classifications. 68M14

1. Introduction.

With the increasing number of cloud providers, the scheduling issue has resulted in a hotspot in the CC environment. The scheduler must arrange the scheduling mechanism in an effective manner to use the available resources and allocate them to the requested tasks. Hence, fault tolerance in load balancing is one of the principal threats in CC, that is necessary to allocate the user request task uniformly beyond all the cloud server nodes, detect the fault and eliminate fault from the network and share user request tasks to all the cloud server nodes to improve the cloud network performance.

In [1], Crystal, a server-centric and recursively constructed structure was designed. It was made from the building blocks via vertex configuration pattern of Archimedean solids. It was also found to be intrinsically fault-tolerant due to the incorporation of multi-homing mechanism. Scaling was also doubled in an exponential manner due to its design of architecture in a recursively manner. Besides, a two routing algorithms were also designed to improve routing efficiency and administer different types of failures in the network.

Simulation results revealed that Crystal was found to be highly fault-tolerant with high throughput and also provided low latency. However, the capabilities of the server were not taken into consideration, therefore compromising the overall completion time. Objective of this work is to introduce and evaluate the proposed user request task scheduling and cloud server node load balancing by introducing Osmotic Hybrid Vector Classifier model considering the capabilities of each virtual machine (VM), therefore improving the overall completion time.

An energy-efficient fault-tolerant replica management policy was designed in [2] considering the deadline and budget constraints. Simulation results revealed that the dynamic replica placement algorithm efficiently minimized the mean job time minimized the utilization of the network bandwidth and enhanced the storage

[*]Department of Computer Science and Applications, Shri Sakthikailassh Women's College, Salem, Tamilnadu, India (anusuriyaa@gmail.com)

[†]Department of Computer Science, Tiruvalluvar Government Arts College, Rasipuram, Tamilnadu, India (pushpakanmani@gmail.com)

space usage. With the consideration of energy effectiveness, a cluster scaling mechanism considering the energy effectiveness was presented to minimize the system energy consumption by means of sleeping and waking up the data nodes based on load state. In addition, to prevent access failures and data loss due to the node failures, a node failure recovery model according to the availability metrics was utilized for node failures.

Simulation results also revealed that the proposed method was found to be better in terms of both energy efficiency and fault tolerance. However, with a trade-off between exploitation and exploration, while choosing between replicas. To address this issue, in this work, Improved Whale Optimization algorithm is presented that addresses the issues of exploitation by means of Bandit function and exploration by means of Whale optimization, therefore enhancing the percentage of fault detection.

The main contributions of the paper are summarized as follows:

1. The main contribution of the proposed QOHC-WO method is to consider the capabilities of each VM and the task length to assign the tasks into the most appropriate VMs.
2. The novelty of Quadruple Fault Tolerance Level is introduced for assigning the fault tolerance level.
3. A novel Osmotic Hybrid Vector Classifier (OHVC) model is proposed where the classification of node and task is made. The new idea of the Osmotic function is used to perform a migration between virtual machines for reducing the task response time.
4. An Improved Whale Optimization algorithm is introduced for discovering the factor regarding the right assignment of the task to the node in an optimal manner. The novelty of the Bandit function and Whale Optimization is applied to solve the exploitation issue. This helps to enhance the fault tolerance rate.
5. Finally, a series of experiments were conducted to measure the performance analysis of the proposed QOHC-WO method along with conventional methods based on various performance metrics.

The rest of this paper is structured as follows. The related work is summarized in Section 2. Section 3 introduces the problem description, system mode, and then the novel Quadruple Osmotic Hybrid Classification and Whale Optimization (QOHC-WO) method for cloud computing is presented. Section 4 conducts extensive experiments to evaluate the performance of our algorithm followed by a discussion in Section 5. Section 6 concludes the paper.

2. Literature Survey.

Customizable software systems comprises of critical, non-critical and interdependent configurations. Besides, two important factors, like, reliability and performance is heavily said to be influenced on completion of communication or interactions between the configurations. Besides, failure of critical configurations results in severe influence on both reliability and performance.

In [3], first critical configurations were first identified next a fault tolerant candidate was identified, therefore, resulting in the improvement of successful interactions. A Dynamic Fault Tolerant VM Migration (DFTM) method was designed in [4] for providing infrastructure reliability via advanced recovery mechanism. Here, Linear Programming model was used for route analysis and Alternative Switch Identification Algorithm was used to identify the new VM migration. Fault tolerant mechanism specifically, for storage and access was presented in [5] using priority-queue based scheduling, therefore resulting in the improvement of data access.

However, the high performance computing system results in higher failure rates due to huge involvement of servers filled with heavy workloads, resulting in the unavailability of systems for computation. An innovative perspective on applying a fault tolerant mechanism was presented in [6] that in turn eliminated the possibility of network congestion with migration technique to adaptively handle fault occurrences.

Yet another fault tolerant framework called Fault tolerance Algorithm Using Selective Mirrored Tasks Method(FAUSIT) was presented in [7]. Here, a balance between parallelism and the topology was first introduced, next a selective mirrored task method was utilized to enhance the fault tolerance, therefore both minimizing the make span and computing cost incurred. In[8], both the physical resource usage rate and fault tolerant requirement of the user were considered and guaranteed user service in large probability.

A survey on fault tolerance, fault tolerance methods, load balancing techniques and schemes in a CC environment was presented in [9]. Nature inspired meta-heuristic algorithm was used in [10] to solve the issues related to cloud load balancing. To achieve optimal container resource allocation, anew hybridized algorithm, called, Whale Random update assisted Lion Algorithm (WR-LA) was proposed in [11] to prove its superiority in terms of computational complexity. A novel Dynamical Load Balanced Scheduling (DLBS) approach was presented in [12] for both improving the network throughput and balancing workload in a dynamical manner. Besides, efficient heuristic scheduling algorithms were also designed that in turn balanced data flows according to different time slots.

With several overloaded controllers in the network and only one controller with maximum load is usually said to be addressed by means of a single load-balancing operation, therefore improving the load-balancing efficiency. To address this issue, in [13], a load balancing strategy with multiple controllers based on response time was designed. By using the correct response time threshold and with multiple controllers in a simultaneous manner, load-balancing issue was said to be addressed. Dynamic reconfiguration of resources based on the real time requirements was designed in [14]. However, larger number of migration was said to take place. To address this issue, a game based consolidation model with energy and load constraints was designed in [15].

The cloud schedulers utilized considered only single resource (RAM) for workload coordination that in turn resulted in violation due to non-optimal VM placement. To address this issue, a nova scheduler was proposed in [16]that utilized multi-resource based VM placement, enhancing the application performance in terms of execution time. Yet another workflow job scheduling algorithm was designed in [17] on the basis of load balancing, therefore minimizing the system response time issue. A global league championship algorithm was used in [18] for scheduling of the task globally in cloud environment, therefore significantly minimizing the task execution time. However, the user's quality of service requirement was not satisfied. An improved ant colony algorithm was proposed in [19] to improve make span, throughput and average turnaround time for job scheduling. Performance comparison of heuristic algorithm and analysis of task scheduling were made in [20].

Structured Allocation-based Consistent Hashing (SACH) was introduced in [23] to enhance the load balancing of cloud infrastructure. But, the memory was not minimized. An elastic pull-based Dynamic Fault Tolerant (E-DFT) scheduling mechanism was introduced in [24] to reduce the response time. However, it failed to enhance efficiency. Energy and cost optimization mechanism was developed in [25] for increasing the fault tolerance and obtaining reliability. The designed mechanism failed to reduce the response time.

