Cost Effective Hybrid Fault Tolerant Scheduling Model for Cloud Computing Environment

Hybrid Fault Tolerant Scheduling

Annabathula.Phani Sheetal[1], K.Ravindranath[2]
Department of Computer Science and Engineering
Koneru Lakshmaiah Education Foundation
Green Fields, Vaddeswaram, Guntur-522502, Andhra Pradesh, India

***Abstract*—Cloud computing provides flexible and cost effective way for end users to access data from multiplatform environment. Despite of support by the features of cloud computing, there are also chances of resource failure. Hence there is a need of a fault tolerant mechanism to achieve undisrupted performance of cloud services. The task reallocation and duplication are the two commonly used fault tolerant mechanisms. But task replication method results in huge storage and computational overhead, when the number of tasks is increasing gradually. If the number of faults is high, it incurs more storage overhead and time complexity based on task criticality. In order to solve these issues, we propose to develop a Cost Effective Hybrid Fault Tolerant Scheduling (CEHFTS) Model for cloud computing. In this model, the Failure Occurrence Probability (FoP) of each VM is estimated by finding the previous failures and successful executions. Then an adaptive fault recovery timer is maintained during a fault, which is adjusted based on the type of faults. Experimental results have shown that CEHFTS model achieves 43% reduced storage cost and 13% reduced response delay for critical tasks, when compared to existing technique.**

***Keywords*—Cloud computing; failures; fault tolerant; critical tasks; scheduling; fault recovery; overhead**

I. INTRODUCTION

Cloud computing is one of the new areas of technology that has emerged recently. On-demand resource sharing can be accomplished by utilizing internet and cloud services [1]. The primary goal is to offer services or distribute resources at the request of the customer with the lowest possible cost. There is a huge amount of heterogeneous resources [2], a vast user base, and a great range of application tasks in the cloud computing system. They have to deal with numerous individual activities, as well as big data. In terms of finance and science, cloud computing is a very valuable service [3]. A difficult challenge arises when deadlines are imposed on the jobs, since the required resources may be unavailable due to failures [4].

Along with missing project deadlines, fault tolerance is an important cloud computing aspect. It is designed to handle real-time tasks even if a failure occurs. Due to the higher storage overhead of taking backups, it is vital to use alternative approaches that have a high resource utilization ratio. Even though cloud features are attractive and a fault tolerant technique is required to guarantee uninterrupted cloud service functioning [5].

Both internal and external factors contribute to some of the flaws. The two most often utilized fault-tolerant strategies in cloud computing are task reallocation and duplication. In task reallocation, a task is reassigned after a fault occurs. This method boosts the overall system resource usage. However, failing to meet the deadline constraints [6] of jobs could cause the response time to be prolonged. The rise in the amount of electricity consumed while making resource allocations causes resource requests to fail in data centers.

Both traditional application components and new traditional application components may be heterogeneous in their component composition, composed of scientific techniques. With virtualization technology, these components have their own specialized execution environment. Workflows can also be implemented and run in cloud environments that offer an ever-growing resource pool [7] that you pay just for what you use. On-demand cloud resources can thus be acquired and shared on workflows. Clouds serve as a favored scientific process execution environment due to these characteristics.

However, processes that operate in the clouds face obstacles, because the complexity and dynamics of processes and cloud breakdowns occur more frequently. Failures tend to stop continuous implementation and have considerable impacts on the performance of processes, particularly for big, long-run workflows. It is crucial that scientific workflow scheduling fault tolerants be achieved in order to provide consumers with flawless experience.

Due to the time-consciousness of many scientific procedures, their successful completion does not depend exclusively upon the correct results but also on the time when those results are available. For a workflow that must complete the working phase and generate the computer results, a deadline is specified for the workflow. Failures lead to deadline conscious workflows that cannot be concluded on time without an appropriate faults-tolerant schedule. In that circumstance, QoS is badly damaged, although beyond the deadline[8] the results may be acquired. Therefore, it is required and crucial to plan a defect tolerant workflow since processes can be successfully completed before the deadline.

This paper is organized as follows. Section 2 presents the motivation and objectives. Section 3 presents the proposed methodology. Section 4 presents the simulation results. Section 5 presents the conclusion.

II. Motivation and Objectives

In fault-tolerant workflow scheduling [6], task replication method is used. Here duplicate multiple copies of tasks [9] can execute simultaneously. But it results in huge storage and computational overhead, when the number of tasks is growing. Moreover, it did not present the strategy for finding the exact number of duplicate copies.

