Energy-aware Fault-tolerant Scheduling Scheme based on Intelligent Prediction Model for Cloud Data Center

Avinab Marahatta[1,2], Ce Chi[2], Fa Zhang[2], Zhiyong Liu[2]
[1]*University of Chinese Academy of Sciences, China*
[2]*High Performance Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences, China*
Email:{avinab.marahatta, chice18s, zhangfa, zyliu}@ict.ac.cn

***Abstract*—As cloud computing becomes increasingly popular, more and more applications are migrated to clouds. Due to multi-step computation of data streams and heterogeneous task dependencies, task failure occurs frequently, resulting in poor user experience and additional energy consumption. To reduce task execution failure as well as energy consumption, we propose a novel energy-aware proactive fault-tolerant scheduling scheme for cloud data centers(CDCs) in this paper. Firstly, a prediction model based on machine learning approach is trained to classify the arriving tasks into "failure-prone tasks" and "non-failure-prone tasks" according to the predicted failure rate. Then, two efficient scheduling mechanisms are proposed to allocate two types of tasks to the most appropriate hosts in a CDC. Vector reconstruction method is developed to construct super tasks from failure-prone tasks and schedule these super tasks and non-failure-prone tasks to most suitable physical host, separately. All the tasks are scheduled in an earliest-deadline-first manner. Our evaluation results show that the proposed scheme can intelligently predict task failure and achieves better fault tolerance and reduces total energy consumption than existing schemes.**

***Index Terms*—Cloud computing, Cloud data center, Scheduling, Fault-tolerance, Energy-efficiency, Task failure, Prediction**

I. Introduction

Cloud data center (CDC) is the home that supports the computation, storage and various applications of individuals and enterprises. CDC provides facilities to host a wide range of applications, such as health care, scientific computing, smart grid, e-commerce and nuclear science [1], in different domains. The task and resource failures are inevitable due to the growing number of CDC resources which provide ICT infrastructures. However, reliable on-demand resources are needed for service provider to fulfill Service Level Agreement (SLA) of customers [2]. Therefore, it is of great significance to ensure the reliability and availability in such systems [3].

However, modeling and examining dynamic fault-tolerant technique for virtualized CDC is challenging. First, cloud applications are usually large-scale and consist of a huge number of distributed computing nodes. The complex structure and fault-tolerant behavior of heterogeneous CDC applications are hard to describe. Second, CDC applications dynamically adjust the virtual machine(VM) configurations to meet user requests which require multi-dimensional resources, such as CPU, RAM, disk storage. The concurrency and uncertainty of requests will rise the complication of model validation and verification in the scheduling.

This work was partially supported by the National Key Research and Development Program of China (grant number 2017YFB1010001); the National Natural Science Foundation of China (grant numbers 61520106005, 61761136014). The Corresponding Author is Zhiyong Liu (zyliu@iact.ac.cn).

Proactive fault tolerant techniques are widely adopted in CDCs [4], [5]. However, efficient implementation of proactive fault tolerance relies heavily on the prior knowledge of the failed tasks. Generally, the task is failed due to over-utilization of resources, unavailable resources, hardware failures, execution time or execution cost exceeds the threshold value, required libraries are not installed properly, system running out of memory or disk space, and so on.

Previous studies have adopted various fault tolerance mechanisms, such as check-pointing [6], [7], migration [4], load balancing [8], replication [9], [10], retry [11], [12], task resubmission [13], etc. Existing failure prediction techniques are mainly based on statistical approaches [5]. Recently, data have very complex structures and parameters. So, simple statistical approach might not capture the patterns in more complex data. Some studies have used machine learning approach to predict task failure [14], [15], and they have not leveraged the prediction to facilitate the task scheduling. In addition, while the spare resources used in proactive fault-tolerant schemes, that will result in additional energy consumption, very limited fault-tolerant studies take optimization of energy consumption into consideration.

In this paper, our aim is to predict a task failure according to the requested resources before the actual failure occurs, and leverage the prediction to design task scheduling scheme, thus to reduce the task execution failure as well as the total energy consumption. To this end, a novel Prediction based Energy-aware Fault-tolerant Scheduling scheme (PEFS) is proposed. The scheme involves two stages: 1) prediction of task failure probability, and 2) task scheduling. In the first stage, task parameters (involving the requested resources, actually allocated resources, and whether failure occurred) are gathered from historical data set. Then, all the task parameters are inserted into TensorFlow[1] as inputs. Using deep neural network approach, a model is trained to predict the failure

[1]https://www.tensorflow.org/.

rate of each arriving task. In this way, all the arriving tasks can be classified into failure-prone tasks and non-failure-prone tasks based on model outputs. In the second stage, a scheduling algorithm based on vector bin packing is proposed to schedule the two types of tasks efficiently. The main difference between these two scheduling processes is that, for the failure-prone tasks, super tasks are generated firstly based on an elegant vector reconstruction method for fault-tolerant purpose. Replication strategy is applied to replicate only the fault-prone tasks, then arranged into super tasks in a way of vector reconstruction that the execution of different copies of the tasks in different hosts will not be overlapped so that redundant execution will be avoided. To verify the accuracy of the prediction, comprehensive experiments are conducted to compare the predicted failures with the actual failures. We also compare the total energy consumption, resource utilization, and task failure ratio of the proposed scheme with existing methods to show its superiority.