Based on the literature review presented above it is recognized that task response time and the number of VM migrations at the deployment of VM. Our proposed work is different from all aforementioned VM placement schemes as we have proposed Osmotic Hybrid Vector Classifier with Quadruple Fault Tolerance level. This in turn performs migration of VM based on an Osmotic function on occurrence of fault and therefore reduced the task response time. We empowered Improved Whale Optimization Algorithm to consider the optimization of task allocation to the corresponding node in an optimal manner to minimize VM migration.

3. Methodology. In this section, the proposed method of allocation of the task (i.e. user request task) to the corresponding node (i.e. cloud server node) with appropriate fault tolerance in distribution process in the CC environment is designed. Fig. 3.1 shows the sample cloud server fault tolerance in load balancing.

As depicted in the above figure, with n tasks submitted to the CC environment, with the n node being ready for allocating the required resources, a fault tolerance mechanism is introduced in the load balancer. With this fault tolerance mechanism smooth flow between the task and node is said to be accomplished in CC environment. With the above said basic design, a method, called, Quadruple Osmotic Hybrid Classification and Whale Optimization (QOHC-WO) is proposed with the objective of improving the task response time, number of VM migrations and the percentage of fault detected rate. The WOHC-WO is split into following steps, namely, user task request, transformation of fault tolerance levels, user request task and cloud server node classification, finally optimization. Figure 3.2 given below shows the architecture of the QOHC-WO method.

Figure 3.2 demonstrates the Block diagram of Quadruple Osmotic Hybrid Classification and Whale Optimization. Initially, the user task request users submit resource requests are considered in the CC environment. Next, according to the Quadruple Fault Tolerance Level as presented above, the fault tolerance levels of user request tasks are characterized and a correlation between the task and node is established that can regulate certain fault tolerance levels of user request tasks. The third step is the appropriate classification of user request task and cloud server. In our work, the user request tasks are classified based on the dependent task (DT) and the in dependent task (IT) and cloud server nodes are classified based on the nodes either overload (OL), under load (UL) or balanced (B) respectively by means of Osmotic Hybrid Vector Classifier. According to the specification of the tasks and load in the nodes, load balancing is said to be attained at the cloud server with higher level of fault tolerance. This is performed via Osmotic function that migrates between the VM

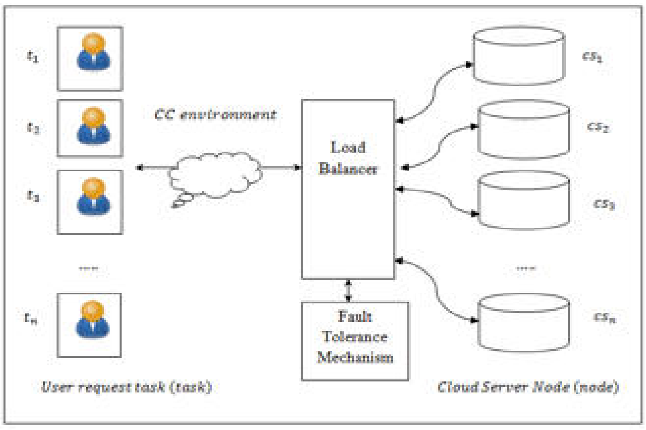


Fig. 3.1: Cloud server fault tolerance in load balancer

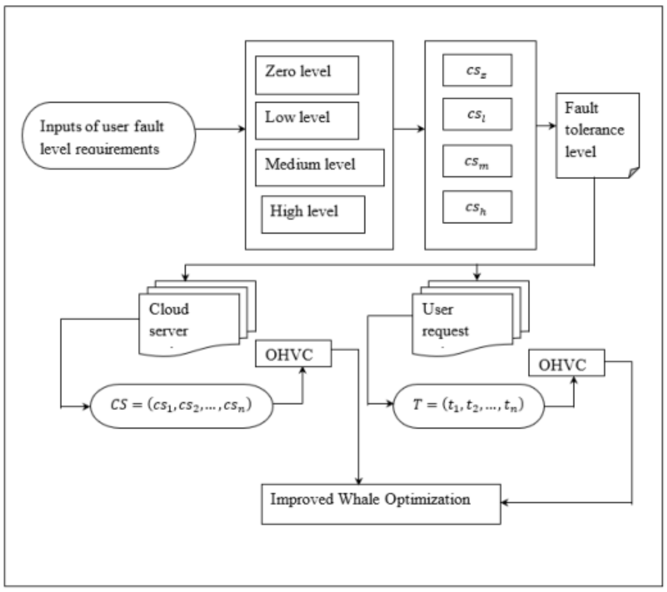

Inputs of user fault level requirements
Zero level
Low level
Medium level
High level
cs_z
cs_l
cs_m
cs_h
Fault tolerance level
Cloud server
User request
OHVC
OHVC
$CS = (cs_1, cs_2, \dots, cs_n)$
$T = (t_1, t_2, \dots, t_n)$
Improved Whale Optimization

Fig. 3.2: Block diagram of Quadruple Osmotic Hybrid Classification and Whale Optimization

machine in case of fault and therefore handling higher level of faults. Finally, optimization of task assignment to node is determined using the Improved Whale Optimization algorithm, addressing exploitation using Bandit function and exploration using Whale function and hence called improved algorithm. To start with the problem description, followed by the system model and then the elaborate description of the proposed method is presented.

According to the specification of the tasks and load in the nodes, load balancing is said to be attained at the cloud server with higher level of fault tolerance. This is performed via Osmotic function that migrates between the VM machine in case of fault and therefore handling higher level of faults. Finally, optimization of task assignment to node is determined using the Improved Whale Optimization algorithm, addressing exploitation using Bandit function and exploration using Whale function and hence called improved algorithm. To start with the problem description, followed by the system model and then the elaborate description of the proposed method is presented.

3.1. Problem description. The features of the requirements of several application programs in VMs to physical resources decide specific needs of the physical resources. Besides, the VM resources are specifically split into four types, namely, CPU, memory, I/O, and network resources. The significance of each resource differs with several applications settings of the user's. Hence, the physical machines are dispended to the VMs based on the discrete user demands that in turn utilize physical resource in an appropriate manner and enhance the user's service experience.

But, most of the current research work take into consideration only the user's individual requirements (i.e. user request task) but the question of the states of the node (i.e. cloud server nodes) and which task to be assigned to which node in an optimal manner with differences of individual demands of fault tolerance is said to be ignored. This results in the imperfection of appropriate fault tolerance in distribution process of virtual machine. Therefore, the distribution issue of virtual machines based on the states of the cloud server nodes (i.e., under load/overload/balanced) and the task dependencies (i.e. dependent task/ independent task) needs to be addressed imperatively.

3.2. System model. Let the user request task be represented as '$T = (t_1,\ t_2,\ \ldots, t_n)$' and the available cloud server nodes in the Cloud Computing environment denoted as '$CS =\ cs_1,\ cs_2,\ \ldots, cs_n$' with the maximal number of cloud server nodes represented as 'n'. Besides, the set of virtual machines utilized in the cloud server node 'cs_j' is represented as '$V_s = \{v_{j1},\ v_{j2},\ \ldots, v_{jn}\}$'.

