Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research paper

An Energy Efficient Fault Tolerance Technique Based on Load Balancing Algorithm for High-Performance Computing in Cloud Computing

H. Jahanpour, H. Barati*, A. Mehranzadeh

Department of Computer Engineering, Dezful Branch, Islamic Azad University, Dezful, Iran.

Article Info

Article History:
Received 12 November 2019
Reviewed 27 December 2019
Revised 02 February 2020
Accepted 04 May 2020

Keywords:
Cloud computing
Fault tolerance
High-Performance computing
Virtual machines
Load balancing

*Corresponding Author's Email Address:
hbarati@iaud.ac.ir

Abstract

Background and Objectives: Cloud Computing has brought a new dimension to the IT world. The technology of cloud computing allows employing a large number of Virtual Machines to run intensive applications. Each failure in running applications fails system operations. To solve the problem, it is required to restart the systems.
Methods: In this paper, to predict and avoid failure in HPC systems, a method of fault tolerance to High-Performance Computing systems (HPC) in the cloud is called Daemon-COA-MMT (DCM), has been proposed. In the proposed method, the Daemon Fault Tolerance technique has been enhanced, and COA-MMT has been utilized for load balancing. The method consists of four modules, which are used to determine the host state. When the system is in the alarm state, the current host may face failure. Then the most optimal host for migration is selected, and process-level migration is performed. The method causes decreased migration overheads, decreased system performance speed, optimal use of underutilized hosts instead of leasing new hosts, appropriate load balancing, equal use of hardware resources by all hosts, focusing on QoS and SLA, and the significant decrease of energy consumption.
Results: The simulation results revealed that in terms of parameters, the proposed method declines average job makespan, average response time, and average task execution cost by 18.06%, 35.68%, and 24.6%, respectively. The proposed fault tolerance algorithm has improved energy consumption by 30% and decreased the HPC systems' failure rate.
Conclusion: In this study, the Daemon Fault Tolerance technique has been enhanced, and COA-MMT has been utilized for load balancing in high performance computing in the cloud computing.

Introduction

Cloud computing is the greatest revolution in the computing world, so that significant organizations and companies have changed their traditional data processing systems to cloud service to store a large amount of data [1]. Cloud computing advantages are running computation-intensive applications, decreased time of applying the hardware, and cost [2]. It reduces the time of applying the hardware and cost. There are two critical roles in cloud computing: cloud service providers and users [3]. The providers such as Amazon and Bare Metal Cloud offer Virtual Machines (VMs), the hardware, etc. to their clients in return for the

subscription. Based on the services provided by them, clouds are divided into four categories: Infrastructure as a Service (IaaS), Software as a Service (SaaS), Platform as a Service (PaaS), and Hardware as a Service (HaaS) [4, 5]. HaaS focuses on the hardware. The service can be leased for research, massive information, and configuration of HPC systems [4]. HPC is a branch of software science that causes great scientific and computing jobs so rapidly and less costly by integrating the computing power of many small and medium computers [6]. HPC systems can process a large volume of data and analyze the results so rapidly. Using HaaS, running conventional computation-intensive applications on HPC systems in the cloud will be possible [7]. Despite different advantages such as fastness, resource provisioning, cost reduction, multitenant services, etc., cloud computing faces various challenges, including load balancing, security, reliability, possession, green technology, backing up data, and transferring data [8]. Some of the most critical cloud computing challenges are reliability and resource availability, especially at the HaaS level [9].

A system will be called fault tolerance if it fulfills its determined duties properly, even in the presence of software and hardware failures [10]. In fault-tolerance systems, the system's restarting is refrained to decrease operational costs and energy consumption [11]. The importance of fault tolerance is to develop the availability of resources, reliability of cloud services, and running applications. To minimize the effects of a failure on the system and provide accurate and successful running of applications, failures should be predicted and managed [12]. If fault tolerance is not provided, the system will incur irreparable damage [13]. Therefore, fault tolerance is an essential feature of cloud computing systems, especially HPC systems, since it results in shorter running times in the presence of failure. Also, load balancing is one of the main challenges of cloud computing, which divides workload evenly between hosts to satisfy users and increase the rate of resource consumption [14]. Load balancing aims to minimize energy consumption and reduce carbon dioxide emissions in cloud computing [15]. Decreased energy consumption in cloud computing systems leads to less carbon dioxide in cloud infrastructures, which causes less warming and pollution of the environment. Less energy consumption and carbon dioxide emission are essential criteria for energy-efficient load balancing in cloud computing, which causes green computing [16]. To provide energy-efficient fault tolerance, increase Quality of Service (QoS), cause effective use of resources, lessen violation of Service Level Agreement (SLA), reduce response time, and accurately examine the system's state. Then the failure is predicted and refrained through effective use of resources. This method causes energy-efficient fault tolerance and proper load balancing among hosts. In the proposed method, the Daemon Fault Tolerance technique has been enhanced, and COA-MMT has been utilized for load balancing. The proposed method consists of four modules: node monitoring with Lm sensors module, rule-based predictor module, migration policy module based on COA-MMT, and controller module of DCM. The method causes decreased migration overheads, decreased system performance speed, optimal use of underutilized hosts instead of leasing new hosts, appropriate load balancing, equal use of hardware resources by all hosts, focusing on QoS and SLA, and a significant decrease of energy consumption. The paper is organized as follows: Next Section includes related work in fault tolerance and load balancing. In Next Section, the proposed method, DCM, and its modules for fault tolerance in HPC systems are discussed in detail. Next Section consists of the simulation and evaluation of the method. Finally, Section presents the conclusions.

Related Work

This part investigates specific algorithms in load balancing and fault tolerance in cloud computing and individually represent their advantages and disadvantages.

Pan et al. [17] represented Particle Swarm Optimization (PSO) algorithm, an evolutionary computing method, originated from particles' social and natural behavior. Particles possess state and speed and move in a multidimensional search space. Each particle determines its speed based on its own best state and the state of the best particle in the society, which reduces response time [18-19].

Huang et al. [20] suggested the Genetic Algorithm. In this algorithm, the gene cost is first calculated through the current scheduling solution's ratio to the best scheduling solution. Then based on the gene cost, a scheduling strategy is decided. Finally, the least costly solution, which is similar to the final scheduling solution, is selected. The standard genetic algorithm guarantees the load balancing of the system more effectively compared with other methods. The rate of loading fluctuation of VMs plays an essential role in load balancing.

Abdullah et al. [21] suggested that Bat Algorithm provides load balancing. In this method, first, each bat receives a primary value. The speed and location of each bat are randomly determined in a d-dimensional space. Each bat's fitness function is calculated, and the best state for the bat is determined based on the least value of the function. This method provides optimal utilization of all resources and is more efficient. Its convergence speed is superior to that of PSO. However, an increased number of requests causes a longer response time. Also,

users wait a long time to receive service. The result will be decreased QoS and users' satisfaction and increased violation of SLA and system costs.