The main contribution of this work includes:

- Use deep neural network for prediction of possible failures of tasks, and train a model that can predict failure tasks with accuracy above 84%.
- Arriving tasks are classified into two categories based on prediction results, and two scheduling schemes are proposed to schedule the two types of tasks separately to reduce task failure rate as well as energy consumption.
- A unique fault-tolerant mechanism is developed to schedule failure-prone task by constructing super tasks based on vector reconstruction method.
- Conduct extensive simulation experiments in CloudSim[2] toolkit to evaluate our scheme. The results validate that, our scheme outperforms the state-of-the-art in terms of failure ratio, resource utilization, and energy consumption.

The rest of the paper is organized as follows: Section II discusses the related work and Section III presents the system design and models. Section IV describes the proposed scheduling scheme. Section V details the experimental setup and quantitative analysis. Section VI reports the experimental results and discussion. Section VII concludes the paper.

II. Related Work

When multiple task instances from different applications start to execute on numerous hosts, some of the hosts may fail accidentally, resulting in a fault in the system. This phenomenon is usually avoided by fault tolerance mechanism [9]. Various factors lead to host failure. Besides, a failure event usually stimulates another fault event. These failures may include operating system crashes, network partitions, hardware malfunctions, power outages, abrupt software failures, etc. [16].

The existing fault-tolerant techniques in CDCs mainly include replication, check-point, job migration, retry, task resubmission, etc. Some studies [17], [18] introduced methods based on certain principles, such as retry, resubmission, replication, renovation of software, screening and migration, to harmonize fault-tolerant mechanism with CDC task scheduling. However, for parallel and distributed computing systems, the most adopted and acknowledged method is to replicate data to multiple hosts [9].

A heuristic approach called resubmission impact has been proposed [19]. However, the resubmission may cause considerable delay. A classification based energy-aware fault-tolerant dynamic scheduling scheme has been presented in [9] with elastic resource provisioning. The tasks are classified using Bayes classifier, replicated and mapped to most suitable virtual machines.

A rearrangement based improved fault-tolerant scheduling algorithm (RTFR) has been presented to deal with the dynamic scheduling issue for tasks in cloud systems [20]. A primary-backup model is adopted to realize fault-tolerance in this method. The corresponding backup copy will be released after the primary replica is completed, so as to release the resource it occupies. Besides, the waiting tasks can be rearranged to utilize the released resources. Whereas in most existing algorithms, after the task is sent to the waiting queue of the virtual machine, the execution sequence is fixed and cannot be changed.

The backup overlapping mechanism and virtual machine migration strategy are adopted in cloud to improve resource utilization [21]. It is a primary-backup approach similar to the proposal in [20]. A dynamic integrated task scheduling algorithm is presented in [22] by modifying breadth first search algorithm to find the overall optimal virtual machine for each task. The above techniques can not only minimize the makespan and response time for tasks, but also can be easily integrate with other virtual machine management techniques to reduce energy consumption and improve fault tolerance.

By observing resource behavior during job execution, various statistics and probability based techniques can be used to identify the failure rate of jobs. The history of resource failure can be leveraged for reliable selection of resources and fault-tolerant scheduling [23]. It depends on a scheduling indicator, which is used to generate the scheduling decision whenever a job arrives to a grid scheduler. The scheduling technique selects resources with the lowest failure rate.

A multi-constrained load balancing fault-tolerant scheduling is proposed to reduce the makespan, cost, and task failure rate while improving resource utilization [24]. Resource selection is made on the basis of initial failure rates, number of jobs submitted, successfully executed jobs and processing capabilities of the resources. A new scheduling approach named PreAntPolicy is proposed [25] that consists of a prediction model based on fractal mathematics and a scheduler on the basis of an improved ant colony algorithm. The prediction model determines whether to trigger the execution of the scheduler by virtue of load trend prediction, and the scheduler is responsible for resource scheduling while minimizing energy consumption under the premise of guaranteeing the Quality-of-Service (QoS). The combination of energy-aware optimal allocation and consolidation algorithm is developed to as a bin packing problem with a minimum power consumption objective [26].

An adaptive fault-tolerant job scheduling strategy is pro-

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

posed by [27]. It is a checkpointing method. The proposed strategy dynamically updates the failure index based on the successful completion of the assigned tasks, so as to maintain grid failure index. The resource broker uses fault index from the scheduler to apply different scheduling intensity for arriving tasks. The success and the fault index value of the resource is decreased if job completes within the defined time.

As far as we know, no work has been done so far using deep neural network to predict task failure and leveraging the prediction information to facilitate the energy-aware fault-tolerant scheduling in CDCs as we do in this paper.

III. Model Design

As mentioned above, the Prediction based Energy-aware Fault-tolerant Scheduling scheme (PEFS) proposed in this paper involves two stages: 1) failure prediction and, 2) task scheduling, as illustrated in Fig. 1. Predictor is designed based on deep neural network to train and test the task in historical data set (HDS). This predictor is used to predict the probability of task failure, and based on this probability, tasks are classified into failure-prone and non-failure-prone tasks. Failure-prone and non-failure-prone tasks are organized in failure-prone task queue and non-failure task queue, respectively, then schedule these types of tasks using scheduling method, separately. The power model is adopted to address the energy saving concern of the CDC. Similarly, the fault model is used to design fault-tolerant mechanism for the failure-prone tasks.