3.3. Quadruple Fault Tolerance Degree. Fault tolerance is a main part of distributed system to guarantee the continuity and functionality of a system at a point where there's a fault or failure. In this work, the level of fault tolerance is divided in to four types. They are Zero Level Fault Tolerance, Low Level Fault Tolerance, Medium LevelFault Tolerance and High Level Fault Tolerance respectively and is represented as given below in the form of a quadruple

$$U = \{U_z,\ U_l,\ U_m,\ U_h\} \tag{3.1}$$

From the above equation (3.1), U_z relates to the zero levelfault tolerance, U_l relates to the low levelfault tolerance, U_m represents the medium level fault tolerance and U_h represents the high levelfault tolerance respectively. The user request task can select one levelin a random manner from the quadruple. However, specific fault-tolerance are not said to be operated directly. Therefore, in this work, the user's request task fault tolerance levelscorrelate to the states of the cloud server nodes (i.e. under load/overload/balanced). The fault tolerance of the cloud server nodes in this work is represented in the form of quadruple and is written as given below.

$$CS = \{cs_z,\ cs_l,\ cs_m,\ cs_h\} \tag{3.2}$$

Then the level of quadruple of the nodes 'CS' and the quadruple of the task corresponds with each other. For example, a task assigns the zero level fault tolerance as the requirement, then this degree corresponds to 'cs_z' in the nodes. In a similar manner, if a task assigns the primary level fault tolerance as the requirement, then this degree corresponds to 'cs_p' in the nodes and so on. The cases scenarios for the quadruples are given below.

Case 1: If the value of 'cs_h' is three, we need four nodes to provide service for the tasks. If any three nodes have failure and the CC environment can however furnish the service that meets the requirement of tasks, this condition correlates with high level fault tolerance of tasks.

Case 2: If the value of 'cs_m' is two, we require three nodes to provide service for the tasks. If any two nodes have downtime at the same time and the tasks can still be arrived at, this condition correlates with the medium level fault tolerance of tasks.

Case 3: If the value of 'cs_l' is one, we require two nodes to provide service for the tasks and this condition correlates with the low degree fault tolerance of tasks.

Case 4: If the value of 'cs_z' is zero, we require only one node to provide service for the tasks and this condition correlates with the zero level fault tolerance of tasks.

Therefore, we can measure both the quantity of the required nodes and the tasks. The quantity of node and the task is represented as 'NCS' and 'NT' respectively. Hence, the constraint association is represented as given below.

$$NCS \ \geq CS + 1 \tag{3.3}$$

$$NT \ \geq T + 1 \tag{3.4}$$

The level of fault tolerance submitted by the tasks is zero level fault tolerance. With the above said fault tolerance degree, the proposed method predict the status of node and task based on the response mechanism that integrates fault tolerance need and task constraints for solving the fault tolerance of VM.

3.4. Osmotic Hybrid Vector Classifier. In this section, the classification of task and the node is performed by means of Osmotic Support Vector Machine. As both the task and node are classified, this model is said to be a hybrid classifier model, called, Osmotic Hybrid Vector Classifier (OHVC). Support Vector Machine (SVM)[21] being a supervised machine learning technique classifies the given data. In this work, the classification of the tasks into dependent task and independent task and classification of nodes into under load, over load or balanced node is done.

Here, a kernel type of technique is followed that modifies the data and based on these modification, an optimal boundary between the probable outputs are said to be made. The main objective here remains in identifying the optimal separating hyperplane that maximizes the margin of the data (i.e., task and nodes). The hypothesized hyperplane equation is given below:

$$w^T x + b = 0, \ x \ \in CS, \ T \tag{3.5}$$

From the above equation (3.5), 'w' and 'x' symbolizes the vectors, with '$w^T x$' representing the dot products of the two different vectors. In this work, besides the vector 'x', includes both the cloud server nodes 'CS' and the task 'T' respectively. Let 'H_0' represent a hyperplane separating the data (i.e. cloud server node and the task) and satisfying the following equation.

$$wx + b = 0 \tag{3.6}$$

Let us select two other hyperplanes 'H_1' and 'H_2' and have the following equations

$$wx + b = 1 \tag{3.7}$$

$$wx + b = -1 \tag{3.8}$$

Then with the introduction of a constraint (y_i) always having the value of '-1', maximizes the margin, resulting in optimal hyperplane as given below.

$$Min \ in \ (w, b) \ \frac{w^2}{2} \tag{3.9}$$

$$Subject \ to \ y_i \left(wx + b\right), \ for \ i = 1, 2, .., n \tag{3.10}$$

From the result of the above value, the optimal hyperplane 'OH' is obtained. With the obtained result, Osmotic concept is used to address the issues related to migration. Osmosis refers to the uncontrolled variation of molecules from a higher to a lower water concentration. The objective behind the application of osmotic function in our work is to migrate between the virtual machines in case of the occurrence of fault. The osmotic function is represented as given below:

$$\pi = OFs = CFData_{inflow} \tag{3.11}$$

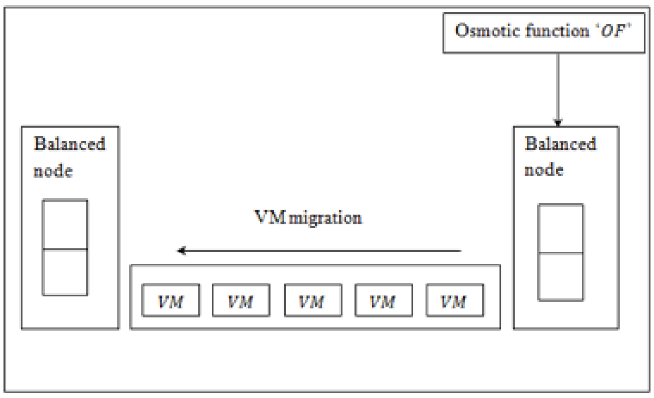


Fig. 3.3: The migration operation between VMs using Osmotic function

From the above equation (3.11), 'π' refers to the osmosis pressure law, with 'CF' representing the correction factor (i.e., migration between VMs) and '$Data_{inflow}$' refers to the inflow of data (i.e., either the task or the availability of nodes). Fig. 3.3 illustrates the migration operation between VMs using Osmotic function.

In CC environment each node (CS) has a different number of virtual machines (VMs). The set of all 'CS' in data center is '$CS = \ cs_1,\ cs_2,\ \ldots, cs_n$', where '$n$' refers to the number of 'CS' and the set of all VMs in datacenter is '$VM = VM_1,\ VM_2,\ \ldots, VM_n$' In Osmotic Hybrid Support Vector Machine, the hyperplane identified through the SVM measures the standard deviation 'σ' for each 'CS' to identify the under load 'UL', over load 'OL' and balanced 'B' nodes. This is said to be hybrid because of the classification of both node and the task performed via Osmotic SVM. This necessitates to identify the load of each 'CS' that depends on the load of 'VM' deployed into it. Then, the average load of the '$jth\ VM$' in '$ith\ CS$' (VM_{ij}) is written as follows:

$$VM_{ij} = LU_{CPU_j} + LU_{MEM_j} + LU_{BW_j} \tag{3.12}$$

From the above equation (3.12), 'LU_{CPU_j}', 'LU_{MEM_j}' and 'LU_{BW_j}' refers to the load utilization of the cpu, memory and the bandwidth of the 'jth' virtual machine respectively. The average load of '$CS_i\,(Avg_i)$' and the standard deviation for '$CS_i\,(\sigma_i)$' is written as follows:

$$AVG_i = \frac{\sum_{j=1}^{m} VM_j}{m} \tag{3.13}$$

$$CS_i\sigma_i = \sqrt{\frac{1}{n}(AvgLoad - AVG_i)^2} \tag{3.14}$$

From the above equation (3.14), 'n' refers to the number of all 'CS' and '$AvgLoad$' is the average load of all 'CS':

$$AvgLoad = \frac{1}{n}\sum_{i=1}^{n} AVGL_i \tag{3.15}$$

In a similar manner, the average task of the '$jth\ VM$' in '$ith\ CS$' (VM_{ij}) is written as follows:

$$VM_{ij} = \ Q_{IT} + Q_{DT} \tag{3.16}$$

From the above equation [16], 'Q_{IT}' and 'Q_{DT}' refers to the independent and dependent task queue of the 'jth' virtual machine respectively. In a similar manner, the standard deviation for '$T_i\,(\sigma_i)$' is written as follows:

$$T_i\sigma_i = \sqrt{\frac{1}{n}(AvgQueue - AVG_i)^2} \tag{3.17}$$

$$AvgQueue = \frac{1}{n}\sum_{i=1}^{n} AVGQ_i \tag{3.18}$$

From the above equation (3.18), 'n' refers to the number of all 'T' and '$AvgQueue$' is the average queue of all 'T'. The pseudo code representation of Osmotic Hybrid Vector Classifier is given below.