The Power, memory, and other network elements were addressed in order to enhance the reliability of resources. In task scheduling, the VM with the greatest level of reliability is picked. Hussein El Ghor et al. [8] has presented two scheduling algorithms. The first approach uses energy-efficient fault-free scheduling and the second approach provides essential slack time for fault recovery. The proposed model is limited in handling small group of tasks that cannot improve the performance levels.

Redundancy is a common way to protect against errors that can be done quickly or in space. Any additional resources required to perform the identical duplication or resilience replication process are designated as spatial redundancy. When considered in depth, an example of spatial redundancy may be parallel execution. There are considerable space expenses when you simultaneously run many jobs on different resources. In the case of redundancy, the task that was missed with respect to initial resources does not have to be redone because the clock was reset.

With Fault Tolerance Algorithm utilizing Selective Mirrored Tasks Approach (FAUSIT), they proposed a selective mirrored task method to ensure the correct balance between parallelism and topology for applications. DAG-based application fault tolerance is dealt with by limiting the number of make spans and minimizing computation costs.

For important jobs or tasks with permanent failures, the dynamic fault-tolerant workflow scheduling [10] uses spatial re-execution (SRE) and temporal re-execution (TRE) schemes in unison. However, as the number of issues detected for important operations is greater, the SRE strategy consumes more disc space. If the number of faults for non-critical tasks is excessive, then it has an adverse effect on time complexity. Additionally, the backups' odds of failure are not examined.

In FESTAL [11], for each primary task, a backup task is created and VMs are allocated for both. When the primary task is completed successfully within the execution time, then the execution of back task is terminated. But in this approach also, it results in huge storage and computational overhead [12], when the number of tasks becomes high. The performance levels are not satisfactory as the model causes overhead in the system.

In [13], a fault tolerant technique to ensure task completion deadline, has been proposed. But it does not check the types of tasks and storage overhead also is more that improves the delay levels in the system.

In [15], a trust based scheduling technique is proposed. For selecting the trusted computation service, set-based particle swarm optimization (S-PSO) and for selecting the storage service, ser covering problem (SCP) tree search are applied. The task scheduling concept in [16] is built on trust mechanisms. This model uses the Bayesian cognitive technique to determine the trustworthiness of computer nodes by examining the trust relationships that exist between them.

A research paper [17] outlines a trust-based Meta heuristic workflow scheduling technique called TMWS. The manual offers strategies and procedures to avoid the hazards associated with workflow scheduling.

Hence the fault tolerant algorithm should meet the following objectives:

- To reduce storage overhead.
- To reduce time complexity.
- To reduce the chances of backup VM failures.
- To meet deadline of critical tasks.
- To reduce the response delay.
- To reduce the fault recovery time.
- To increase the CPU utilization.

In order to solve these issues, we propose to develop a cost effective hybrid fault tolerant scheduling scheme for cloud computing.

III. Cost-Effective Hybrid Fault Tolerant Scheduling (CHFTS) Model

A. Overview

In this paper, Cost-effective Hybrid Fault Tolerant Scheduling (CHFTS) model is proposed. Figure 1 shows the block diagram of the CHFTS model.

The CHFTS model uses previous failures and successful executions to calculate the VM failure occurrence probability (FoP). The anticipated execution time (EET) is estimated for each job Ti. At this point, the projects with deadlines of less than one week are classed as high-priority activities. The designation of some VMs as fault-tolerant based on FoP. A fixed number of primary and backup VMs are assigned to each task. A recovery timer is initiated if a failure occurs in any VM. A notification will be delivered to the failed VM, and the VM will continue execution from the point in time that the failure occurred.

B. System Model

Virtualization divides each server in the Cloud system into a series of heterogeneous virtual machines (VMs). The fact is that a VM is an essential component of a cloud system.

Suppose that a cloud system offers a set of VM resources, as given by.

$VM=\{VM(1),...,VM(k),....,VM(k)\}$

to users.

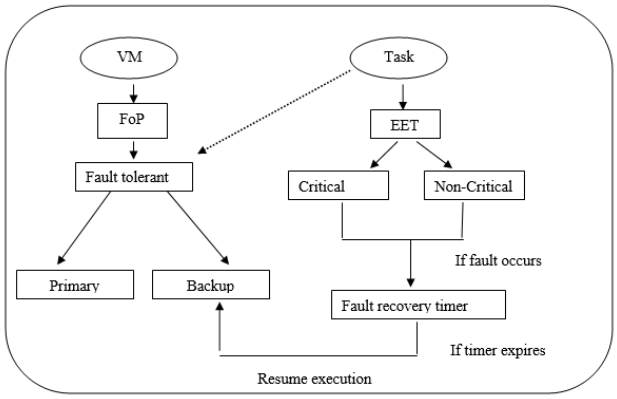

Fig. 1. Block Diagram of CHFTS Model.