Ghafari et al. [22] represented the Artificial Bee colony Algorithm-Minimal Migration Time (Bee-MMT) to provide load balancing. In this method, first, the over-utilized host is determined. Then, one or more VMs are determined to migrate to underutilized hosts. Then VMs migrate from over-utilized hosts to new hosts. On migrating, the previous host switches to sleep mode. In this algorithm, the violation of SLA is used as an essential metric to satisfy QoS. The method provides better response time than conventional methods and reduces energy consumption in cloud computing infrastructure. Also, BEE-MMT causes decreased carbon dioxide and the appropriate efficiency of the resources of the system.

In [23], Daemon's fault tolerance is proposed. This algorithm is based on the methods of predicting failures. This algorithm has four modules with specific duties such as node monitoring module with Linux monitoring sensors, rule-based fault predictor module, migration policy module, and controller module. The first module is the node monitoring module Lm sensors. The monitoring node is an open-source using Lm sensors to monitor the accuracy of the computer's tasks. Modern CPUs are made of sensors used for monitoring CPU temperature, fan speed, memories, number of user's requests, and other parameters. Rule-based prediction is the second stage. At this stage, the failure is predicted based on the history of failures and the system's maximum workload. The predictor module inputs consist of four parameters, to which specific weight values are assigned to calculate the state of the operating system. In the third stage, the migration policy is implemented. The purpose of the policy is to execute computation-intensive entirely with minimum energy consumption. At the fourth stage, the controller module is implemented. As failure is predicted, FTDaemon calls for the module. Occurring failure requires the system to lease an additional node. On migrating from an unhealthy node to a newly leased node, the unhealthy node is abandoned. When the host is not operating at a critical state, there is no need to keep an additional node so that additional nodes' cost and energy are nearly zero. In FTDaemon, when a host is predicted to fail, the system manager will lease a new host from the service provider. This is a main weakness of the method since other hosts, which may be underutilized, will not be considered by the manager, so load balancing is not established.

In [24], reactive fault tolerance is suggested. In this method, while HPC systems are running, they send their results to checkpoints. In case of any failure, the system restarts from the point before the failure. In this method, due to increased system components, the system may not be able to restart repeatedly. The technique leads to decreased energy consumption. Also, the method does not suit the systems needing overuse of VMs or clusters because failures lead to a significant decrease in availability.

In [25], Power-Check fault tolerance has been suggested, which increases the monitoring level in HPC systems using specific intelligent data. The method causes decreased CPU use, lower system performance, and optimized energy consumption.

Yakhchi et al. [26] suggested Cuckoo Optimization Algorithm-Minimum Migration Time (COA-MMT) algorithm to provide load balancing. This algorithm is based on the life of cuckoos. COA-MMT has three steps for load balancing and power consumption management: At the first stage, an over-utilized host is detected. To do this, some hosts are selected randomly and clustered. Using profit function and according to equation (1), the profit value of habitat or cluster is determined. Applying equation (2), for each host, the eggs are laid in a specific range called Egg Laying Radius (ELR) [16]. The host with the most CPU utilization is selected as the overused host.

$$profit = f_p(habitat) = f_p(x_1.x_2.\ldots.x_{Nvar}) \quad (1)$$

In equation (1), f_p denotes the profit function.

$$ELR = \alpha * \frac{Number\ of\ current\ cuckoos\ eggs}{Total\ number\ of\ eggs} * (var_{hi} - var_{low}) \quad (2)$$

In equation (2), α is an integer, supposed to handle the maximum value of ELR. var_{hi} and var_{low} stand for upper bound and lower bound, respectively, which are used for defining ELR. At the second stage, an under-loaded host is detected. The host experiencing the minimum CPU utilization is selected as the host with the least loading value. At the third stage, selection policy is implemented, and one or more VMs are selected to migrate to the host with minimum CPU utilization. Minimal Migration Policy Time (MMT) selects the VMs needing less time to migrate to other hosts. Migration time is calculated by equation (3).

$$v \in V_j \mid \forall\, a \in V_j, \frac{RAM_u(v)}{NET_j} \leq \frac{RAM_u(a)}{NET_j} \quad (3)$$

In equation (3) V_j is a set of VMs currently allocated to host j. NET_j denotes the spare network bandwidth available for the host j; and $RAM_u(a)$ is the amount of RAM currently utilized by the VM a. This method decreased the violation of SLA compared with Bee-MMT. Using this method leads to increased QoS and satisfaction of users.

Tamilvizhi and Parvathavarthini in [27] proposed a concept of fault management with the emphasis on the

hardware and network faults handling. This proposed work introduces an innovative perspective on adopting a fault-tolerant mechanism to avoid network congestion and health monitoring for fault detection with migration techniques to handle faults adaptively. This work's primary goal is to develop an effective cloud architecture that could tolerate fault occurrences beforehand or after hand and then suggest appropriate solutions to maintain data traffic and the system's availability, thus making it more reliable and flexible.

Neelima and Reddy in [28] proposed a load balancing task scheduling algorithm in the cloud using the Adaptive Dragonfly algorithm (ADA), which provides minimum time and cost while balancing the load. In this method, to attain better performance, a multi-objective function is developed based on three parameters: completion time, processing costs, and load. Based on the multi-objective function, we assign a task to VM. The proposed methodology's main objective is to assign the task to VM using ADA, which minimizes the total execution time and cost while balancing the load.

Durga Devi et al. [29] proposed a dynamic load balancing in a heterogeneous environment by Modified Adaptive Neuro-Fuzzy Inference System (MANFIS). Parameters of MANFIS are optimized by introducing Fire-fly Algorithm. In this method, the adopted Modified Adaptive Neuro-Fuzzy Inference System (MANFIS) for VM load balancing is based on the CPU utilization and turnaround time. Also adopted Enhanced Elliptic Curve Cryptography to provide security between cloud users and cloud servers. There are two key implication of proposed methodology. First, is to optimize load balancing based on CPU utilization and Turnaround time. Second, is to provide data security using Enhanced Elliptic Curve Cryptography.

Kong et al. [30] proposed a fast heuristic algorithm based on the zero imbalance approach as a new concept in the heterogeneous environment. This approach focuses on minimizing the completion time difference among heterogeneous VMs without priority methods and complex scheduling decision to the particular cloud configuration. This mechanism consists of combining load balancing and task allocation. To achieve this mechanism, this algorithm collects each task's size, the processing speed of each VM, the bandwidth of each VM, the number of VMs and tasks, as information to implement load balancing and task allocation in the balancing phase. Moreover, the assignment of tasks is performed on any VMs under the control of modified optimal completion time. The proposed algorithm identifies the suitable VMs for the appropriate unassigned tasks based on earliest finish time in the task allocation phase. Table 1 shows a comparison between the mentioned algorithms.

Proposed Method

As shown in the related work section, FTDaemon is not perfect, which results in increasing migration overheads, increasing system cost, increasing energy consumption, decreasing system performance speed, ignoring underutilized hosts, inappropriate load balancing, unequal use of hardware resources by some hosts, ignoring QoS, and violating SLA. An energy-efficient fault tolerance approach has been suggested to predict and avoid failure occurrence in HPC systems. The proposed algorithm is called Daemon-COA-MMT (DCM). The method causes decreased migration overheads, decreased system performance speed, optimal use of underutilized hosts instead of leasing new hosts, appropriate load balancing, equal use of hardware resources by all hosts, focusing on QoS and SLA, and a significant decrease of energy consumption. Our method employs four modules. To predict and prevent failures, the proposed method utilizes these parameters: CPU temperature, CPU utilization, number of users' requests, voltage, and fan speed parameters. The architecture of the method consisted of some modules, is illustrated in Fig. 1.