Algorithm 1: Osmotic Hybrid Vector Classification

```
Input: Task '$T = (t_1, t_2, \dots, t_n)$', Cloud Server nodes '$CS = cs_1, cs_2, \dots, cs_n$',
Output: Time-efficient hybrid classification
1: Begin
2:     Initialize fault tolerance level, load '$L$', queue '$Q$'
3:         For each task 'T' with cloud server nodes 'CS'
4:             Obtain fault tolerance level using (1)
5:             Obtain hypothesized hyperplane using (5)
6:             Select three hyperplanes using (6), (7) and (8)
7:             Obtain optimal hyperplane '$OH$' using (9) and (10)
8:             Measure osmotic function using (11)
//Node classifier
9:             Measure average load of virtual machine using (12)
10:            Measure average load and standard deviation of cloud server node using (13)
and (14)
11:            If '$\sigma_i < OH$' then '$CS_i$' is under load end if call '$OF$'
12:            If '$\sigma_i > OH$' then '$CS_i$' is over load end if call '$OF$'
13:            If '$\sigma_i = OH$' then '$CS_i$' is balanced end if
//Task classifier
14:            Measure average task of virtual machine using (16)
15:            Measure standard deviation for task using (17)
16:            If '$\sigma_i$' has influence on queue '$Q$' then '$T$' is dependent task end if call '$OF$'
17:             If '$\sigma_i$' does not has influence on queue '$Q$' then '$T$' is independent task end
if
18:        End for
19: End
```

As given in the above algorithm, to start with, the number of tasks (i.e. 500) to be assigned to the node, along with the queue, load and fault tolerance level is initialized. In this work, a quadruple fault tolerance level is used. Next, the average task and the average load in the node are analyzed. Followed by which the osmotic pressure is applied upon the occurrence of fault so that migration between the tasks and the node is said to take place, in order to achieve more cloud balanced system. Also as given in the above algorithm, the osmotic function 'OF' is applied at three different places. The first place, when overload is said to occur at the node, the second place, when under load is said to occur and the third place when the task is said to be dependent task. This is because of the reason that fault occurs during these three cases. In this way by applying osmotic function virtual machines are said to be migrated across CC environment, with minimum completion time or task response time.

3.5. Improved Whale Optimization Algorithm. Finally, a scheduling algorithm is carried out to distribute the task and the available actual resources (i.e. file) from the node and deployment rules are applied to optimize data center. In this work, an Improved Whale Optimization (IWO) Algorithm is used with the objective of improving the percentage of fault being detected by means of exploitation-exploration tradeoff. The algorithm is called as improved because a Bandit function is used to the Whale Optimization so that the problem related to choosing between options is said to be reduced and therefore improving the fault detection rate. In other words, by combining the exploitation of Bandit with the exploration of Whale, promising candidate solution is said to be arrived at. The pseudo code representation of Improved Whale Optimization Algorithm is given below.

Algorithm 2: Improved Whale Optimization Algorithm

```
Input: Coefficient vector represented by '$E$', random vectors '$rv_1$' and '$rv_2$'
Output: Optimized task-node allocation
1: Initialize current iteration '$CI$', current task position vector '$V_p$', position vector of the node
in queue '$V$'
2:     Begin
3:         For each Coefficient vector
4:             Measure prey encircling (i.e., initialization of task and node in queue) using (19)
               and (20)
5:             Evaluate positive vector using (21) and (22)
6:             Perform searching of prey (i.e., appropriate task with respect to node) using (23)
               and (24)
7:             Evaluate bandit function using (25)
8:             Return (optimal task-node allocation)
9:         End for
10: End
```

Figure 3.4 shows the flow process of Quadruple Osmotic Hybrid Classification and Whale Optimization. Initially, the number of user tasks and nodes as input is taken from the dataset. Next, the Quadruple Fault Tolerance Level is employed to find the fault tolerance level of data. Followed by, the Osmotic Hybrid Vector Classifier is utilized to measure the average load and standard deviation ‘ ’ for each ‘CS’.In addition, the node and task are classified by using this algorithm. Lastly, Improved Whale Optimization is employed to perform searching of prey depending on the bandit function for finding the optimal task-node allocation.

Whale optimization algorithm (WOA) is a new biological Meta-heuristic algorithm. The significant concept of meta-heuristic algorithms such as particle swarm optimization (PSO), genetic algorithm (GA) is used to achieve optimal solutions. But, it failed to find the local optimal solution with slow convergence speed and low convergence accuracy. In order to overcome the issue, the advantage of the Whale Optimization algorithm is easy to implement the process and robust. Whale Optimization is a population-based algorithm that simulates the social behavior of humpback whales. The population-based algorithm is used for avoiding the local optima and getting a globally optimal solution to handle the dissimilar constrained or unconstrained optimization issues. Whale Optimization is a population based algorithm that simulates the social behavior of humpback whales. The Whale Optimization in this work is used to optimally allocate the task to the corresponding node. This is performed by applying two rules to enhance the position of candidate solutions in each step that are encircling prey, search for prey and optimal prey allocation. They are described as follows by the following equations starting with prey encircling (i.e. initialization of task and node in queue).

$$D = [F.V_p\,(CI) - V\,(CI)] \tag{3.19}$$

$$V\,(CI+1) = V_p\,(CI) - E.D \tag{3.20}$$

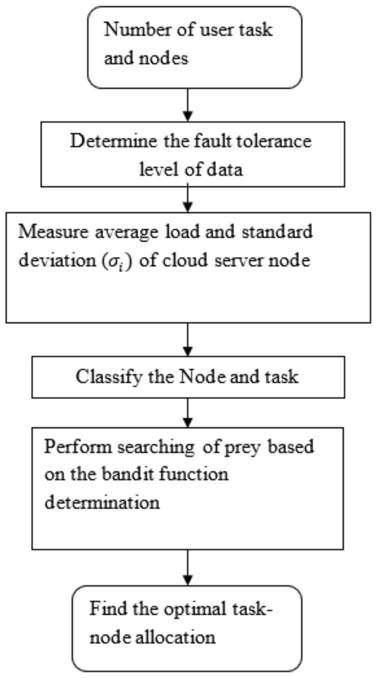

Number of user task and nodes
Determine the fault tolerance level of data
Measure average load and standard deviation (σ_i) of cloud server node
Classify the Node and task
Perform searching of prey based on the bandit function determination
Find the optimal task-node allocation

Fig. 3.4: Flow Process of Quadruple Osmotic Hybrid Classification and Whale Optimization

From the above equations (3.19) and (3.20) 'CI' refer to the current iteration with coefficient vector represented by 'E', 'F',current task position vector being 'V_p' and 'V' referring to the position vector of the node in queue 'Q'. The vectors 'E' and 'F' are measured as given below.

$$E = 2e.rv_1 - e \tag{3.21}$$

$$F = 2.rv_2 \tag{3.22}$$

From the above equations (3.21) and (3.22), 'rv_1' and 'rv_2' refers to the random vectors with elements 'e' decreased from '2' to '0' over certain iteration period. Next searching for prey (i.e. appropriate task with respect to node) is written as follows.