The model represents the system structure, as seen in Figure 2. Processing capacity and cost per hour are particularly important for selecting VM(k) because of processing capacity and cost per hour, respectively (k). Users can access an endless number of VMs on cloud computing platforms that utilize virtualization. In addition, all VMs are hosted in one cloud data center to provide consistent bandwidth for all the VMs.

C. Fault Model

If a task is incomplete because of internal or external issues, it may be considered as a task failure.

Faults may occur during the execution of scientific work flows in cloud computing environments. There are various reasons behind these faults. The most common reason for task failure is the failure of corresponding VM. The other reasons for faults include unavailability of enough resources, resource overloading, delayed execution time etc.

There are two cases of faults that may occur in cloud, during execution. The first one is a permanent fault and other one is a temporary fault. In temporary faults, the faults may be recovered within short span of time whereas in permanent faults, the faults cannot be rectified until the failed element is repaired or replaced. Fail-signal or acceptance test is a fault detection mechanism, commonly used.

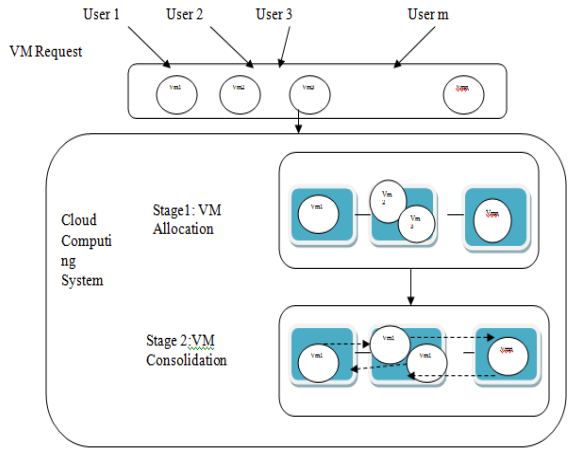

Fig. 2. System Model.

Due to task failures, the make span time of work flows will be increased and the service-level agreement (SLA) will be violated. Re-execution is one of the commonly used less expensive fault tolerant technique, which improves the reliability of workflows. Re-execution can be implemented in two methods: Spatial re-execution on other resources (SRE) and Temporal re-execution on the same resources, after fault recovery (TRE).

1) Estimation of FoP: The exponential probability density function for a specified mean time (M) between failures, is given by.

$$f(t) = \frac{1}{M} e^{-t/m} \tag{1}$$

Then, the probability of a failures, that occurs before time Δt for a VM is given by.

$$P(t \le \Delta t) = \int_{\Delta t}^{0} \frac{1}{M} e^{-t/m} dt = 1 - e^{-\Delta t/m} \tag{2}$$

Hence the probability of successful completion during the time Δ, is given by.

$$P(t > \Delta t) = 1 - P(t \le \Delta t) = e^{-\Delta t/m} \tag{3}$$

Let Tc be the computation time for a subtask executing on a VM and φ be the computation interval between two check points.

Then, the average number of attempts (NoA) required to finish Tc is given by.

$$NoA = \frac{Tc/\varphi}{P(t > \Delta t)} = \frac{Tc\, e^{\Delta t/M}}{\varphi} \tag{4}$$

The number of success is given by

$$\text{NoS} = \frac{T_c}{\varphi} \tag{5}$$

Hence the number of failures NoF, during Δt is given by

(ie) NoF = NoA – NoS (6)

$$\frac{T_c e^{\Delta t/M}}{\varphi} - \frac{T_c}{\varphi} = \frac{T_c}{\varphi}(e^{\Delta t/M} - 1) \tag{7}$$

Hence

$$\text{FoP} = \text{NoF} / T_c, = \frac{1}{\phi}(e^{\Delta t / M} - 1) \tag{8}$$

Note: It is assumed that only single failure occurs during one computation interval.

D. Estimation of Expected End Time (EET)

Let P(k) is considered as processing capacity of VM_k, k=,1,2...K.

Let $S(t_i)$ and $W(t_i)$ be the workload size of given input task t_i.