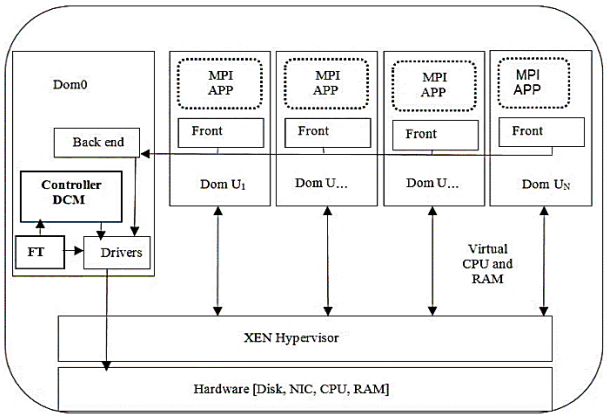

Fig. 1: The architecture of the proposed method.

The proposed method consists of four modules:

- Node monitoring with Lm sensors module
- Rule-based predictor module
- Migration policy module based on COA-MMT
- Controller module of DCM

The modules are described as follows:

A. Host Monitoring Modules in Proposed Method

In the proposed method, Lm sensors are used since most HPC systems run Linux, and Lm sensors utilize the Linux operating system. Lm sensors cause the development of the DCM method, which could easily be deployed on HPC systems in clouds. HPC systems, possessing more than 100000 CPUs, impose massive overhead on the networks and HPC systems. Therefore,

the method should check the parameters periodically to reduce monitoring overhead.

The information collected at intervals of 600 seconds, which are changeable, is sent to the host. Whenever the monitoring parameters inside the Lm sensors exceed the maximum value, the alarm will be triggered, which indicates that the failure is likely to occur.

B. Rule-Based Prediction Modules in Proposed Method

DCM Fault Tolerance is executed on each node in the user's space, and the failure is predicted based on the history of failure, maximum operating values, and information obtained from the system.

When the monitoring node with Lm sensors indicates a failure, the rule-based predictor module runs. The rule-based predictor module inputs are five parameters: temperature T, voltage V, fan speed F, CPU utilization C, and several user's requests R from the host.

The reason for using number of user's requests is that the second module investigates hardware resources and the cause of creating workload for the host's hardware resources. To calculate the respective host's actual state, specific fixed weight values are assigned to the host. The values are 1, 1.5, 2, 2.5, and 3 for the very good, good, normal, alarm, and critical areas, respectively (Table 2).

Table 1: Comparison and summary of previous methods

Ref	Approach	Advantage	Disadvantage
[17]	An improved particle algorithm to achieve resource load balancing optimization in the cloud environment	Improve Resource utilization, Good Performance	It is valid for equal-sized population
[20]	a Genetic Algorithm based resource management algorithm for allocating cloud-based virtual machines on physical machines	Obtained an optimized distribution strategy	High computational overhead
[21]	The comparison of load balancing techniques and BAT algorithm techniques are described	provides optimal utilization of all resources	Increased number of requests causes a longer response time
[22]	An algorithm to detect over utilized hosts and then migrate VMs based on artificial bee colony algorithm (ABC)	Greater power consumption saving, Decreasing the CO2 emission and operational cost	High complexity for selecting best overloaded host, No prediction for future workload of hosts
[23]	Energy efficient fault tolerance for HPC in the cloud that develop a generic FT algorithm for HPC systems in the cloud.	Reduced the energy consumption of computation-intensive applications	Low accuracy of failure prediction mechanism that is unsuitable for HPC workload.
[24]	Fault Tolerance (FT) approach to HPC systems in the cloud to reduce the wall clock execution time in the presence of faults	Improved the execution time, reduce energy consumption	Does not suit the systems needing overuse of VMs or clusters
[25]	A power-aware check pointing framework Power-Check to address the problem of marginal energy benefits	Reduction in the amount of energy consumed, improving the check pointing performance	Job partitioning however not considered in this approach.
[26]	An approach based on Cuckoo Optimization Algorithm (COA) to detect over-utilized hosts.	Reduced the power consumption	May be cause SLA violation
[27]	Adopting a fault tolerant mechanism to avoid network congestion and health monitoring for fault detection with migration technique	Reduced energy consumption and cost overhead	This method no worries about how to cover the error
[28]	A load balancing task scheduling algorithm in cloud using Adaptive Dragonfly algorithm (ADA)	Well-balanced load across virtual machines	High computational overhead
[29]	Dynamic load balancing in a heterogeneous environment is handled by Modified Adaptive Neuro Fuzzy Inference System (MANFIS)	Improving the turnaround time and maximizing the CPU utilization	High communication overhead
[30]	A fast heuristic algorithm based on the zero imbalance approach, as a new concept in the heterogeneous environment	strikes the balance between the requirements of cloud users and providers	Ignoring power consumption in the data center and live VM migration.

Table 2: Weight of parameters based on measured

CPU Utilization	Number of Users' Requests from Host	Fan Speed	Voltage	Temperature	Weight of Parameter
0-16	0-50	0-500	0.94-0.85	0-15	1
16-32	50-100	500-1000	0.94-1.03	15-30	1.5
32-48	100-150	1000-1500	1.03-1.12	30-45	2
48-64	150-200	1500-2000	1.12-1.21	45-60	2.5
64-80	200-250	2000-2500	1.21-1. 30	60-75	3

As shown in Table 2, the host is at an excellent state when parameters are as follows: temperature 0-15, voltage 0.85-0.94, fan speed 0-500, CPU utilization 0-16, and the number of requests 0-50. The weight of each parameter is 1. The rule-based predictor module is shown in Fig. 2.

As shown in Fig. 2, the main parameters $(T_i, F_i, V_i, C_i, R_i)$ are inserted into a calculating module and a_c values are obtained from equation (5). The result is compared with the threshold, and the output of the rule-based predictor module is obtained. The threshold is calculated based on system log, constructive information, current sensor values, and CPU utilization values.

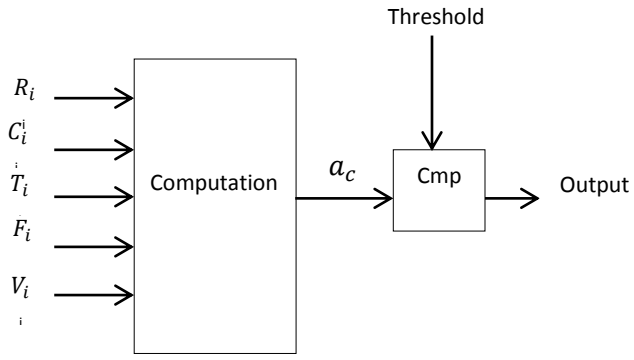

Fig. 2: Rule based predictor module.