A. Task and Resource Model

In a CDC environment, the service providers receive independent tasks submitted by the end-users. A set of independent tasks is given as $T = \{T_1, T_2, T_3, \cdots, T_k\}$. Each task is associated with a set of parameters $T_k^{\varphi} = T_k^p, T_k^m, T_k^s$, where T_k^p, T_k^m and T_k^s represents the CPU, RAM and disk storage required to execute a given task T_k. In addition, T_k can be modeled as $T_k = (t_k^a, t_k^d, t_k^l)$, where t_k^a, t_k^d and t_k^l represents the arrival time, the deadline and the work volume, respectively. The tasks are categorized into failure-prone and non-failure-prone tasks in accordance with their proneness to failures. A set of failure-prone tasks is designed as $T = \{T_1, T_2, T_3, \cdots, T_l\}$ for failure-prone scheduling process based on fault-tolerant mechanism. Similarly, a set of non-failure-prone tasks $T = \{T_1, T_2, T_3, \cdots, T_l\}$ is scheduled by using non-failure-prone scheduling process.

A CDC consists of a set of hosts $H = \{H_1, H_2, \cdots, H_i\}$, providing the physical infrastructures for creating virtualized resources to satisfy the end-users requirements. V_j^{φ} is the virtual machine requirement, which is modeled as $V_j^{\varphi} = \{V_j^p, V_j^m, V_j^s\}$, where V_j^p, V_j^m and V_j^s represent the parameters of CPU, RAM and disk storage, respectively.

B. Power Consumption Model

In this paper, the power consumption of the host H_i, indicated as P_i, is expressed as below (based on energy consumption model proposed in [28]).

$$P_i(t) = P_i^{idle} + (P_i^{max} - P_i^{idle}) \times U_i(t) \tag{1}$$

where P_i^{idle} represents the idle power consumption of the H_i. Similarly, P_i^{max} represents the maximum power consumption of the H_i.

Assuming that all the CPU cores are homogeneous. i.e., c = 1,2,$\cdots$, PE_i: $MIPS_{i,c} = MIPS_{i,1}$. The CPU utilization of the host H_i at time t, indicated as $U_i(t)$, is defined as the average percentage of the total allocated computing powers of $V_i(t)$ that is assigned to the H_i. It can be expressed as following.

$$U_i(t) = (\frac{1}{PE_i \times MIPS_{i,1}}) \sum_{c=1}^{PE_i} \sum_{j \subseteq V_i(t)} mips_{j,c} \tag{2}$$

The total energy consumption of the host in time period $[t_1, t_2]$ is formulated as below:

$$E_i = \int_{t_1}^{t_2} P_i(U_i(t))dt \tag{3}$$

where,
$U_i(t)$: the utilization of the host H_i at time t; $0 \leq U_i(t) \leq 1$.
PE_i: the number of cores of the host H_i.
$mips_{j,c}$: the assigned MIPS of the c^{th} core to the V_j on the host H_i.
$MIPS_{i,c}$: the max computing power capacity of the c^{th} core on the host H_i.

C. Fault-tolerant Model

The task failures may occur due to unavailability of resources, hardware failures, execution cost and time exceed than threshold value, system running out of memory or disk space, over-utilization of resources, improper installation of required libraries, and so on. These faults can be transient or permanent, and are assumed to be independent. So, developing fault-tolerant scheduling scheme needs to guarantee the deadline of all the tasks in the system that are met before faults occurs even under the worst-case scenario.

As we know, replication strategy is widely used for fault tolerance, which generally replicates tasks into two or more copies, then schedule to different hosts. So, there are more possibilities of wastage of resources and increase the unusual energy consumption. Thus, in this paper, only failure-prone tasks are replicated. First, three consecutive tasks are taken from failure-prone task queue, and each task is replicated into three copies. Then, vector reconstruction method is designed to reconstruct super task from replicate copies as shown in Section IV B and Fig. 3. Reconstructed super tasks are mapped to the most suitable hosts, allocated with resources, and then scheduled in different hosts, separately. The sequence of replicate copies of tasks in super tasks are designed as shown in Fig. 3, so that the execution of different copies of the tasks in different hosts will not be overlapped to avoid redundant execution.

IV. Scheduling Scheme

Upon task arrival in the system, a task joins the queue of the entire system in earliest-deadline-first manner. Then,

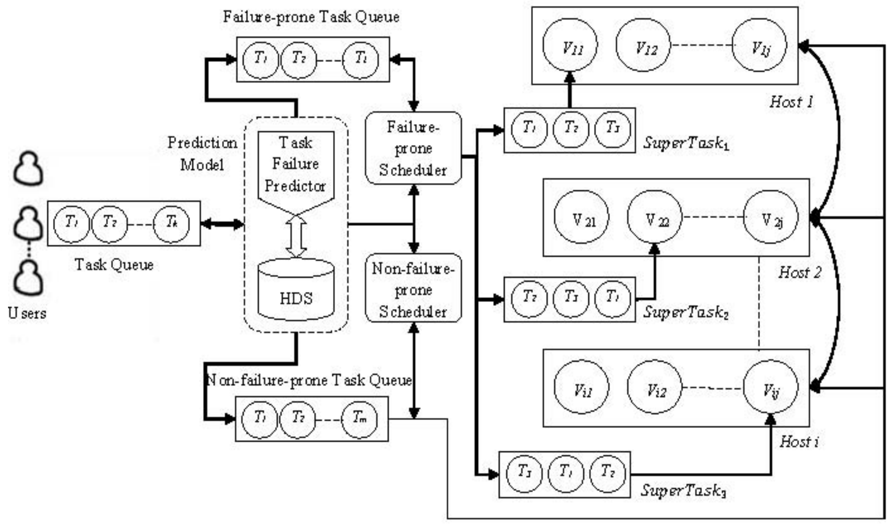


Fig. 1: System diagram.