If and only if all the input data has been received from all prior tasks, then the task is started (ti).

start time of t_i is signified by Eq. (9).

$$T_{start}(t_i) = \max_{tj \in pre(ti)} \{T_{end}(t_j)\} \quad (9)$$

Note that if $t_i = t_{enrty}$, then $T_{start}(t_i) = 0$. Then, the transmission time is computed by,

$$T_{trans}(ti) = S(t_i) / BW \quad (10)$$

where *BW* is the bandwidth between two VMs.

Then, the execution time is given by,

$$T_{exec}(ti, VM(k)) = W(ti) / P(k) \quad (11)$$

Thus, the end time of task t_i is given by.

$$T_{end}(ti) = T_{start}(ti) + T_{trans}(ti) + T_{exec}(ti, VM(k)) \quad (12)$$

Hence the make span of workflow is given by.

$$makespan = T_{end}(t_{exit}) \quad (13)$$

E. Fault Tolerant Scheduling

Let {FVM} be the set of fault tolerant VMs with maximum FoP.

The type of each task is categorized into CRITICAL and NON-CRITICAL based on the values of EET. (ie) The tasks with shorter EET are considered as CRITICAL and others are considered as NON-CRITICAL.

For each task, a pair of VMs <VM_p, VM_b> from the k VMs, are allocated, where VM_p and VM_b are primary and backup VMs. If there is a fault occurs in VM_p, then a fault recovery timer (T_{FR}) will be started.

Case-1: If the task is a critical task, then T_{FR} is fixed such that.

T_{FR} < EET-FOT, where FOT is the time of fault occurrence.

Case-2: If the task is a non-critical task, then T_{FR} is fixed such that.

T_{FR} < (EET-FOT)* δ , where δ is a scaling factor ranges between (2, 4).

The total cost of the Hybrid Fault Tolerant Scheduling Model is calculated as.

$$CC = NoS * \min\left(NoA\right) + \frac{\text{TEnd} - \text{Texec}}{\text{count(VM)}} + \Delta t * \varphi \quad (14)$$

If the fault cannot be recovered within T_{FR}, (ie) If fault recovery time (FRT) is more than T_{FR}, then immediately, a notification will be sent to VM_b, which will resume the execution from that point.

Algorithm - Fault Tolerant Scheduling

Input : List of VMs and Tasks
Output: fault tolerant scheduling

foreach VMj
  Estimate FoP_j using (5)
  If FoP_j >= Max(FoP) , then
    Include FoP_j in {FVM}
  End if
End for
foreach task t_i
  Estimate EET_i using (11)
  if EET_i <= EET_{min}, **then**
    Type(t_i) = CRITICAL
  else
    Type(t_i) = NON-CRITICAL
  end if
end For
foreach incoming task t_j
  Allocate (VM_p, VM_b) $\in$ {FVM}
  if fault occurs at time FOT, **then**
    if Type(t_i) = CRITICAL, **then**
      Set T_{FR} < EET-FOT
    else
      Set T_{FR} < (EET-FOT)* δ
    end if
    if FRT<T_{FR}, **then**
      Resume t_i at FRT
    else
      Send notification to VM_b
      Resume t_i at VM_b
    end if
  end if
end For

Special outcomes of this approach

1) Since fault free VMs are selected, the chances of faults are less.

2) It does not result in huge storage since the VMs are allocated only when the fault is not recoverable (permanent fault) within a time span.

3) The time critical tasks will be executed within their deadlines since the tasks are migrated to backup VM, within the deadline.

4) The non critical tasks can be executed in the same VMs if the fault is recovered within tolerable time span (which is an even multiple of its deadline).

5) If the task could not be recovered beyond the tolerable time span, then only the non critical task are migrated to backup VMs, thereby reducing the storage overhead significantly.

6) Since the execution of tasks starts in the backup VM only from the time of fault, the time complexity will be less.

IV. Experimental Results

The proposed CEHFTS model is compared with the Spatial-Temporal Re-Execution method (SRE) [10]. The NASA workload [14] has been used as the emulator of Web users requests to the Access Point (AP). This workload represents realistic load deviations over a period time. It comprises 100960 user requests sent to the Web servers during a day. Table 1 shows the experimental parameters assigned in this work.