In (4), K_c constant is employed to improve the accuracy of prediction. As K_c is negligible, we set $K_c = 0$. On calculating a_c based on determining ranges in (5), the host state is determined based on ac at the current state.

$$a_c = \prod_{i=1}^{n} T_i * F_i * V_i * C_i * R_i + K_c \quad (4)$$

As shown in (6), five states can be assigned to each host. The categorization is based on these five parameters: temperature (T), voltage (V), fan speed (F), CPU utilization (C), and the number of user's requests (R) from the host. These five areas are used to improve the accuracy of prediction and detect suspicious hosts immediately. The method of determining the state of hosts is shown in Table 3.

$$a_c = \begin{cases} Critical\ State & 117.18 \le |a_c| \le 243 \\ Alarm\ State & 40 \le |a_c| \le 97.66 \\ Normal\ State & 10.12 \le |a_c| \le 32 \\ Good\ State & 1.5 \le |a_c| \le 7.59 \\ Very\ Good\ State & |a_c| = 1 \end{cases} \quad (5)$$

C. Migration Policy Based on Proposed Method

When an alarm is triggered, the migration policy is activated.

It is not needed to lease a new host to eliminate the alarm state since the third module examines all hosts to find underutilized hosts.

The purpose of the method's migration policy is to complete computation-intensive computation with minimum energy consumption. In the DCM algorithm, when a failure is predicted, COA-MMT load balancing is executed.

On investigating the load balancing area, we chose the COA-MMT load balancing algorithm for the third module due to its rapid and exact detection of optimal point, providing appropriate load balancing and SLA, increasing QoS, and containing MMT policy. COA-MMT technique is executed in three steps to provide load balancing in the system.

In the third module of the proposed method, the COA-MMT load-balancing algorithm is implemented in two steps to establish load balancing and manage power utilization. According to the method, the host monitoring and rule-based predictor modules are determined based on the over-utilized host's hardware parameters. Hence, the COA-MMT load balancing algorithm does not need to search for the over-utilized host, which causes overheads and increased energy consumption. Migration policy based on COA-MMT optimization includes two steps: detecting the under loaded host and selection policy.

Table 3: Calculating a_c based on the parameters of the proposed method

State	Amounts aC					Threshold
Critical	3*2.5*2.5*2.5*2.5 =117.18	3*3*2.5*2.5*2.5 =140.62	3*3*3*2.5*2.5 =168.75	3*3*3*3*2.5 =202.50	3*3*3*3*3 =243	117.18
Alarm	2*2*2*2*2.5 =40	2*2*2*2.5*2.5 =50	2*2*2.5*2.5*2.5 =62.50	2*2.5*2.5*2.5*2. 5 = 78.12	2.5*2.5*2.5*2.5* 2.5 = 97.66	40
Normal	1.5*1.5*1.5*1.5*2 =10.12	1.5*1.5*1.5*2*2 =13.50	1.5*1.5*2*2*2 =18	1.5*2*2*2*2 =24	2*2*2*2*2 =32	10.12
Good	1*1*1*1*1.5 =1.5	1*1*1*1.5*1.5 =2.25	1*1*1.5*1.5*1.5 =3.37	1*1.5*1.5*1.5*1. 5 = 5.06	1.5*1.5*1.5*1.5* 1.5 = 7.59	1.5
Very Good	1*1*1*1*1 =1	_	_	_	_	1

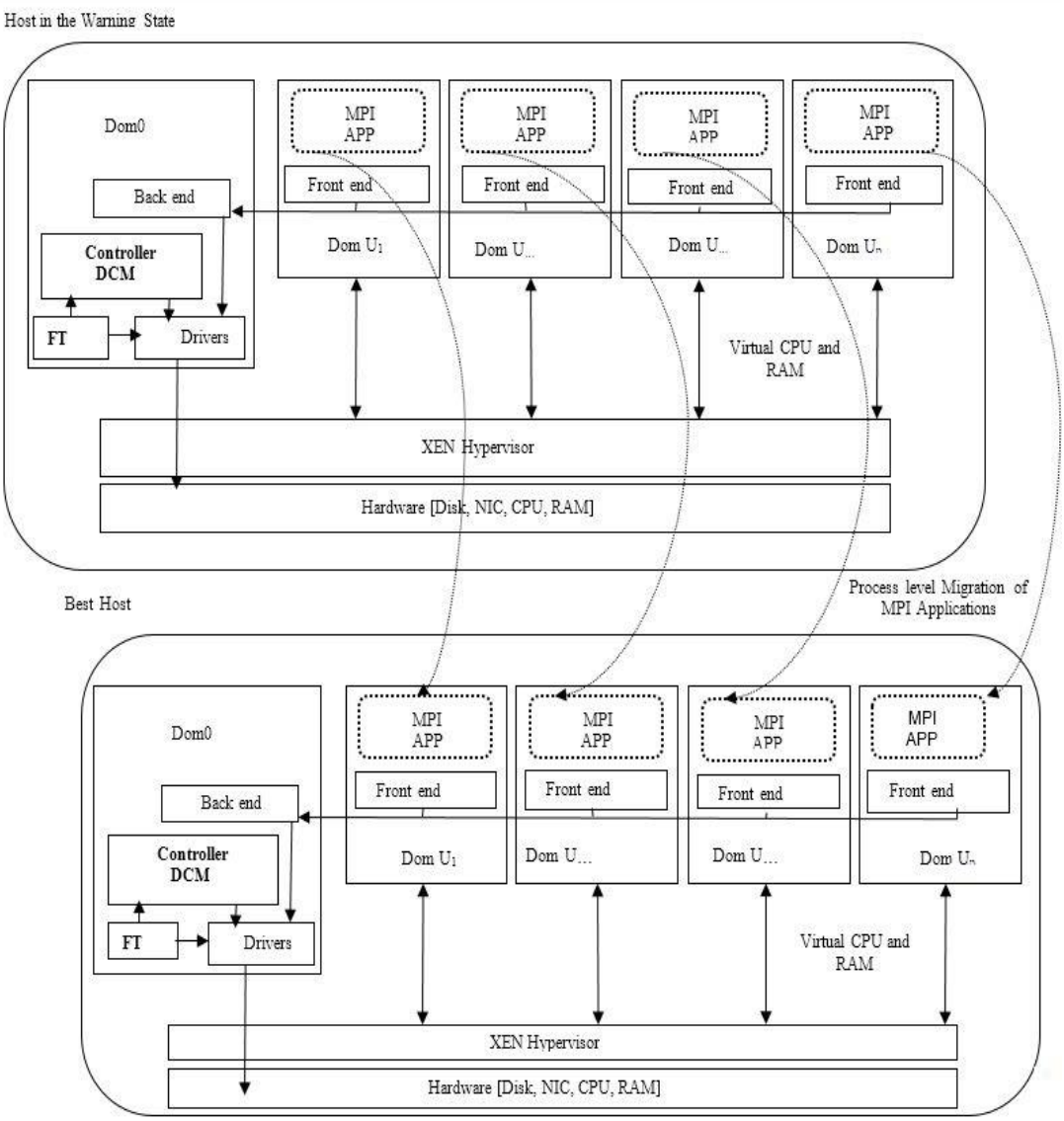

Fig. 3: Process level migration from the host, in warning state to the proper host.