predictor classifies the task into failure-prone task and non-failure-prone task in accordance with their predicted failure probability. Based on the classification, failure-prone task queue and non-failure-prone task queue are generated. Vectors are constructed for both failure-prone and non failure-prone tasks before mapping to hosts. Then they are organized and scheduled, separately, as shown in Algorithm 2.

A. Task Failure Prediction

The task failure prediction involves multiple steps as shown in Fig. 2, which includes preprocessing, training the preprocessed data, prediction of task failure as well as checks the accuracy of predicted results.

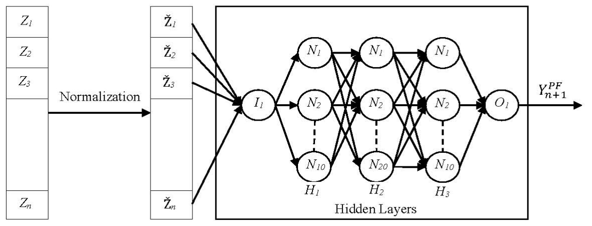


Fig. 2: Deep neural network for task failure prediction.

The Internet Data Set[3] [29] is used to train the predictor. Let the data set used for normalization is defined by Z and normalized data set is Ž. The equation (4) is used to normalize the data set Z in the range (0, 1). $\check{Z}_i = \dfrac{Z_i - Z_{min}}{Z_{max} - Z_{min}}$ (4)

where Z_{max} and Z_{min} are the maximum and minimum values respectively obtained from data set Z. The normalized data Ž is fed into the network as input which in followed by training and evaluation of the network along with task failure prediction.

The architecture with $p-q-r$ input parameters are defined, where p, q and r represents the number of neurons in input, hidden and output layers, respectively. Actual failures are extracted and analyzed to predict failure ratio of upcoming new task on the CDC. Input data set (Z) and corresponding output (Y) are constructed in equation (5), where each Z_i denotes one data point and Z_j represents the number of actual requests received.

$$Z = \begin{bmatrix} Z_{11} & Z_{21} & \dots & Z_{1l} \\ Z_{21} & Z_{22} & \dots & Z_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{i1} & Z_{k2} & \dots & Z_{kl} \end{bmatrix}, Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_k \end{bmatrix} \tag{5}$$

In this paper, percentage split is used to divide the data set into training set and testing set, where training data is used to train the prediction model and testing data is applied to evaluate the accuracy of predicted results. The accuracy of the predicted task failures is measured using root mean squared error (ě) and mean absolute percentage error (ê) given by equation (6) and equation (7), respectively.

$$\check{e} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i^{PF})^2} \tag{6}$$

$$\hat{e} = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i - Y_i^{PF}|}{Y_i} \tag{7}$$

where, Y_i^{PF} and Y_i are the predicted and actual task failure, respectively, and n is the total number of samples. The prediction algorithm is described in Algorithm 1.

B. Vector Reconstruction

Failure-prone tasks are arranged in failure-prone task queue in earliest-deadline-first manner. First, three consecutive tasks are taken from failure-prone task queue, and each task is replicated into three copies. Then, vector reconstruction method is designed to reconstruct super task from replicate copies as shown in Fig. 3, $SuperTask_1, SuperTask_2, SuperTask_3$. Each requested resources of each task from consecutive three tasks is evaluated, then highest requested resource values are chosen, i.e, resource

[3]Whole experiment is conducted based on this data set.

Algorithm 1 Failure Prediction

```
1:  procedure PREDICTION()
2:      Initialize weight vector $w_j$
3:      For each all $Z$ data set ($i = 1$ to $n$)
4:          For each $j = 1$ to $n$
5:              Evaluate output $Y_i^{PF} = \sum_{j=1}^{n} Z_j w_j$
6:              Compare output $Y_i^{PF}$ with actual output $Y_i$
7:              Error $Y_i - Y_i^{PF}$
8:              Use Gradient Descent to minimize the error
9:              Adopt $w_j$ in current layer
10:             Repeat until ě and ê has been minimized.
11:         End For
12:     End For
13: end procedure
```

vector $< CPU_{high}.RAM_{high}.Disk_{high} >$ is computed for all of three super tasks. Then, These super tasks are mapped to the most suitable hosts having sufficient resources, separately.

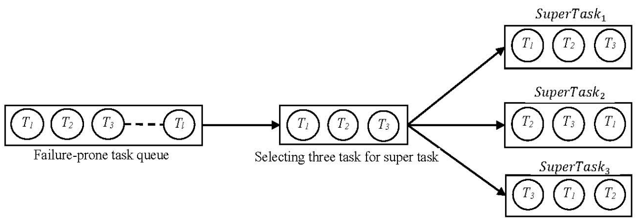


Fig. 3: Failure-prone scheduling with preparing super task (fault-tolerant mechanism).