TABLE I. Experimental Parameters

Parameter	Value
Work load	NASA traces
Resource Utilization Thresholds	$U^{low-thr} = 20\% and U^{high\_thr} = 80\%$
Response Time Thresholds	$RT^{low-thr} = 200ms and RT^{high\_thr} = 1000ms$
Scaling Intervals	$\Delta t = 10min$
Desired Response Time	DRT = 1000ms=1s
Fault rate	1 to 2
Configuration of VMs	Medium and Large
Maximum On-demand VM Limitation	*Max*VM=10VM
Task and Resources Scheduling Policy	Time-Shared

A. Results for Critical Tasks

In this section, work flows with strict deadlines are considered (critical tasks) and allocated to VMs. The FoP of each task is increased in terms of fault rate. The performance metrics response delay, percentage of missed deadlines, CPU utilization and storage overhead of server are considered for evaluation.

Table 2 Performance Results of CEHFTS and SRE schemes for fault rate of critical tasks.

Figure 3 shows the response delay measured for the critical tasks when the fault rate is increased from 1 to 2. Since in the case of time bound critical tasks, backup VM migration is mostly supported, the response delay will be high. Hence, CEHFTS has the delay in the range of 1.4 to 2.8 seconds whereas SRE has delay in the range of 1.7 to 3.1 seconds. However, in case of CEHFTS, the failed tasks are resumed when the recovery time is less than EET. Hence it has 13% reduced response delay, when compared to SRE scheme.

TABLE II. Shows the Performance Results of both CEHFTS and SRE Schemes for Varying the Fault Rate of Critical Tasks

Fault rate (Critical)	Response Delay (sec)		% of Missed Deadlines		CPU Utilization (%)		Storage cost of Server	
	CEHFTS	SRE	CEHFTS	SRE	CEHFTS	SRE	CEHFTS	SRE
25	1.472	1.782	18	25	57	47	150	270
50	1.573	1.893	26	32	52	43.1	178	356
75	2.069	2.245	31	38	45.3	38.2	287	452
100	2.275	2.628	35	44	44.2	34	368	628

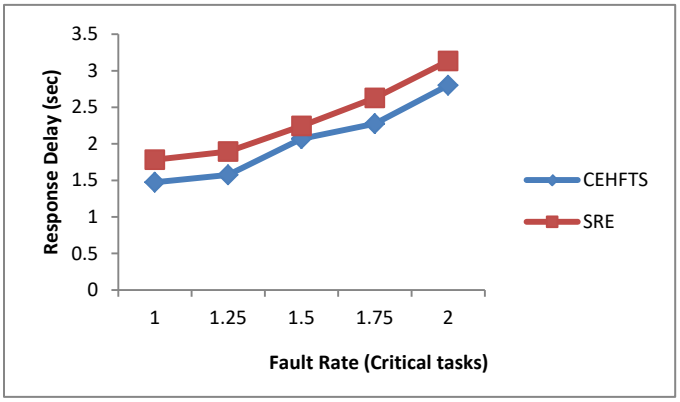

Fig. 3. Response Delay for Critical Tasks.

Figure 4 shows the missed deadlines for the critical tasks when the fault rate is increased from 1 to 2. Since in the case of time bound critical tasks, if the recovery time is higher than EET or if the fault is permanent fault, the chances of deadline miss are high. As depicted from the figure, CEHFTS has the missed deadlines in the range of 18% to 41% whereas SRE has missed deadlines in the range of 25% to 47%. However, in case of CEHFTS, the failed tasks are resumed when the recovery time is less than EET. Hence it has 19% reduced missed deadlines, when compared to SRE scheme.

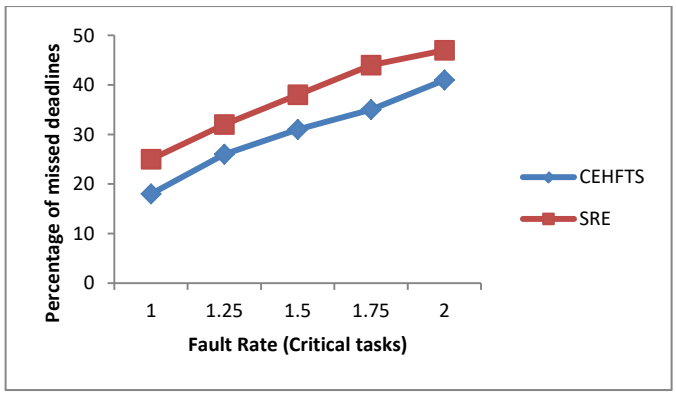

Fig. 4. Missed Deadlines for Critical Tasks.