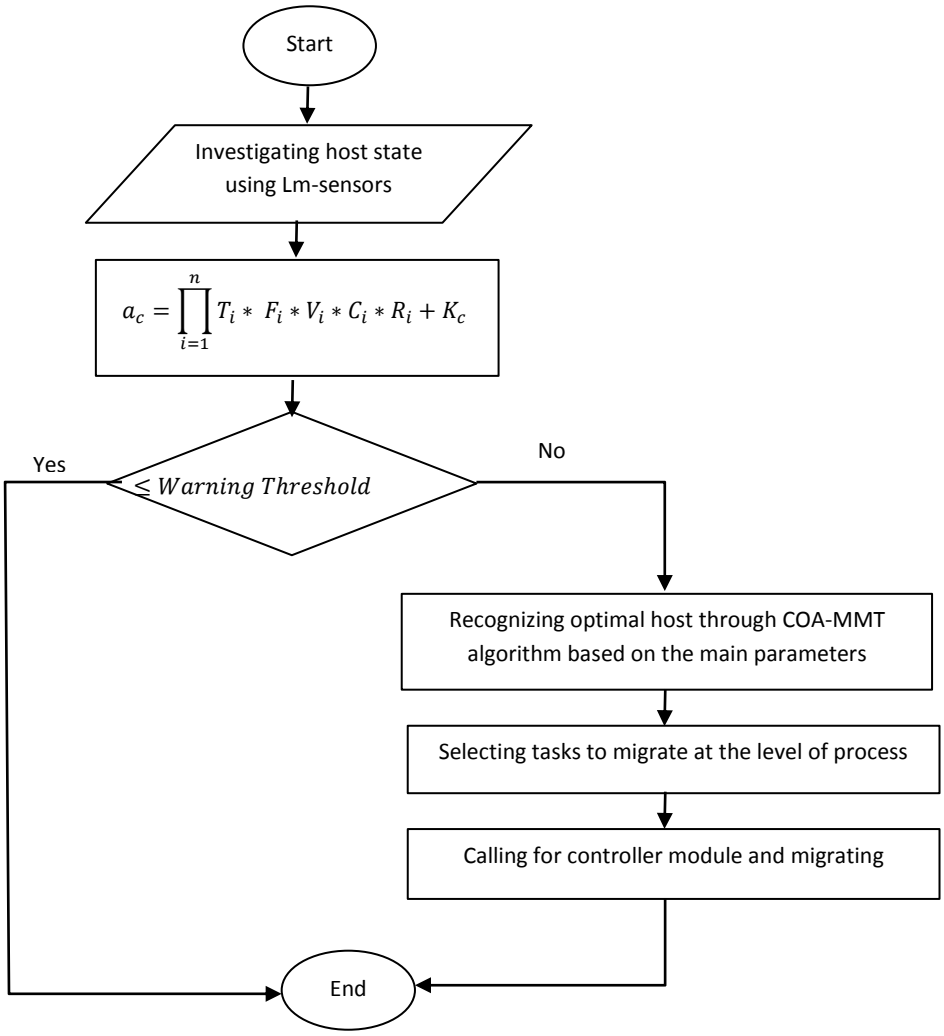

Fig. 4: Flowchart of the proposed method.

D. Controller Module in Proposed Method

The controller module is responsible for the implementation of three introductory modules. It is installed on all nodes. On predicting a failure, the controller module is called for, and the following steps are taken:

- Several VMs installed on a host, which is in the alarm state, request some information about their current host from the information center.
- The ID of programs running on the unhealthy VMs installed on the current host is obtained through the information center.
- An appropriate host, determined by COA-MMT, is selected
- The process level migration from a host in an alarm state to a host in the proper state is performed.
- The details of the running VMs on a proper host is published to the head host.

In Fig. 3, the process level migration from a host, which is in an alarm state to a proper host, is shown. Once the method detects a host, which is in an alarm state, the host is selected based on COA-MMT, and the controller module performs process level migration to a proper host.

The flowchart of the proposed method is illustrated in Fig. 4. First, the state of the host is examined by Lm sensors. Second, five main parameters are selected, and a_c is calculated through equation (5). The result is compared with the warning threshold. If a_c is lower than the warning threshold, the host state will be proper, and other steps are redundant. Otherwise, using the COA-MMT algorithm, the proper host is selected. The tasks are selected on the host, which is at the alarm state. Finally, process level migration is performed. Algorithm 1 shows the pseudo-code for proposed method.

Simulation and Result

Using Cloudsim 3.0, the proposed method, DCM, has been simulated. The proposed method's efficiency has been evaluated in scenarios A, B, and C compared with Power-Check [15] and Tamilvizhi et al. method [27]. In scenario A, five users with five brokers, and two data centers have been created. The first data center contains three hosts, while the second data center contains two hosts. Ten VMs are also created using the Time-Shared policy, each with 512 MB and one CPU managed by Xen, as Virtual Machine Manager (VMM), on Linux operating system. The host's memory is 2048 MB, with a storage capacity of 1,000,000 MB and a bandwidth of 10,000 Mb/sec. The number of submitted tasks (cloudlets) ranges between 10 and 100, each with 600 MB files.

In scenario B, we set 10 cloud users with ten brokers and five data centers. Each data center contains three hosts, making a total of 15 hosts. A total of 25 VMs are also created using the Time-Shared policy, each with 512 BM and one CPU managed by Xen, as VMM, on Linux operating system.

The host memory is 2048 MB, with a storage capacity of 1,000,000 MB and a bandwidth of 10,000 Mb/sec. Moreover, the number of submitted tasks ranges between 50 and 500, each with 1000 MB files.

In scenario C, fifteen cloud users with fifteen brokers and eight data centers have been created. Each data center contains three hosts, making a total of 24 hosts. 30 VMs are also created using the Time-Shared policy, each with 512 BM and one CPU managed by Xen, as VMM, on Linux operating system. The host's memory is 2048 MB, with a storage capacity of 1,000,000 MB and a bandwidth of 10,000 Mb/sec. Also, the number of submitted tasks ranges between 500 and 1000, each with a file size of 1400 MB. To improve accuracy, simulation is performed ten times in a row.

Table 4 illustrates the conditions and parameters of the simulation.