$$D = (f * t_{rand} - t) \tag{3.23}$$

$$V\,(CI + 1) = t_{rand} - E.D \tag{3.24}$$

From the above equations (3.23) and (3.24), 't_{rand}' refers to the random task position vector, therefore converging towards exploration. Finally optimal prey allocation with the solution towards exploitation is performed by applying the Bandit function. Here, with the utilization of Bandit function, exploitation is identified as a set of distributions '$CS = \{cs_1,\ cs_2,\ \ldots, cs_n\}$', each distribution being linked with the rewards delivered by one of the levers. Let '$\mu = \{\mu_1,\ \mu_2,\ \ldots, \mu_n\}$' represents the mean values linked with these reward distributions corresponding to each node. The node iteratively plays one lever per round and looks at the corresponding reward. The objective here remains in maximizing the sum of the collected rewards by the node. In a similar manner, the regret after rounds is said to be the anticipated difference between the reward sum associated with an optimal strategy and the sum of the collected rewards. It is mathematically expressed as given below:

$$\rho = T\mu^2 - \sum_{CI=1}^{T} R_{CI}\,[CS] \tag{3.25}$$

From the above equation (3.25), the cloud server nodes 'CS' reward 'R' in round 'CI' is analyzed with the maximal reward mean being 'μ^2', ensuring fair exploitation of the cloud server node. In this manner, a trade-off between exploitation-exploration is said to be obtained resulting in maximal percentage of fault detected.

4. Experimental setup. In this section, the performance evaluation of the proposed method, Quadruple Osmotic Hybrid Classification and Whale Optimization (QOHC-WO), and three other existing methods, Crystal [1], energy-efficient fault-tolerant replica management policy [2], and Proactive fault tolerance framework (PFTF) algorithm [6] is implemented with CloudSim simulation using JAVA platform. Experiments are conducted with the aid of Personal Cloud datasets via Active Personal Cloud Measurement extracted from http://cloudspaces.eu/results/datasets. The number of 17 attributes such as row id, account id, file size (i.e. task size), operation_time_start, operation_time_end, time zone, operation_id, operation type, bandwidth trace, node_ip, node_name, quoto_start, quoto_end, quoto_total (storage capacity), capped, failed and failure info. These are utilized for performing efficient task scheduling to the multiple virtual machines in the cloud. The implementation is conducted with the hardware specification of Windows 10 Operating system, core i3-4130 3.40GHZ Processor, 4GB RAM, 1TB (1000 GB) Hard disk, ASUSTek P5G41C-M Motherboard, Internet Protocol. For accomplishing the experimental evaluation, the QOHC-WO considers a number of task in the range of 50-500 from the Personal Cloud datasets. The effectiveness of the method is evaluated in terms of task response time, the number of VM migrations, and the percentage of fault detected rate. The results of four different techniques are discussed with the aid of tables and graphical representation.

4.1. Performance evaluation of task response time. The response time for a task is the time between when it becomes active (e.g. an external event or occurrence of fault) and the time it completes. Several factors can cause the response time of a task to be longer than its execution time. In our work, the response time of a task is evaluated based on the standard deviation of server and the task respectively. It is measured as given below:

$$TRes_{time} = \sum_{i=1}^{n} T_i * Time\ \ [CS_i\sigma_i +\ T_i\sigma_i] \tag{4.1}$$

From the above equation (4.1), the task response time '$TRes_{time}$' is measured based on the number of tasks 'T_i', and the time consumed in obtaining the average node '$CS_i\sigma_i$' and obtaining the average task '$T_i\sigma_i$' respectively. It is measured in terms of milliseconds (ms). The sample calculation of task response time using the proposed QOHC-WO, existing Crystal [1] and energy-efficient fault-tolerant replica management policy [2] and PFTF algorithm [6]. is given below.

Sample calculations for task response time:

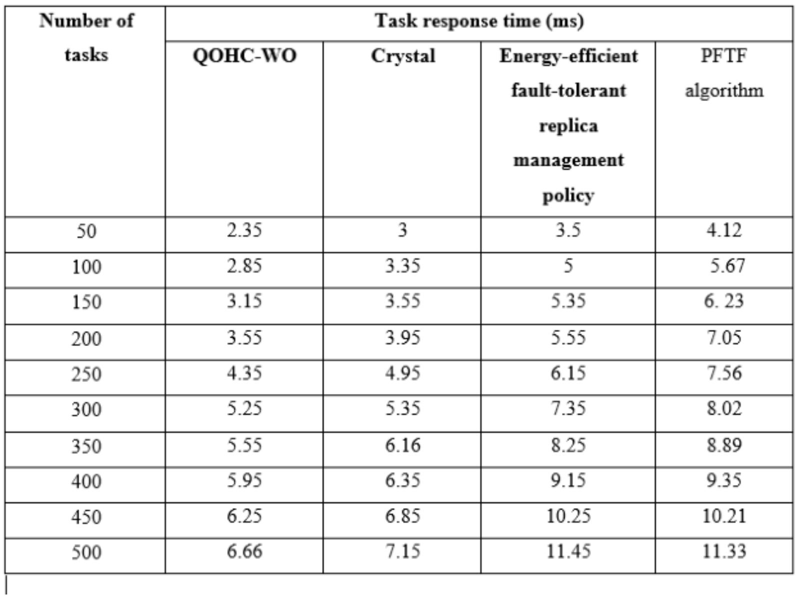
Proposed QOHC-WO: With '50' numbers of tasks considered for experimentation, '0.035ms' time consumed in calculating average nodes and '0.012ms' time consumed in calculating average tasks, the overall task response time is measured as given below.

$$TRes_{time} = 50 * [0.035ms + 0.012ms] = 2.35ms$$

Existing Crystal [1]: With '50' numbers of tasks considered for experimentation, '0.043ms' time consumed in calculating average nodes and '0.017ms' time consumed in calculating average tasks, the overall task response time is measured as given below.

$$TRes_{time} = 50 * [0.043ms + 0.017ms] =\ 3ms$$

Table 4.1: Comparison of task response time for different tasks



Number of tasks	Task response time (ms)			
	QOHC-WO	Crystal	Energy-efficient fault-tolerant replica management policy	PFTF algorithm
50	2.35	3	3.5	4.12
100	2.85	3.35	5	5.67
150	3.15	3.55	5.35	6. 23
200	3.55	3.95	5.55	7.05
250	4.35	4.95	6.15	7.56
300	5.25	5.35	7.35	8.02
350	5.55	6.16	8.25	8.89
400	5.95	6.35	9.15	9.35
450	6.25	6.85	10.25	10.21
500	6.66	7.15	11.45	11.33

Existing Energy-efficient fault-tolerant replica management policy [2]: With '50' numbers of tasks considered for experimentation, '0.048ms' time consumed in calculating average nodes and '0.022ms' time consumed in calculating average tasks, the overall task response time is measured as given below.

$$TRes_{time} = 50 * [0.048ms + 0.022ms] = 3.5ms$$

Existing Proactive fault tolerance framework (PFTF) algorithm [6]: With '50' numbers of tasks considered for experimentation, '0.0412ms' time consumed in calculating average nodes and '0.0412ms' time consumed in calculating average tasks, the overall task response time is measured as given below.

$$TRes_{time} = 50 * [0.0412ms + 0.0412ms] = 4.12ms$$

Table 4.1 summarizes the comparison of task response time for 500 different tasks presented in the nodes in CC environment. Task response time of QOHC-WO is compared with Crystal [1] and Energy-efficient fault-tolerant replica management policy [2] in table 1. In order to perform a fair comparison similar tasks were provided for three different methods.

Fig. 4.1 given above illustrates the task response time. The x axis in the above figure denotes the tasks considered for experimentation in the range of 50 to 500. The y axis in the above figure denotes the corresponding task response time observed using the proposed QOHC-WO and the existing [1] and [2] methods. The task here considered obtain from the active personal cloud measurement, varies with different files of different sizes.