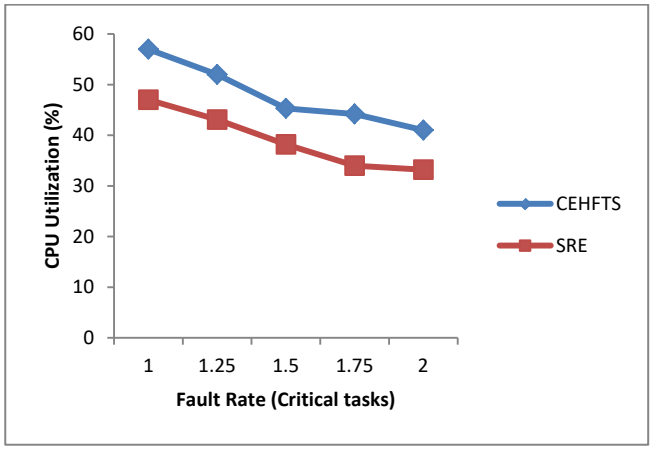

Fig. 5. CPU Utilization (%) for Critical Tasks.

Figure 5 shows the CPU utilization for the critical tasks when the fault rate is increased from 1 to 2. In case of permanent failures, the CPU utilization of primary VMs will be affected for the critical tasks. As it can be seen, from the figure, the utilization of CEHFTS reduces from 57% to 41% and the utilization of SRE reduces from 47% to 33%. However, CEHFTS has 18% higher CPU utilization than SRE scheme, since it minimizes the use of backup VMs.

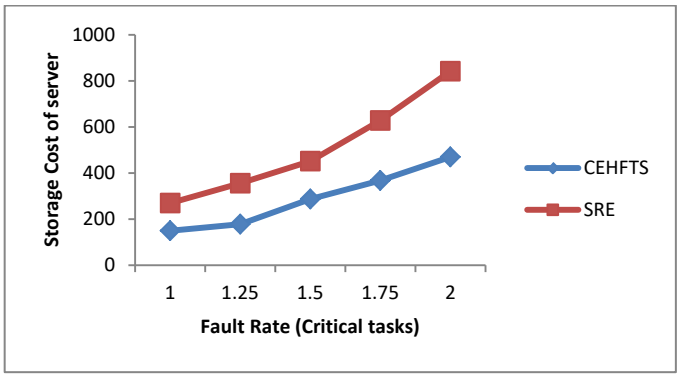

Fig. 6. Storage Cost for Critical Tasks.

Figure 6 shows the storage cost of servers in case of critical tasks. Since in case of time bound critical tasks, if the recovery time is higher than EET or if the fault is permanent fault, the tasks are migrated to backup VMs, leading to increased storage cost. As depicted from the figure, CEHFTS has cost in the range of 150 to 470 whereas SRE has cost in the range of 270 to 840. However, in case of CEHFTS, the failed critical tasks are resumed when the recovery time is less than EET. Hence it has 43% reduced storage cost, when compared to SRE scheme.

B. Results for Non-Critical Tasks

In this section, work flows with elastic (flexible) deadlines are considered (critical tasks) and allocated to VMs. The FoP of each task is increased in terms of fault rate. The performance metrics response delay, CPU utilization and storage overhead of server are considered for evaluation.

Table 3 Performance Results of CEHFTS and SRE schemes for fault rate of non-critical tasks

TABLE III. SHOWS THE PERFORMANCE RESULTS OF BOTH CEHFTS AND SRE SCHEMES FOR VARYING THE FAULT RATE OF CRITICAL TASKS

Fault rate (Non-critical)	Response Delay (sec)		CPU Utilization (%)		Storage cost of Server	
	CEHFTS	SRE	CEHFTS	SRE	CEHFTS	SRE
25	2.672	3.582	52	45	120	165
50	3.573	4.813	46	42.7	138	228
75	5.069	7.145	44.2	36.4	187	280
100	5.755	7.628	41.5	33.1	228	355

Figure 7 shows the response delay measured for the non-critical tasks when the fault rate is increased from 1 to 2. Since in case of non-critical tasks, the fault response timer is set as high, the response delay will be higher than that of critical tasks. Hence, CEHFTS has the delay in the range of 2.6 to 6.4 seconds whereas SRE has delay in the range of 3.5 to 8.3 seconds. However, in case of CEHFTS, the failed tasks are resumed when the recovery time is less than EET. Hence it has 25% reduced response delay, when compared to SRE scheme.

Figure 8 shows the CPU utilization for the non-critical tasks when the fault rate is increased from 1 to 2. In case of permanent failures, the CPU utilization of primary VMs will be affected for the non-critical tasks also. As it can be seen, from the figure, the utilization of CEHFTS reduces from 52% to 40% and the utilization of SRE reduces from 45% to 30%. However, CEHFTS has 16% higher CPU utilization than SRE scheme, since it minimizes the use of backup VMs.