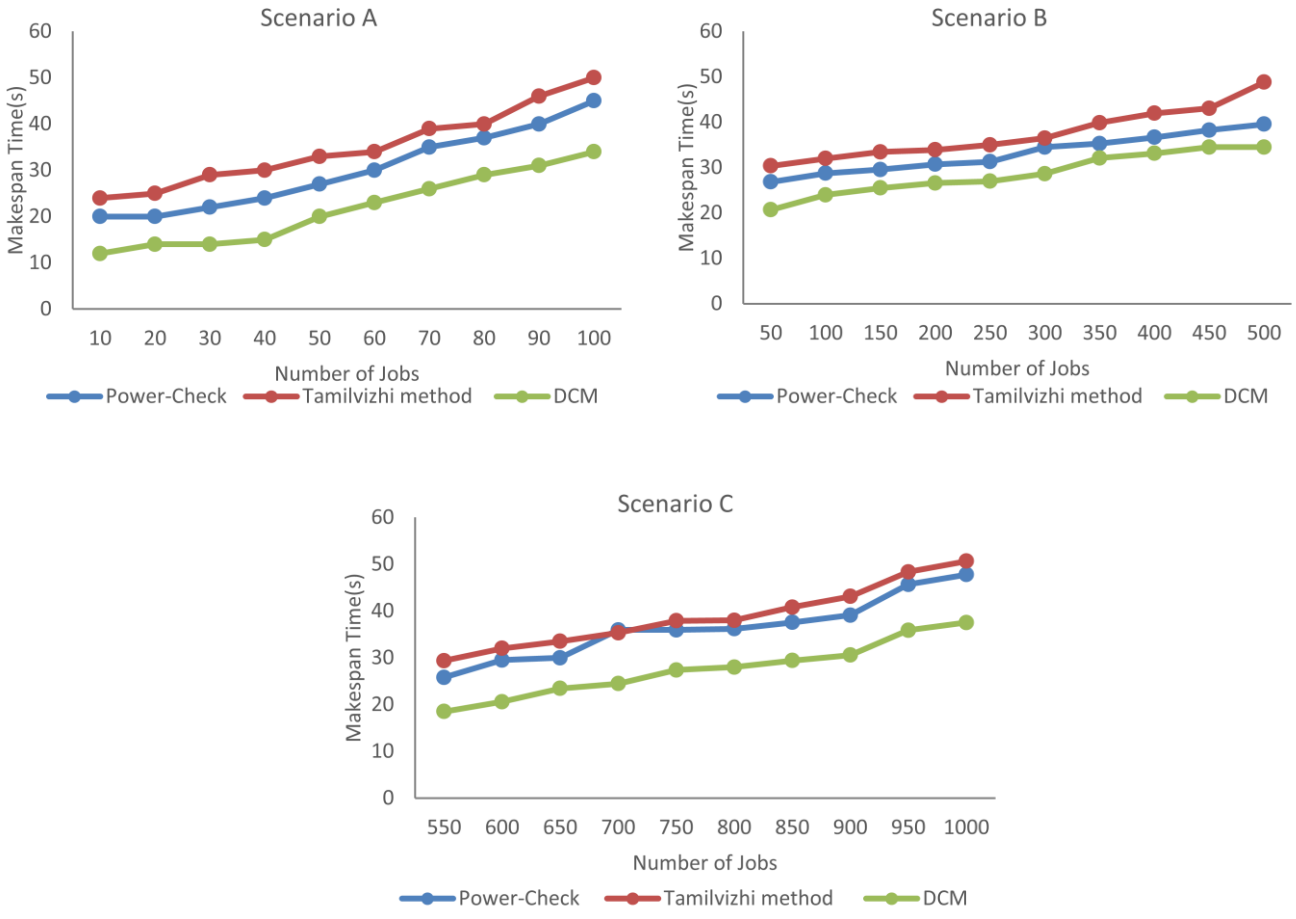

Fig. 5: Average job makespan. a.senario A's makespan time. b.senario B's makespan time. c. senario C's makespan time.

Table 4: Conditions and simulation parameters

Scenarios	A	B	C
Cloud users	5	10	15
Brokers	5	10	15
Data centers	2	5	8
Virtual Machines	10	25	30
Bandwidth (Mb/sec)	10000	10000	10000
Total Host	5	15	24
Number of Tasks	10 - 100	50 - 500	550 - 1000
Storage Capacity	1000000	1000000	1000000
host memory(MB)	2048	2048	2048

The average job makespan, average response time, failure rate, energy consumption, and average task execution costs are presented compared to two other algorithms.

The interval between request and completion of the request is called job makespan. Figure 5 shows the average job makespan of the proposed method in scenario A, B, and C compared with Power-Check and Tamilvizhi method.

Algorithm 1: Proposed method

```
Initialization: (n: number of host, T: temperature, V: voltage, F: fan speed, C: CPU utilization, R: several user's requests)
for (i=1 ; i<= n ;i++)
    Using Lm sensors, the host_i state mode is obtained (T_i, F_i, V_i, C_i, R_i);
    Determine weight of parameter for (T_i, F_i, V_i, C_i, R_i);
    Calculating ac values for host_i;
    Determine the state of host_i;
    if (a_c > Threshold)
        Recognizing optimal host through COA-MMT algorithm;
        Selecting tasks to migrate at the level of process
        Calling for controller module and migrating
        The details of the running VMs on a proper host is published to the head host.
    end if
end for
end
```

As shown in Fig. 5, in scenario A, the average job makespan has improved compared with other methods. Also, an increased number of tasks results in improving the method. In scenario B, when the number of tasks changes from 50 to 500, which is more than that of scenario A, the average job makespan of the method is less compared with other methods. Also, in scenario C, when the number of tasks changes from 550 to 1000, the average job makespan improves substantially. Decreased average job makespan shows that the method's task execution time is lower compared with other methods.

The most optimal host is chosen for migration in the proposed method, and load balancing is established. Thereby, its average job makespan is decreased by 9.50% and 18.06%, respectively, compared with the Tamilvizhi method and Power-Check.

In the proposed method, first the important information of the nodes is collected using sensors and then the status of the nodes is checked for fault by using the prediction module. If a node is at faulty, jobs will be properly transferred to the appropriate machines by migration policy agents. In this way, a proper load balance will be created on the proposed method and makespan is reduced.

Fig. 6 shows the failure rate (FR), calculated by equation (6). Here, FR is calculated concerning the total failure of the system.

In the proposed method, by using the module to predict the status of the node and check the status of the node in terms of fault, an attempt is made to prevent fault in machines. Also, by performing the migration operation properly, a stable situation is provided to prevent fault. As shown in Fig. 6, increased workload and number of tasks cause the failure rate to decrease in HPC systems.

In the failure rate of the DCM method is decreased compared with the Tamilvizhi method and Power-Check by 21.03% and 10.21%, respectively.

Therefore, there is a relationship between average job makespan, failure rate, and reliability in HPC systems since the decrease of average job makespan leads to increased failure rate and reliability.

$$FR = \frac{1}{MTTF} \quad (6)$$

where FR is failure rate and MTTF is min time to failure. Equation (6) derives the rate of failure of our method concerning the system's total failure.

The interval between request and the first response is called response time.

The proposed method's response time compared with

the Tamilvizhi method and Power-Check in three considered scenarios is shown in Fig. 7.

As shown in Fig. 7, the method results in proper load balancing between all hosts. As mentioned, in the proposed method, by applying the appropriate migration method, a good load balancing is created.

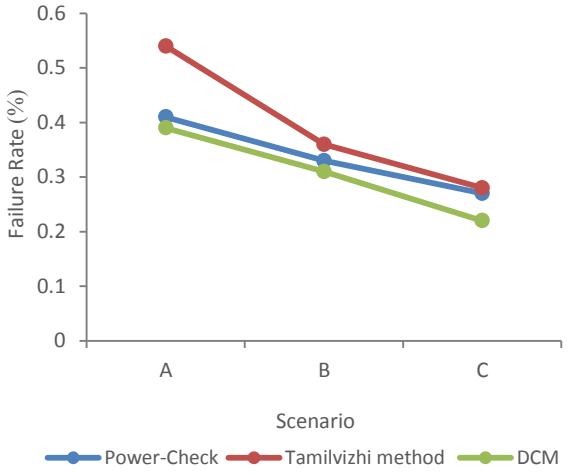

Fig. 6: Investigating the rate of failure in three scenarios.