For example, Let T_1, T_2, T_3 are three consecutive failure-prone tasks (first three predicted failure tasks from failure-prone task queue), and they have different vector such as,
T_1= $< CPU.RAM.Disk >$ = $< 4.2.5 >$,
T_2= $< CPU.RAM.Disk >$ = $< 6.4.3 >$,
T_3= $< CPU.RAM.Disk >$= $< 5.7.4 >$.

First, requested resources (CPU, RAM, Disk) are compared among these consecutive three tasks. From comparison, we can see that T_1 has highest disk: 5, T_2 has highest CPU: 6, and T_3 has highest RAM:7. Then, highest resource vector value $< CPU_{high}.RAM_{high}.Disk_{high} >$ is chosen for the resource vector value of super tasks. Thus, final resource vector value $< 5.6.7 >$ is allocated to each of the super tasks, i.e.,
$SuperTask_1 = 5.6.7$,
$SuperTask_2 = 5.6.7$,
$SuperTask_3 = 5.6.7$

C. Vector Bin Packing

A resource allocation on different hosts can be considered as a vector bin packing problem, where VMs are treated as bins. Greedy algorithm is one of the most natural heuristics for one dimensional bin packing, generally mentioned to as first fit decreasing (FFD) [30]. All the items are sorted by its size in decreasing order, and the three dimensions of the requirement of CPU, RAM, and storage are used in sorting, and then placed

Algorithm 2 Task Scheduling Procedure

```
1:  procedure PEFS()
2:      Task $T_k$, historical data set, $VM_List$- a set of VMs, and $Host_List$- a set of hosts
3:      Predict task failure: PREDICTION()
4:      If $T_k$ prediction status fail
5:          Keep $T_k$ on failure-prone task queue
6:          Choose first three task from failure-prone task queue i.e., $T_1, T_2, T_3$ from { $T_1, T_2, T_3, \cdots, T_l$ }
7:          Create $SuperTask_1, SuperTask_2, SuperTask_3$ using vector reconstruction method
8:          $VMList$= Sort VM list by order(enough resources)
9:          $HostList$= Sort host list by order(energy consumption)
10:         $a \longleftarrow$ size of $HostList$, $b \longleftarrow$ size of $VMList$
11:         For $i = 1$ to $a$
12:             For $j = 1$ to $b$
13:                 If $V_j$ resources can schedule $SuperTask$
14:                     $m \longleftarrow$ order of sub tasks of $SuperTask$
15:                     For $n = 1$ to $m$
16:                         If $T_n$ of $SuperTask$ already not executed
17:                             Schedule $T_n$ of $SuperTask$ to $V_j$ of $Host_i$
18:                         End If
19:                     End For
20:                 End If
21:             End For
22:         End For
23:     Else
24:         Keep $T_k$ on non-failure-prone task queue
25:         Create Vector for Task $T_k$
26:         $VMList$= Sort VM list by order(enough resource)
27:         $HostList$= Sort host list by order(energy consumption)
28:         a $\longleftarrow$ size of $HostList$, b $\longleftarrow$ size of $VMList$
29:         For $i = 1$ to $a$
30:             For $j = 1$ to $b$
31:                 If $V_j$ resources can schedule $T_k$
32:                     Schedule $T_k$ to $Host_i$
33:                 End If
34:             End For
35:         End For
36:     End If
37: end procedure
```

sequentially in the first bin that has enough capacity. There are generally two natural options of generalizing FFD for scaling and normalizing across multi-dimension case to decide how to assign a weight to a vector i.e., FFD product weight (FFDProd) and FFD sum weight (FFDSum).

In this paper, we present norm-based greedy heuristic based on FFDSum that looks at the difference between the vector G^l and residual capacity c(t) under a certain norm [31]. The item vector G^l that minimizes the quantity $\sum_i w_i(G^l - c(t)_i)^2$ and the assignment does not violate the capacity constraints for the l_2 norm, from all unassigned items. FFDSum is more robust if the dimensions can be assigned smaller coefficients w_i and have a lower impact on the ordering. Average demand heuristic call FFDAvgSum, $AvgDem_i = \frac{1}{n} \sum_{l=1}^{n} G_i^l$ in i dimension is

TABLE I: Types of VM

VM Type	Cores	MIPS	RAM (MB)	Disk Storage (GB)
VM1	1	1000	1536	5
VM2	1	1500	3840	5
VM3	1	2500	871	5

chosen w_i.

V. Experiments

Experiments are conducted to validate the proposed scheme. In the experiments, the predicted task failure is compared with actual task failure to validate the failure prediction. Also, the performance of the proposed scheduling scheme PEFS is compared with some existing techniques, real-time fault-tolerant scheduling algorithm with rearrangement (RFTR) [20], dynamic fault tolerant scheduling mechanism (DFTS) [21] and modified breadth first search (MBFS) [22] as all of them are designed for fault-tolerant scheduling.