Figure 9 shows the storage cost of servers in case of non-critical tasks. Since, if the fault is permanent fault, the tasks are migrated to backup VMs, leading to increased storage overhead. As depicted from the figure, CEHFTS has cost in the range of 120 to 270 whereas SRE has cost in the range of 165 to 472. However, in case of CEHFTS, the failed non-critical tasks are resumed when the recovery time is less than EET. Hence it has 35% reduced storage cost, when compared to SRE scheme.

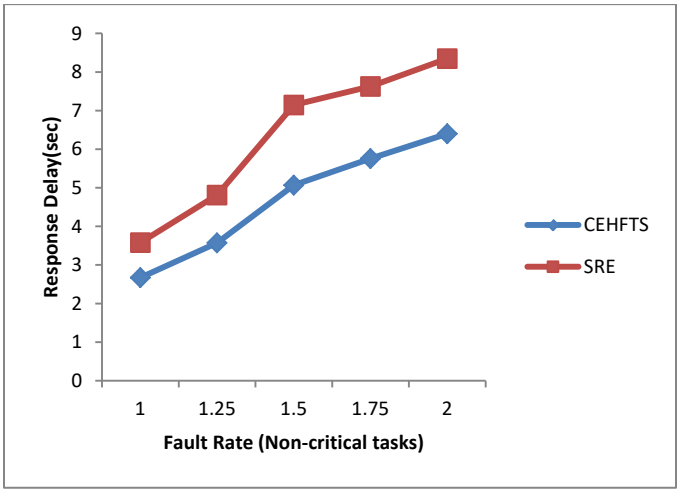

Fig. 7. Response Delay for Non-critical Tasks.

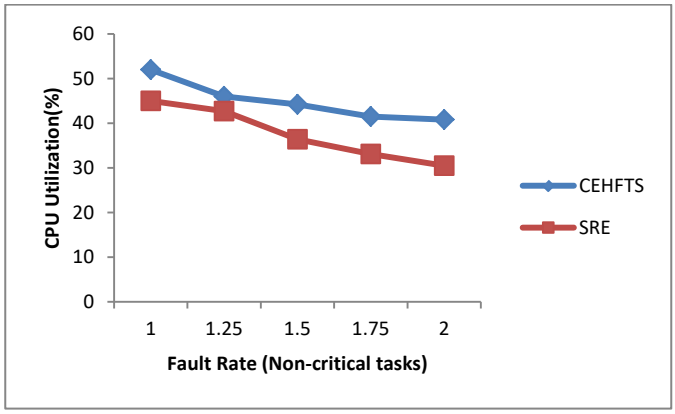

Fig. 8. CPU Utilization (%) for Non-critical Tasks.

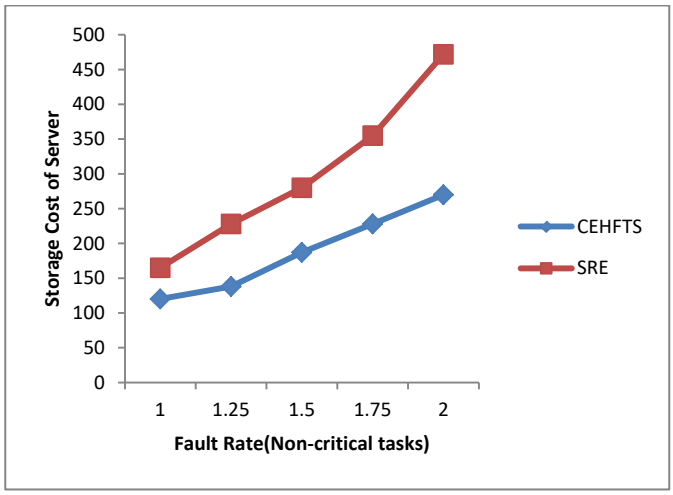

Fig. 9. Storage Overhead for Non-critical Tasks.

V. Conclusion

In this research, a new hybrid fault tolerant scheduling approach, called Cost Effective Hybrid Fault Tolerant Scheduling (CEHFTS), is proposed for cloud computing. For this model, the prior failures and successful executions are used to estimate the VM failure probability (FoP). Estimated anticipated execution time (EET) is calculated for each task. At this point, the projects with deadlines of less than one week are classed as high-priority activities. An adaptive fault recovery timer is kept running until a fault occurs, which is then increased as a function of the various fault types. Experiments are conducted for critical and non-critical tasks by varying the fault rate. Experimental results have shown that CEHFTS model achieves 43% reduced storage cost and 13% response delay for critical tasks, when compared to existing technique. Future work focus on grouping the tasks based on its type and requirements so that the CPU utilization and battery power can be further reduced. The task deadline and time scheduling can still be improved to enhance the performance levels.