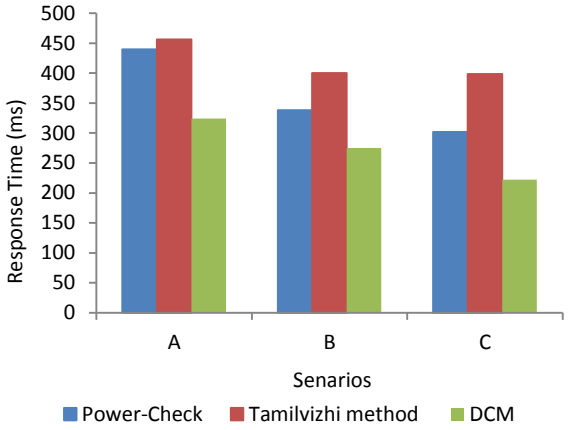

Fig. 7: Average Response Time.

Balancing the load between the machines makes the jobs on the machines faster. Also, the productivity of the CPUs of the system is increased. Therefore, the average response time is decreased compared with the Tamilvizhi method and Power-Check by 45.83% and 35.68%, respectively.

The costs that the service provider incur to respond to users' requests are called task execution costs. The average task execution costs of the proposed method, in comparison with the Tamilvizhi method and Power-Check, in these three considered scenarios, are shown in Fig. 8.

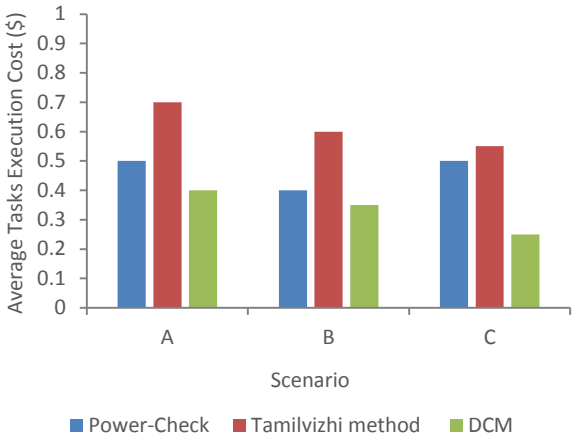

Fig. 8: Average Tasks Execution Costs

As shown in Fig. 8, the most proper host is selected for migration in the proposed method. Due to the exact prediction of failure, the possibility of failure and disturbance in tasks' performance is significantly reduced.

Also, the computational overhead is minimized, and the speed of the system is increased. Therefore, the method declines the average task execution costs compared with the Tamilvizhi method and Power-Check by 44.71% and 44.16%, respectively.

Fig. 9 illustrates the proposed method's average energy consumption compared to the Tamilvizhi method and Power-Check in these three considered scenarios.

The proposed method employs an exact load balancing method to minimize migration time (Fig. 9). In the method, the proper host for migration is attentively selected.

Therefore, on average, the DCM fault tolerance algorithm's energy consumption is optimized by 30% compared with that of other methods. Also, the energy consumption of the Tamilvizhi method is higher compared with that of others.

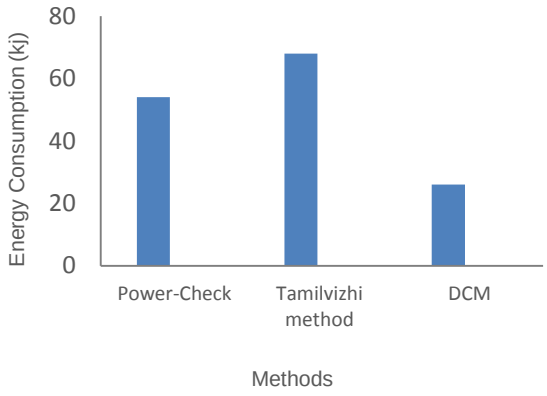

Fig. 9: Energy Consumption in HPC Systems.

Conclusion

In recent years, cloud computing has become a popular computing technology in all industries and provides more benefits than other technologies. One of the main challenges of cloud computing is fault tolerance, which avoids restarting the system and declines operational costs and energy consumption. In this paper the DCM method to enhance FTDaemon is proposed. In the proposed method, the Daemon Fault Tolerance technique has been enhanced, and COA-MMT has been utilized for load balancing.

The method consists of four modules, which are used to determine the host state. To predict and prevent failures, the proposed method utilizes these parameters: CPU temperature, CPU utilization, number of users' requests, voltage, and fan speed parameters. Based on evaluations and simulations, the proposed method is significantly optimized in task execution costs, job makespan time, and response time and declines the energy consumption compared with the Tamilvizhi method and Power-Check.

Author Contributions

H. Barati and A. Mehranzadeh conceptualized the research. H. Jahanpour designed the experiments and collected the data and she carried out the data analysis. H. barati and H. Jahanpour validated the results. H. Jahanpour wrote the manuscript. H. Barati and A. Mehranzadeh reviewed and edited the manuscript.

Acknowledgment

The authors would like to thank Dezful Branch, Islamic Azad University.

Conflict of Interest

The author declares that there is no conflict of interests regarding the publication of this manuscript. In addition, the ethical issues, including plagiarism, informed consent, misconduct, data fabrication and/or falsification, double publication and/or submission, and redundancy have been completely observed by the authors.

Abbreviations

IT	Information Technology
HPC	High-Performance Computing
DCM	Daemon-COA-MMT
QoS	Quality of Service
SLA	Service Level Agreement
VM	Virtual Machine
SaaS	Software as a Service
IaaS	Infrastructure as a Service
HaaS	Hardware as a Service
PSO	Particle Swarm Optimization
Bee-MMT	Bee colony Algorithm-Minimal Migration Time
MMT	Minimal Migration Policy Time
CPU	Central Processing Unit
FTDaemon	Daemon's fault tolerance
COA-MMT	Cuckoo Optimization Algorithm-Minimum Migration Time
ELR	Egg Laying Radius
ADA	Adaptive Dragonfly algorithm

References

[1] M. Vaishnnave, K.S. Devi, P. Srinivasan, "A survey on cloud computing and hybrid cloud," Int. J. Appl. Eng. Res., 14: 429-434, 2019.

[2] M.U. Bokhari, Q. Makki, Y.K. Tamandani, "A survey on cloud computing," Big Data Analytics: 149-164, 2018.

[3] F.A. Ibrahim, E.E. Hemayed, "Trusted cloud computing architectures for infrastructure as a service: Survey and systematic literature review," Computers & Security, 82: 196-226, 2019.

[4] A.M. Caulfield, E.S. Chung, A. Putnam, H. Angepat, D. Firestone, J. Fowers, et al., "Configurable clouds," IEEE Micro, 37(3): 52-61, 2017.

[5] F. Zafar, A. Khan, S.U.R. Malik, M. Ahmed, A. Anjum, M.I. Khan, et al., "A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends," Computers & Security: 65, 29-49, 2017.