Figure 4.1 given above illustrates the task response time. The x axis in the above figure denotes the tasks considered for experimentation in the range of 50 to 500. The y axis in the above figure denotes the corresponding task response time observed using the proposed QOHC-WO and the existing [1], [2] and [6] methods. The task here considered obtain from the active personal cloud measurement, varies with different files of different sizes. Hence, the task response time is not linearly increased. Besides, from the simulation conducted using CloudSim, for 50 numbers of tasks, the task response time using the proposed method was found to be 2.35 ms, 3 ms using [1], 3.5 ms using [2] and 4.12ms using [6]. From this it is inferred that the task response time is said to be comparatively lesser using the QOHC-WO method when compared to [1], [2]

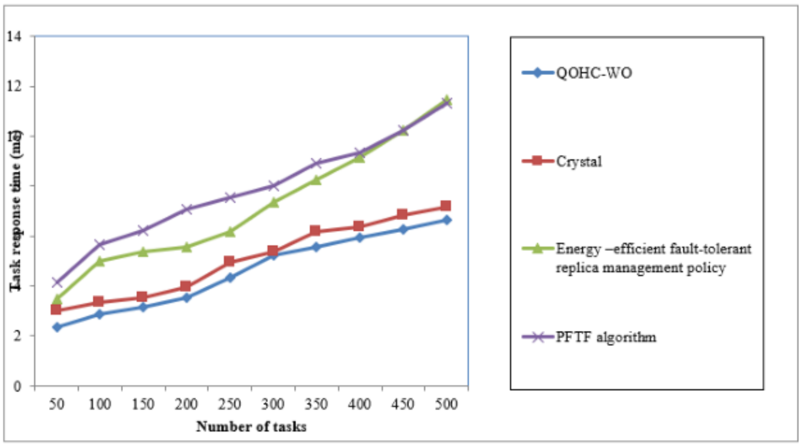


Fig. 4.1: Task response time

and [6]. This is because of the application of Osmotic Hybrid Vector Classification algorithm. By applying this algorithm, two main things are said to happen. First, the classification of node is node and then the classification of task is performed separately using the using the hybrid vector classification algorithm. With this, imperative classification is made via optimal hyperplane identification. Next, with the application of Osmotic function upon the occurrence of fault, efficient migration between the virtual machines is said to take place. Therefore, the task response time using the QOHC-WO method is reduced by 10% compared to [1], 36% compared to [2] and 42% compared to [6] respectively.

4.2. Performance evaluation of task migration. The task migration is obtained using the net migration rate [22]. The net migration rate refers to the difference between the number of immigrants (i.e., task entering into the node for request) and the number of emigrants (i.e., task leaving the node in search of another node) throughout the year (i.e. for an average task size).

$$T_{mig} = \frac{(T \rightarrow CS - T \leftarrow CS)}{T} * 1000 \tag{4.2}$$

From the above equation (4.2), the task migration 'T_{mig}' is obtained based on the difference between the task entering into the node for request '$T \rightarrow CS$' and the task leaving the node '$T \leftarrow CS$' to the overall task considered for experimentation 'T'. The sample calculation for virtual machine migration using the three methods, QOHC-WO, existing Crystal [1] and energy-efficient fault-tolerant replica management policy [2] is given below.

Sample calculation for VM migration

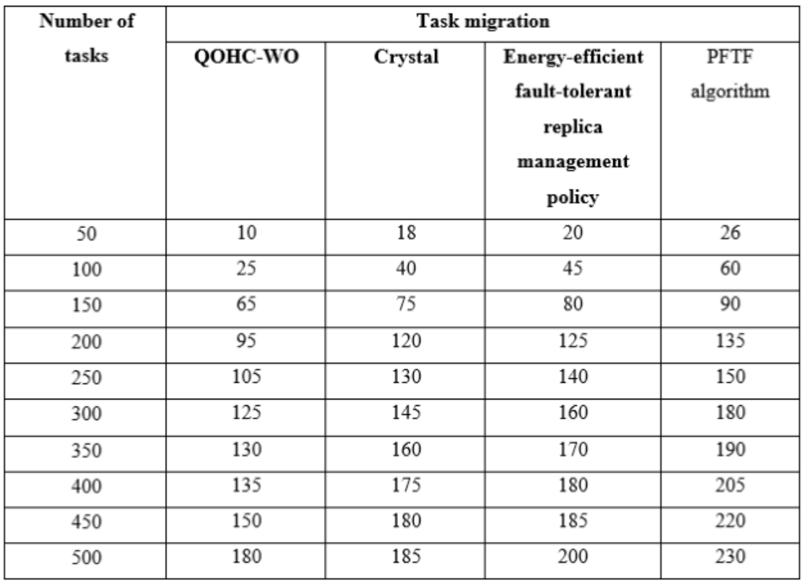
Proposed QOHC-WO: With '50' numbers of tasks considered for experimentation, '7' numbers of task entering the node and '4' numbers of task leaving the node, the overall task migration rate is given below.

$$T_{mig} = \frac{3+2}{50} * 100 = 10$$

Existing Crystal [1]: With '50' numbers of tasks considered for experimentation, '9' numbers of task entering the node and '7' numbers of task leaving the node, the overall task migration rate is given below.

$$T_{mig} = \frac{5+4}{50} * 100 = 18$$

Table 4.2: Comparison of task migration for different tasks



Number of tasks	Task migration			
	QOHC-WO	Crystal	Energy-efficient fault-tolerant replica management policy	PFTF algorithm
50	10	18	20	26
100	25	40	45	60
150	65	75	80	90
200	95	120	125	135
250	105	130	140	150
300	125	145	160	180
350	130	160	170	190
400	135	175	180	205
450	150	180	185	220
500	180	185	200	230

Existing energy-efficient fault-tolerant replica management policy [2]: With '50' numbers of tasks considered for experimentation, '9' numbers of task entering the node and '7' numbers of task leaving the node, the overall task migration rate is given below.

$$T_{mig} = \frac{6+4}{50} * 100 = 20$$

Existing PFTF algorithm [6]: With '50' numbers of tasks considered for experimentation, '8' numbers of task entering the node and '5' numbers of task leaving the node, the overall task migration rate is given below.

$$T_{mig} = \frac{8+5}{50} * 100 = 26$$

Table 4.2 summarizes the comparison of task migration rate for 500 different tasks in CC environment. Task migration of QOHC-WO is compared with Crystal [1] and Energy-efficient fault-tolerant replica management policy [2] and PFTF algorithm [6] in table 4.2.

Fig. 4.2 given above illustrates the task migration for 500 different numbers of tasks. As the upload and download operations alternatively changes since the provider or the node only needed to handle per account at a time, the task migration is not proportionately same. Besides, from the figure it is also inferred, that with the increase in the number of tasks, also the migration rate increases.