A. Experimental Setting

The experiments are conducted to validate the energy-aware fault-tolerant scheduling scheme based on intelligent prediction model by deploying the core methods presented in section III and section IV. The simulation are performed on a machine equipped with CPU (Intel(R) Core(TM) i5-3230M, 2.60GHz), RAM (8.0 GB), disk storage (750GB), Windows 10 operating system, NetBeans IDE 8.2, JDK 8.0. TensorFlow is used to implement deep neural network using the dataset. The simulation is performed using CloudSim [32] to create a CDC system that has identical hosts and heterogeneous VMs. Table I presents the types of VM in simulations.

We consider submission time, waiting time, start time, run time, end time, status of task (failed/ succeed), resources used (CPU, RAM, disk storage), etc. Similarly, task failure information have been gathered for intelligent failure prediction. Based on utilization threshold of CPU, RAM, and disk storage, the resource utilization parameters are measured. If the resource utilization parameter has value more than threshold value then status would be classified as a failed otherwise not failed. The performance (accuracy of prediction model) is measured based on various errors.

B. Parameter setting

All the hosts in CDC are identical and each host has CPU cores (2800 MIPS per core), 8192 MB of RAM, 1 TB of disk storage. The metrics used for comparison include the failure ratio of tasks in scheduling, resource utilization and total energy consumption.

Task failure ratio

This ratio represents the tasks failed because they could not be scheduled.

F_T=(Number of Failure Task)/(Total no of Task)

Resource utilization

The utilization is defined by the ratio between time taken to process tasks and total time.

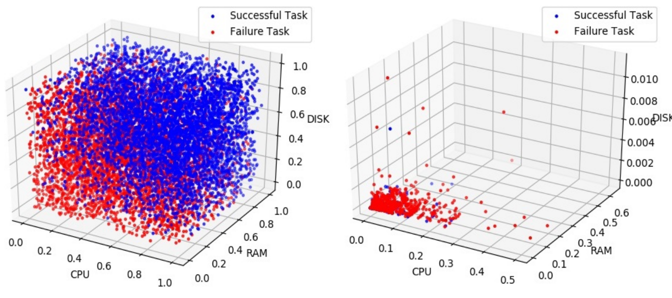
utilization=(Time Processing Tasks)/(Total Time)%

Energy Consumption

The total energy consumed by the CDC.

C. Quantitative Analysis

To demonstrate the relation between failure ratio of tasks and resources, we quantify the impacts of different parameters and their correlation. We perform correlation and covariance analysis based on the data set. The correlation shows the relation between failure task ratio and resource allocation. The analysis result shows that, task failure occurrence if the scheduler allocates minimum resources, linear relation between failure task ratio and resource allocation. Table II shows the average allocated resources for failed and succeed tasks with correlation and covariance analysis result of Internet data set [29]. Similarly, Table III shows the analysis of Google cluster data set [33], where average requested resources for failed and succeed tasks are given with their correlation analysis. From the analysis, we can see that the tasks are failed if the allocated resources are low and requested resources are high. Fig. 4 represents the relation between tasks and computing resources. Red and blue dot represents the failed and succeeded task, respectively. The three dimensions represent the requested and allocated resources (CPU, RAM and Disk Storage) for given task set. Fig. 4(a) indicates the task representation of Internet data set with resources, and Fig. 4(b) indicates the task representation of Google cluster data set with resources.



(a) Internet data set (b) Google data set

Fig. 4: Relation of failure task with resources (CPU, RAM, Disk storage).

VI. Results and Discussion

A. Accuracy of prediction model

Based on our experimental setting, we summarize the results using deep neural network based prediction model. Total 50,000 tasks are taken from Internet data set [29], where 40,000 tasks are used as testing and 10,000 tasks are used for training. Total 1 input layer, 3 hidden layers and 1 output layer are taken for deep neural network. Similarly, 10, 20 and 10 nodes are taken in first, second and third hidden layers, respectively. Errors ě and ê based on equation (6) and equation (7) are used to validate the prediction accuracy of failure task, respectively. Fig. 5 compares the actual task failures and predicted task failures to different task counts. The prediction accuracy stays above 84% however the task count grows.

B. Energy consumption

The total energy consumption is compared among these entire schemes as shown in Fig. 6. The proposed PEFS reduces energy consumption by approximately 28.66%, 21.74%

TABLE II: Internet data set analysis

Resources	Average allocated resources for failed task	Average allocated resources for succeeded task	Correlation coefficient	Covariance
CPU	0.419	0.585	0.286	0.041
RAM	0.416	0.580	0.287	0.041
Disk storage	0.419	0.577	0.275	0.040

TABLE III: Google cluster data set analysis

Resources	Average requested resources for failed task	Average requested resources for succeeded task	Correlation coefficient	Covariance
CPU	0.044	0.029	-0.135	-0.003
RAM	0.029	0.015	-0.180	-0.003
Disk storage	0.0003	0.0001	-0.1268	-0.00003

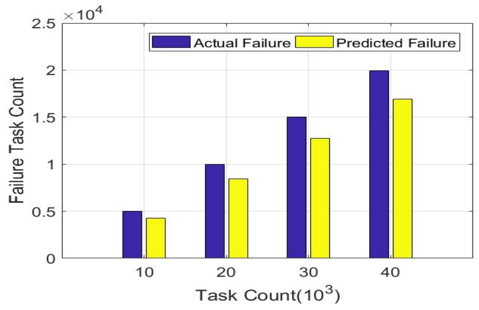


Fig. 5: Actual failure and predicted failure with task count.

and 26.10% relative to those of RFTR, DFTS and MBFS, respectively. The PEFS generates a schedule that uses lower energy consumption than the other approaches.