[6] K. O'brien, I. Pietri, R. Reddy, A. Lastovetsky, R. Sakellariou, "A survey of power and energy predictive models in HPC systems and applications," ACM Computing Surveys (CSUR), 50(3):1-38, 2017.

[7] M.A. Netto, R.N. Calheiros, E.R. Rodrigues, R.L. Cunha, R. Buyya, "HPC cloud for scientific and business applications: taxonomy, vision, and research challenges," ACM Computing Surveys (CSUR), 51(1): 1-29, 2018.

[8] A. Pradhan, S.K. Bisoy, P.K. Mallick, "Load Balancing in Cloud Computing: Survey," Innovation in Electrical Power Engineering, Communication, and Computing Technology: 99-111, 2020.

[9] M.R. Mesbahi, A.M. Rahmani, M. Hosseinzadeh, "Reliability and high availability in cloud computing environments: a reference

roadmap," Human-centric Computing and Information Sciences, 8(1): 20, 2018.

[10] M.N. Cheraghlou, A. Khadem-Zadeh, M. Haghparast, "A survey of fault tolerance architecture in cloud computing,"Journal of Network and Computer Applications, 61: 81-92, 2016.

[11] A. Rezaeipanah, M. Mojarad, A. Fakhari, "Providing a new approach to increase fault tolerance in cloud computing using fuzzy logic," International Journal of Computers and Applications: 1-9, 2000.

[12] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, et al., "Predicting Node failure in cloud service systems. in Proc. the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering: 480-490, 2018.

[13] A.A. Shaikh, S. Ahmad, "Fault tolerance management for cloud environment: a critical review," International Journal of Advanced Research in Computer Science, 9(Special Issue 2): 34, 2018.

[14] A. Hota, S. Mohapatra, S. Mohanty, "Survey of different load balancing approach-based algorithms in cloud computing: a comprehensive review," Computational intelligence in data mining: 99-110, 2019.

[15] P. Kumar, R. Kumar, "Issues and challenges of load balancing techniques in cloud computing: A survey," ACM Computing Surveys (CSUR), 51(6): 1-35, 2019.

[16] M. Kumar, S.C. Sharma, "Dynamic load balancing algorithm to minimize the makespan time and utilize the resources effectively in cloud environment," International Journal of Computers and Applications, 42(1), 108-117, 2020.

[17] K. Pan, J. Chen, "Load balancing in cloud computing environment based on an improved particle swarm optimization," in Proc. 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS): 595-598, 2015.

[18] F. Abazari, M. Analoui, H. Takabi, S. Fu, "MOWS: multi-objective workflow scheduling in cloud computing based on heuristic algorithm," Simulation Modelling Practice and Theory, 93: 119-132, 2019.

[19] M. Abd Elaziz, S. Xiong, K.P.N. Jayasena, L. Li, "Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution," Knowledge-Based Systems, 169: 39-52, 2019.

[20] Y.L. Huang, Z.X. Li, "A GA-based resource management algorithm for smart living applications requiring intensive computing power," in Proc. 2017 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW): 259-260, 2017.

[21] S.S. Abdhullah, K. Jyoti, S. Sharma, U.S. Pandey, "Review of recent load balancing techniques in cloud computing and BAT algorithm variants," in Proc. 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom): 2428-2431, 2016.

[22] S.M. Ghafari, M. Fazeli, A. Patooghy, L. Rikhtechi, "Bee-MMT: A load balancing method for power consumption management in cloud computing," in Proc. 2013 Sixth International Conference on Contemporary Computing (IC3): 76-80, 2013.

[23] I.P. Egwutuoha, S. Chen, D. Levy, B. Selic, R. Calvo, "Energy efficient fault tolerance for high performance computing (HPC) in the cloud," in Proc. 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD): 762-769, 2013.

[24] I.P. Egwutuoha, S. Chen, D. Levy, B. Selic, R. Calvo, "A proactive fault tolerance approach to High Performance Computing (HPC) in the cloud," in Proc. 2012 Second International Conference on Cloud and Green Computing (CGC): 268-273, 2012.

[25] R.R. Chandrasekar, A. Venkatesh, K. Hamidouche, D.K. Panda, "Power-check: An energy-efficient check pointing framework for HPC clusters," in Proc. 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid): 261-270, 2015.

[26] M. Yakhchi, S.M. Ghafari, S. Yakhchi, M. Fazeli, A. Patooghi, "Proposing a load balancing method based on Cuckoo Optimization Algorithm for energy management in cloud computing infrastructures," in Proc. 2015 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO): 1-5, 2015.

[27] T. Tamilvzhi, B. Parvathavarthini, "A novel method for adapive fault tolerance during load balancing in cloud computing," Cluster Computing, 22(5): 10425-10438, 2019.

[28] P. Neelima, A.R.M. Reddy, "An efficient load balancing system using adaptive dragonfly algorithm in cloud computing," Cluster Computing, 23: 2891.2899, 2020.

[29] T.D. Devi, A. Subramani, P. Anitha, "Modified adaptive neuro fuzzy inference system based load balancing for virtual machine with security in cloud computing environment," Journal of Ambient Intelligence and Humanized Computing, 1-8, 2020.

[30] L. Kong, J.P.B. Mapetu, Z. Chen, "Heuristic load balancing based zero imbalance mechanism in cloud computing," Journal of Grid Computing, 18(1): 123-148, 2020.

Biographies

Hoda Jahanpour received her B.Sc. degree in Computer Engineering from Dezful Branch, Islamic Azad University, Dezful, Iran, in 2014. Furthermore, she received her M.Sc. degree in computer systems Architecture from Dezful Branch, Islamic Azad University, Dezful, Iran, in 2016. Her major research interests include Distributed Computing and cloud computing.

Hamid Barati is an Assistant Professor in the Department of Computer Engineering at Dezful Branch, Islamic Azad University, Dezful, Iran. He received his B.S. degree in Computer Hardware Engineering, M.S. degree in Computer Systems Architecture Engineering and Ph.D. degree in Computer Systems Architecture Engineering in 2005, 2007 and 2015 respectively. Currently he is Faculty of Islamic Azad University, Dezful Branch, Iran. His major research experiences and interests include mobile ad hoc networks, interconnection networks and energy-efficient routing and security issues in wireless sensor networks.

Amin Mehranzadeh received the M.Sc. and Ph.D. degrees in Computer Engineering. He is currently an Assistant Professor in Computer Engineering Department at Azad University of Dezful. His research interest is Distributed Computing, Cloud Computing, Embedded Systems and Network-on-Chip systems including Performance and Cost Improvement in Routing and Arbitration of various types of NoC systems. Recently, he has started carrying out research in Deep Neural Network with his team which consists of M.Sc. and Ph.D. students.

How to cite this paper:
H. Jahanpour, H. Barati, A. Mehranzadeh, "An Energy Efficient Fault Tolerance Technique Based on Load Balancing Algorithm for High-Performance Computing in Cloud Computing," Journal of Electrical and Computer Engineering Innovations, 8(2): 169-182, 2020.

DOI: 10.22061/JECEI.2020.7219.371

URL: http://jecei.sru.ac.ir/article_1467.html