Figure 4.2 given above illustrates the task migration for 500 different numbers of tasks. As the upload and download operations alternatively changes since the provider or the node only needed to handle per account at a time, the task migration is not proportionately same. Besides, from the figure it is also inferred, that with the increase in the number of tasks, also the migration rate increases. From the sample calculation, it is observed that with '50' numbers of task, the task migration rate using QOHC-WO method was observed to be '10', '18' using [1] and '20' using [2] respectively. From this analysis it is inferred that the task migration rate is lesser using QOHC-WO method when compared to [1], [2] and [6]. Followed by, various performance results are observed for each method. For each method, ten different results are observed. The performance of the proposed QOHC-WO method is compared to other existing methods. The task migrations are minimal in

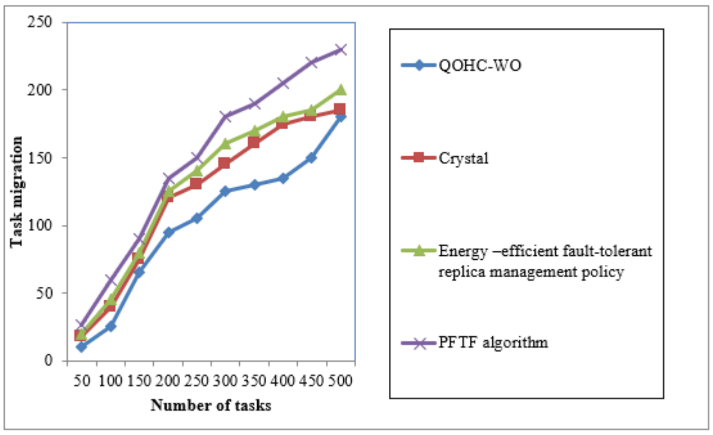


Fig. 4.2: Task migration

the QOHC-WO method due to extensive Osmotic Hybrid Vector Classifier algorithm in identifying the most appropriate VM to each of the tasks which is illustrated in Figure 5. So, the task request was said to be completed in the shortest time. But in the case of [1] and [2] algorithms with vertex configuration pattern [1] and considering the deadline and budget constraints [2] has not taken the task lengths. Instead, it considers only the resource capabilities of the node. Therefore, the task migration is said to be reduced using QOHC-WO method. Finally, the average of ten comparison results designates that the task migration of the proposed method is minimized by 21% when compared to [1], 26% when compared to [2], and 35% compared to [6].

4.3. Performance of fault occurrence rate. Finally, the fault occurrence rate is measured. The fault occurrence rate refers to the occurrences of fault during load balancing in CC environment. Lower the fault occurrence more efficient the method is said to be and vice versa. The fault occurrence rate is expressed as given below.

$$FOR = \frac{F_{occur}}{T} * 100 \tag{4.3}$$

From the above equation (4.3), the fault occurrence rate 'FOR' is measured based on the number of tasks provided to the node 'T' and the fault occurred 'F_{occur}' during that particular session. It is expressed in terms of percentage (%). The sample calculations for fault occurrence rate using the three methods, QOHC-WO, existing Crystal [1] and energy-efficient fault-tolerant replica management policy [2]and PFTF algorithm [6] is given below.

4.4. Sample calculation for fault occurrence rate.

Proposed QOHC-WO: With '50' numbers of tasks considered for experimentation and '4' number of faults occurred the overall fault occurrence rate is measured as given below.

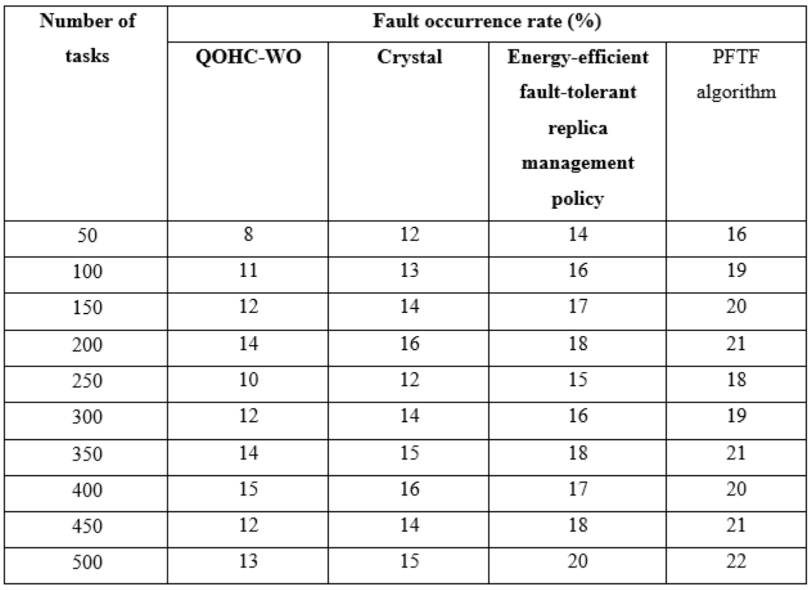
$$FOR = \frac{4}{50} * 100 = 8\%$$

Existing Crystal [1]: With '50' numbers of tasks considered for experimentation and '6' number of faults occurred the overall fault occurrence rate is measured as given below.

$$FOR = \frac{6}{50} * 100 = 12\%$$

Existing Energy-efficient fault-tolerant replica management policy [2]: With '50' numbers of tasks considered for experimentation and '7' number of faults occurred the overall fault occurrence rate is

Table 4.3: Comparison of fault occurrence rate for different tasks



Number of tasks	Fault occurrence rate (%)			
	QOHC-WO	Crystal	Energy-efficient fault-tolerant replica management policy	PFTF algorithm
50	8	12	14	16
100	11	13	16	19
150	12	14	17	20
200	14	16	18	21
250	10	12	15	18
300	12	14	16	19
350	14	15	18	21
400	15	16	17	20
450	12	14	18	21
500	13	15	20	22

measured as given below.

$$FOR = \frac{7}{50} * 100 = 14\%$$

Existing PFTF algorithm [6]: With '50' numbers of tasks considered for experimentation and '8' number of faults occurred the overall fault occurrence rate is measured as given below.

$$FOR = \frac{8}{50} * 100 = 16\%$$

Table 4.3 summarizes the comparison of fault occurrence rate for 500 different tasks. Fault occurrence rate of QOHC-WO is compared with Crystal [1] and Energy-efficient fault-tolerant replica management policy [2] in table 4.3.

Figure 4.3 given above shows the fault occurrence rate. From the figure, it is illustrative that the fault occurrence rate is neither in the increasing trend nor in the decreasing trend. As shown in the graphical chart, there are four various colors of lines such as blue, brown, green and violet indicates the fault occurrence rate of four techniques namely QOHC-WO, Crystal [1] , energy-efficient fault-tolerant replica management policy [2] and PFTF algorithm [6] respectively. Among the four methods, the proposed QOHC-WO has the ability for reducing the fault occurrence rate. This is because of the reason that each task requested with the node differs in terms of resources (i.e. file size). Besides, for experimentation, the average of similar tasks is used for each simulation run and hence fair comparison is said to be ensured for all the three different methods. The fault occurrence rate is found to be reduced using the QOHC-WO method when compared to [1], [2] and [6]. This is because of the application of Improved Whale Optimization Algorithm. By applying this algorithm, first while choosing between options (i.e. between nodes for the corresponding task), an exploitation-exploration tradeoff is said to occur. To have a balanced between this tradeoff, with Bandit exploitation and Whale exploration, the VM migration is said to be minimized and hence the fault occurrence rate is also said to be reduced. By applying the QOHC-WO method, the fault occurrence rate was reduced by 15% when compared to [1], 29% when compared to [2] and 39% when compared to [6] respectively.

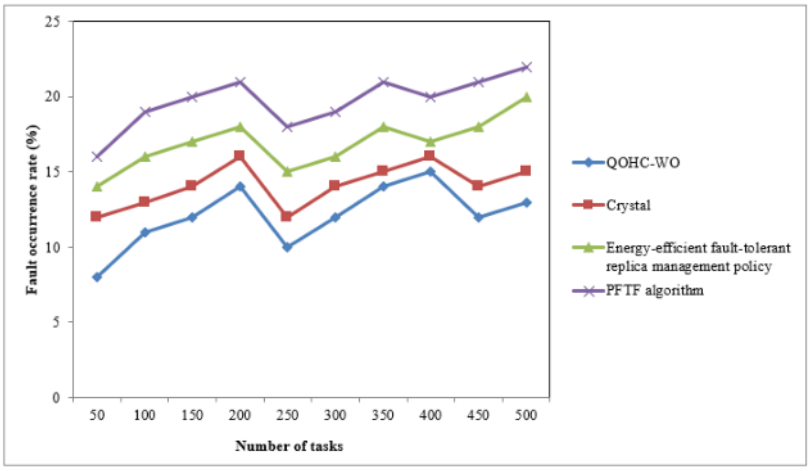


Fig. 4.3: Fault occurrence rate

The fault occurrence rate is found to be reduced using the QOHC-WO method when compared to [1] and [2]. This is because of the application of Improved Whale Optimization Algorithm. By applying this algorithm, first while choosing between options (i.e. between nodes for the corresponding task), an exploitation-exploration tradeoff is said to occur. To have a balanced between this tradeoff, with Bandit exploitation and Whale exploration, the VM migration is said to be minimized and hence the fault occurrence rate is also said to be reduced. By applying the QOHC-WO method, the fault occurrence rate was reduced by 15% when compared to [1] and 29% when compared to [2] respectively.