References

[1] Priti Kumari and Parmeet Kaur," A survey of fault tolerance in cloud computing", Journal of King Saud University – Computer and Information Sciences, Elsevier, 2018.

[2] Vinay K and S M Dilip Kumar," Fault-Tolerant Scheduling for Scientific Workflows in Cloud Environments", IEEE 7th International Advance Computing Conference (IACC) , 2017.

[3] J.Soniya,J.Angela Jennifa Sujana and T.Revathi, "Dynamic Fault Tolerant Scheduling Mechanism for Real Time Tasks in Cloud Computing", IEEE International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT),2016.

[4] Pengze Guo and Zhi Xue, "Real-Time Fault-Tolerant Scheduling Algorithm with Rearrangement in Cloud Systems", IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC),2017.

[5] Sreelekshmi S and K R Remesh Babu, "Synchronized Multi-Load Balancer with Fault Tolerance in Cloud", International Journal of Computer Information Systems and Industrial Management Applications,Vol-10,pp:107-114,2018.

[6] Zhongjin Li, Jiacheng Yu, Haiyang Hu, Jie Chen, Hua Hu, Jidong Ge and Victor Chang, "Fault-Tolerant Scheduling for Scientific Workflow with Task Replication Method in Cloud", IEEE 3rd International Conference on Internet of Things, Big Data and Security,2018.

[7] Diptee H. Devmurari and Kashyap Raiyani, "Resource Reliability using Fault Tolerance aware Scheduling in Cloud", International Journal of Innovative Research in Computer and Communication Engineering, Vol-5,No-4,2017.

[8] Hussein El Ghor, Julia Hage, Nizar Hamadeh And Rafic Hage Chehade, "Energy-Efficient Real-Time Scheduling Algorithm For Fault-Tolerant Autonomous Systems", Scalable Computing: Practice and Experience, Volume 19, Number 4, pp. 387–400,2018.

[9] Hao Wu , Qinggeng Jin ,Chenghua Zhang and He Guo, "A Selective Mirrored Task Based Fault Tolerance Mechanism for Big Data Application Using Cloud", Wireless Communications and Mobile Computing, Hindawi, Volume 2019,pp:1-12,2019.

[10] Na Wu , Decheng Zuo and Zhan Zhang, "Dynamic Fault-Tolerant Workflow Scheduling with Hybrid Spatial-Temporal Re-Execution in Clouds", Information, MDPI, Vol-10,No-169,2019.

[11] Ji Wang, Weidong Bao, Xiaomin Zhu, Laurence T. Yang and Yang Xiang, "FESTAL: Fault-Tolerant Elastic Scheduling Algorithm for Real-Time Tasks in Virtualized Clouds", IEEE Transactions on Computers, Volume: 64 , Issue: 9, Sept. 1 2015.

[12] Punit Gupta and S. P. Ghrera," Power and Fault Aware Reliable Resource Allocation for Cloud Infrastructure", International Conference on Information Security & Privacy (ICISP2015), Elsevier, December 2015, Nagpur, INDIA.

[13] Sheng Di, Cho-Li Wang, "Error-Tolerant Resource Allocation and Payment Minimization for Cloud System", IEEE Transactions On Parallel And Distributed Systems, Vol. 24, No. 6, June 2013, pp 1097-1106, DOI 10.1109/TPDS.2012.309.

[14] M. F. Arlitt and C. L. Williamson, "Internet web servers: Workload characterization and performance implications," IEEE/ACM Transactions on Networking (ToN), vol. 5, pp. 631-645, 1997.

[15] Y.L. YANG, X.G. PENG, J.F. CAO,"Trust-Based Scheduling Strategy for Cloud Workflow Applications", INFORMATICA, 2015, Vol. 26, No. 1, 159–180.

[16] Wei Wang, Guosun Zeng, Junqi Zhang and Daizhong Tang,"Dynamic trust evaluation and scheduling framework for cloud computing", Security and Communication Networks,2012, 5,311–318.

[17] G. Jeeva Rathanam and A. Rajaram,"Trust Based Meta-Heuristics Workflow Scheduling in Cloud Service Environment", Circuits and Systems, 2016, 7, 520-531.