Fig. 7 exhibits that the total energy consumption of the four algorithms grows linearly, when task count increases. In every task count, the proposed scheme optimized energy outstandingly.

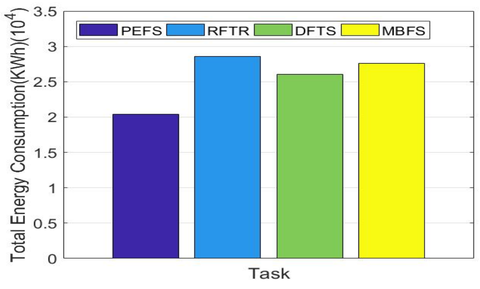


Fig. 6: Total energy consumption.

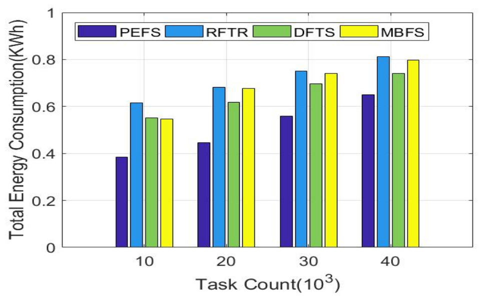


Fig. 7: Total energy consumption with task count.

C. Task Failure ratio

Fig 8 shows the task failure ratio of four algorithms. It can be seen that the failure ratio of the PEFS is lower than the other three algorithms. The proposed PEFS reduces task failure ratio by approximately 18.75%, 16.13% and 29.73% relative to those of RFTR, DFTS and MBFS, respectively.

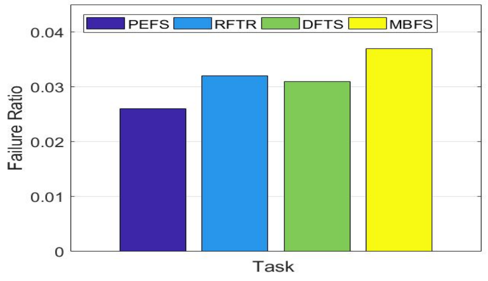


Fig. 8: Failure ratio.

D. Resource Utilization

We observe that the resource utilization of the four algorithms ascend accordingly. Fig. 9 shows that PEFS has higher resource utilization than RFTR, DFTS and MBFS. Fig. 10 shows the resource utilization of four algorithm in different task counts. This occurred due to that PEFS employs super task construction strategies to allocate failure predicted tasks to most suitable physical hosts and virtual machines, where other remaining schemes use replication mechanism for all tasks.

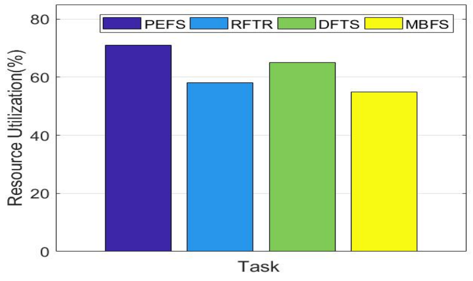


Fig. 9: Average resource utilization.

VII. Conclusion

A prediction based energy-aware fault-tolerant scheduling scheme PEFS for cloud data center is developed in this paper.

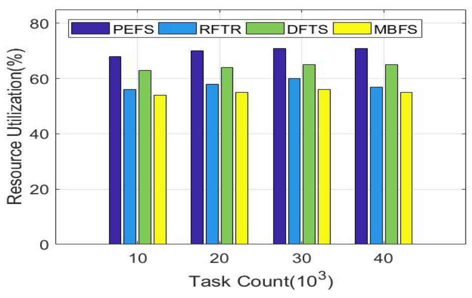


Fig. 10: Average resource utilization with task count.

First, tasks are classified into failure-prone tasks and non-failure-prone tasks using prediction model based on deep neural network and historical data set. Then, only failure-prone tasks are replicated and arranged into super tasks in a way of vector reconstruction method that the execution of different copies of the tasks in different hosts will not be overlapped so that redundant execution will be avoided. The vector bin packing method is used to schedule reconstructed super tasks and non-failure-prone tasks to most suitable hosts. Experiments on Internet Data Set are conducted, and the experimental results validate the merits of the proposed scheme in comparison with existing techniques.

Acknowledgment

The first author gratefully acknowledges CAS-TWAS President's Fellowship for funding his Ph.D. at Chinese Academy of Sciences, Beijing, China.

References

[1] K. Bilal, S. U. Khan, L. Zhang, H. Li, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, D. Chen, M. Iqbal, C.-Z. Xu, and A. Y. Zomaya, "Quantitative comparisons of the state-of-the-art data center architectures," *Concurrency Computation: Practice and Experience*, vol. 25, no. 12, 2012.

[2] D. Kliazovich, P. Bouvry, and S. U. Khan, "Dens: data center energy-efficient network-aware scheduling," *Cluster Computing*, vol. 16, no. 1, p. 6575, 2013.

[3] J. Shuja, S. A. Madani, K. Bilal, K. Hayat, S. U. Khan, and S. Sarwar, "Energy-efficient data centers," *Computing*, vol. 94, no. 12, p. 973994, 2012.