5. Conclusion. Several optimization methods have been available in the literature whose performances rely on various aspects. In this paper, the proposed Quadruple Osmotic Hybrid Classification and Whale Optimization (QOHC-WO) method is designed with classification and optimization to handle the three different scenarios in the CC environment. The osmotic hybrid vector classifier is used to consider the migration of VM in case of occurrences of faults with lesser task response time and the number of task migration. Depending on the classifier outcomes, the quadruple fault tolerance level, and osmotic hybrid vector classifier is employed that distributes the tasks with the nodes. Finally, Improved Whale Optimization is applied to handle the tradeoff between exploitation and exploration with aid of Bandit function and Whale optimizer. This helps to minimize the fault occurrence rate. The comprehensive experimental evaluation is conducted and the obtained result indicates that the proposed QOHC-WO method provides improved performance than the existing classification and optimization approaches, having a higher fault detection rate, and lesser job response time, as well as the number of task migration.

REFERENCES

[1] Nasirian S, Farhad F, *Crystal: A scalable and fault-tolerant Archimedean-based server-centric cloud data center network architecture*, Computer Communications, Elsevier, Sep 2019 [Crystal]

[2] Li L, Ping Wang Y, Chen Y, Luo Y, *Energy-efficient fault-tolerant replica management policy with deadline and budget constraints in edge-cloud environment*, Journal of Network and Computer Applications, Elsevier, Oct 2019 [energy-efficient fault-tolerant replica management policy]

[3] Chinnaiah M R, Nalini N,*Fault tolerant software systems using software configurations for cloud computing*, Journal of Cloud Computing: Advances, Systems and Applications, SpringerOpen, Jul 2018

[4] Sivagami V M, Easwarakumar K S, *An improved dynamic fault tolerant management algorithm during VM migration in Cloud Data Center*, Future Generation Computer Systems, Elsevier, Nov 2018

[5] Qin Y, Yang W, Xiao Ai, Chen L,*Fault tolerant storage and data access optimization in data center networks*, Journal of Network and Computer Applications, Elsevier, Apr 2018

[6] Tamilvizhi T, Parvathavarthini B, *A novel method for adaptive fault tolerance during load balancing in cloud computing*, Cluster Computing, Springer, Jul 2017

[7] Wu H, Jin Q, Zhang C, Guo H, *A Selective Mirrored Task Based Fault Tolerance Mechanism for Big Data Application Using Cloud*, Wireless Communications and Mobile Computing, Wiley, Feb 2019

[8] Liu S, Jia W, Pan W,*Fault-tolerant feedback virtual machine deployment based on user-personalized requirements*, Frontiers of Computer Science, Springer, Jul 2017

[9] Kumar S, Rana D S, Dimri S C,*Fault Tolerance and Load Balancing algorithm in Cloud Computing: A survey*, International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 7, July 2015

[10] Milan S T, Rajabion L, Ranjbar H, Navimipour N J*, Nature inspired meta-heuristic algorithms for solving the load-balancing problem in cloud environments*, Computers and Operations Research, Elsevier, May 2019

[11] Kapil N. Vhatkar, Girish P. Bhole,*Optimal container resource allocation in cloud architecture: A new hybrid model*, Journal of King Saud University –Computer and Information Sciences, Elsevier, Oct 2019

[12] Tang F, Laurence T. Y, Tang C, Li J, Guo M,*A Dynamical and Load-Balanced FlowScheduling Approach for Big Data Centers inClouds*, IEEE Transaction on Cloud Computing, IEEE Transactions on Cloud Computing Volume: 6, Issue: 4, Oct.-Dec. 1 2018

[13] Cui J, Lu Q, Zhong H, Tian M, and Liu L, *A Load-balancing Mechanism for Distributed SDN Control Plane Using Response Time*, IEEE Transactions on Cloud Computing Volume: 6, Issue: 4, Oct.-Dec. 1 2018

[14] Sotiriadis S, Bessis N, Amza C, Buyya R,*Vertical and horizontal elasticity for dynamic virtual machine reconfiguration*,IEEE Transactions on Services Computing Volume: 12, Issue: 2, March-April 1 2019

[15] Guo L, Hu G, Dong Y, Luo Y, Zhu Y,*A Game Based Consolidation Method of Virtual Machines in Cloud Data Centers With Energy and Load Constraints*, IEEE Access, Dec 2017

[16] Liaqat M, Naveed A, Alim R L, Shuja J,Man Ko K,*Characterizing Dynamic Load Balancing in Cloud Environments Using Virtual Machine Deployment Models*, IEEE Access, Sep 2019

[17] Li C, Tang J, Ma T, Yang X, Luo Y,*Load balance based workflow job scheduling algorithm in distributed cloud*, Journal of Network and Computer Applications, Elsevier, Dec 2019

[18] Abdulhamid S M, Shafie M Latiff A,Abdul-Salaam G, Hussain Madni S M, *Secure Scientific Applications Scheduling Technique for Cloud Computing Environment Using Global League Championship Algorithm*, PLOS ONE |, 2016.

[19] Idris H, Ezugwu A E, Sahalu B , Junaidu, Aderemi O. A, *An improved ant colony optimization algorithm with fault tolerance for job scheduling in grid computing systems*, PLOS ONE, https://doi.org/10.1371/journal.pone.0177567 May 17, 2017.

[20] Hussain Madni S H, Abd Latiff M S, Abdullahi M,Abdulhamid S M, Usman M J,*Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment*, PLOS ONE | https://doi.org/10.1371/journal.pone.0176321 May 3, 2017.

[21] Nalepa J, Kawulok M, *Selecting training sets for support vector machines: a review*, Artificial Intelligence Review, Springer, Jan 2018.

[22] Net Migration Rate: Definition, Formula & Statistics - Video & Lesson Transcript |Study.com. *Study.com.* Retrieved 2017-03-29.

[23] Yuichi Nakatani, "Structured Allocation-Based Consistent Hashing With Improved Balancing for Cloud Infrastructure", IEEE Transactions on Parallel and Distributed Systems (Volume: 32, Issue: 9, Sept. 1 2021) DOI: 10.1109/TPDS.2021.3058963.

[24] PushpanjaliGupta, Prasan KumarSahoo, Bharadwaj Veeravalli, "Dynamic fault tolerant scheduling with response time minimization for multiple failures in cloud", Journal of Parallel and Distributed Computing, Elsevier, Volume 158, December 2021, Pages 80-93 https://doi.org/10.1016/j.jpdc.2021.07.019.

[25] Sudha Danthuluri , Sanjay Chitnis, "Energy and cost optimization mechanism for workflow scheduling in the cloud", Materials Today: Proceedings, Elsevier,2021. https://doi.org/10.1016/j.matpr.2021.07.168.

Edited by: Vinoth Kumar
Received: Jul 27, 2022
Accepted: Oct 26, 2022