[4] J. Liu, S. Wang, A. Zhou, S. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, no. 99, 2017.

[5] O. Hannache and M. Batouche, "Probabilistic model for evaluating a proactive fault tolerance approach in the cloud," *IEEE International Conference on Service Operations And Logistics, And Informatics*, pp. 94–99, 2015.

[6] B. Mohammed, M. Kiran, M. Kabiru, and I.-U. Awan, "Failover strategy for fault tolerance in cloud computing environment," *Software: Practice and Experience*, vol. 47, no. 9, p. 12431274, 2017.

[7] J. Zhao, Y. Xiang, T. Lan, H. H. Huang, and S. Subramaniam, "Elastic reliability optimization through peer-to-peer checkpointing in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 491–502, 2017.

[8] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," *IEEE International Conference on Advanced Information Networking and Applications Workshops*, pp. 551–556, 2010.

[9] A. Marahatta, Y.-S. Wang, F. Zhang, A. K. Sangaiah, S. K. Sah Tyagi, and Z. Liu, "Energy-aware fault-tolerant dynamic task scheduling scheme for for virtualized cloud data center," *Mobile Networks and Applications*, 2018.

[10] A. Zhou, S. Wang, C.-H. Hsu, M. H. Kim, and K. S. Wong, "Network failure-aware redundant virtual machine placement in a cloud data center," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, 2017.

[11] C. Wang, L. Xing, H. Wang, Z. Zhang, and Y. Dai, "Processing time analysis of cloud services with retrying fault-tolerance technique," *IEEE International Conference on Communications in China*, pp. 63–67, 2012.

[12] G. Ramalingam and K. Vaswani, "Fault tolerance via idempotence," *SIGPLAN Not.*, vol. 48, no. 1, pp. 249–262, 2013.

[13] K. Plankensteiner, R. Prodan, and T. Fahringer, "A new fault tolerance heuristic for scientific workflows in highly distributed environments based on resubmission impact," *IEEE International Conference on e-Science*, pp. 313–320, 2009.

[14] M. A. Mukwevho and T. Celik, "Toward a smart cloud: A review of fault-tolerance methods in cloud systems," *IEEE Transactions on Services Computing*, 2018.

[15] J. Wu, P. Zhang, and C. Liu, "A novel multiagent reinforcement learning approach for job scheduling in grid computing," *Future Generation Computer Systems*, vol. 27, no. 5, p. 430439, 2011.

[16] M. A. Shafii, L. M. Shafie Abd, and B. M. Bakri, "On-demand grid provisioning using cloud infrastructures and related virtualization tools : A survey and taxonomy," *International Journal of Advanced Studies in Computer Science and Engineering IJASCSE*, vol. 3, no. 1, pp. 49–59, 2014.

[17] V. S. Kushwah, S. K. Goyal, and P. Narwariya, "A survey on various fault tolerant approaches for cloud environment during load balancing," *IJCNWMC*, vol. 4, no. 6, pp. 25–34, 2014.

[18] P. Kassian, P. Radu, F. Thomas, K. Attila, and P. Kacsuk, "Fault-tolerant behavior in state-of-the-art gridworkflow management systems," *Institute for Computer Science University of Innsbruck Attila Kert CoreGRID Technical Report Number TR-0091*, 2007.

[19] P. Kassian and P. Radu, "Meeting soft deadlines in scientific workflows using resubmission impact," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, pp. 890–901, 2012.

[20] P. Guo and Z. Xue, "Real-time fault-tolerant scheduling algorithm with rearrangement in cloud systems," *2017 IEEE 2nd Information Technology, Networking , Electronic and Automation Control Conference (ITNEC)*, pp. 399–402, 2017.

[21] J. Soniya, J. Angela, J. Sujana, and T. Revathi, "Dynamic fault-tolerant scheduling mechanism for real time tasks in cloud computing," *International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016.

[22] R. K. Yadav and V. Kushwaha, "An energy preserving and fault tolerant task scheduler in cloud computing," *IEEE ICAETR*, 2014.

[23] M. Amoon, "A fault-tolerant scheduling system for computational grids," *Comput. Elect. Eng.*, vol. 38, no. 2, p. 399412, 2012.

[24] P. Keerthika and S. P, "A multiconstrained grid scheduling algorithm with load balancing and fault tolerance," *Sci. World J.*, 2015.

[25] H. Duan, C. Chen, G. Min, and Y. Wu, "Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems," *Future Generation Computer Systems*, vol. 74, pp. 142–150, 2017.

[26] C. Ghribi, M. Hadji, and D. Zeghlache, "Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms," *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2013.

[27] B. Nazir, K. Qureshi, and P. Manuel, "Adaptive check pointing strategy to tolerate faults in economy based grid," *Journal of Supercomputing*, vol. 50, no. 1, pp. 1–18, 2009.

[28] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ISCA*, pp. 13–23, 2007.

[29] "Internet data set." [Online]. Available: https://github.com/somec001/InternetData

[30] V. V. Vazirani, "Approximation algorithms," *Springer-Verlag, New York, Inc.*, 2001.

[31] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *Microsoft Research, Tech. Rep.*, 2011.

[32] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "Cloudsim:a toolkit for modeling and simulation of cloud computing environments and evaluation of resource," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[33] "Google cluster data set." [Online]. Available: https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md