Fault tolerant workflow scheduling based on replication and resubmission of tasks in Cloud Computing

Jayadivya S K*
Department of Computer Science & Engineering
National Institute of Technology
Tiruchirappalli, Tamil Nadu, India
jayadivyask@gmail.com

Jaya Nirmala S
Department of Computer Science & Engineering
National Institute of Technology
Tiruchirappalli, Tamil Nadu, India
sjaya@nitt.edu

Mary Saira Bhanu S
Department of Computer Science & Engineering
National Institute of Technology
Tiruchirappalli, Tamil Nadu, India
msb@nitt.edu

***Abstract*—The aim of workflow scheduling system is to schedule the workflows within the user given deadline to achieve a good success rate. Workflow is a set of tasks processed in a predefined order based on its data and control dependency. Scheduling these workflows in a computing environment, like cloud environment, is an NP-Complete problem and it becomes more challenging when failures of tasks are considered. To overcome these failures, the workflow scheduling system should be fault tolerant. In this paper, the proposed Fault Tolerant Workflow Scheduling algorithm (FTWS) provides fault tolerance by using replication and resubmission of tasks based on priority of the tasks. The replication of tasks depends on a heuristic metric which is calculated by finding the tradeoff between the replication factor and resubmission factor. The heuristic metric is considered because replication alone may lead to resource wastage and resubmission alone may increase makespan. Tasks are prioritized based on the criticality of the task which is calculated by using parameters like out degree, earliest deadline and high resubmission impact. Priority helps in meeting the deadline of a task and thereby reducing wastage of resources. FTWS schedules workflows within a deadline even in the presence of failures without using any history of information. The experiments were conducted in a simulated cloud environment by scheduling workflows in the presence of failures which are generated randomly. The experimental results of the proposed work demonstrate the effective success rate in-spite of various failures**.

Keywords-Cloud computing; Scheduling; Workflows; Replication; Resubmission; Fault tolerance;

I. INTRODUCTION

Cloud computing [1][2] has emerged as a global – infrastructure for applications by providing large scale services through cloud servers. The services can be either storage service or computation service. These services can be configured dynamically by making use of virtualization. Any application in cloud computing environment can be represented by a workflow. However, this computing environment still cannot deliver the quality, robustness and reliability that are needed for the execution of various workflows because of different failures like link failure, failure of server providing the service, malicious code in the executing node, datasets required by the task may be locked by other tasks etc [3]. The scheduling system in a cloud should overcome such failures which can be provided by fault tolerant scheduling algorithms.

Workflow is a sequence of tasks processed in a specific order based on data or control dependency between these tasks. A workflow has a set of parameters and it is represented as a Directed Acyclic Graph (DAG) [4] in which the nodes represent individual application tasks and directed arcs stand for precedence relationship among

the tasks. Mapping between the tasks and services depends on the deadline of a workflow and the computation power of the resources available. Execution of these workflows can last for hours, days or even weeks and hence the realization of workflow failure at the end may lead to missing of a deadline. The workflow execution system with low tolerance for failures may lead to a situation that user may not realize the failure of workflow there by tasks may miss their deadline. Hence the workflow scheduling system should exhibit high levels of tolerance for failures.

Scheduling of workflows within the deadline is a challenging issue when fault tolerance is considered. There are many fault-tolerant scheduling algorithms which make use of tasks replication and task resubmission [5] and few algorithms make use of history of resource information to find failure probability [6]. Each method has its own disadvantages and advantages which are discussed in section II.

In this paper, fault tolerance is achieved by compromising task replication and task resubmission methods. Tasks are replicated based on heuristic metric and priority of the task. Heuristic metric is calculated by finding the tradeoff between replication and resubmission factor which gives the replication number based on impact of the resubmission [7]. The heuristic metric is considered because replication alone may lead to resource wastage and resubmission alone may increase makespan. Tasks are prioritized based on out degree, earliest deadline and high resubmission impact [8]. Tasks are scheduled to meet deadline by considering priority and thereby to reduce wastage of resources by restricting unnecessary replication of tasks.

The rest of this paper is structured as follows: The related work is explained in section 2. Section 3 discusses about the proposed strategy. Section 4 describes about experimental results and section 5 gives the conclusion and future work.

II. RELATED WORK

In this section, few fault tolerance techniques used in scheduling of workflows are discussed. As for cloud workflow systems, similar to many other grid and distributed workflow systems, scheduling is a very important component which determines the performance of a whole system. These workflow scheduling systems should be fault tolerant for the failures that occur in the computing environment.

Fault tolerant scheduling algorithms can be categorized based on check pointing, traces of data, replication of tasks and resubmission of tasks. Each category has its own advantages and disadvantages.

Fault tolerant workflow scheduling is provided by making use of failure probability which is considered in algorithms [9][10][11]. Zhang et al. [9] described an approach for combined fault tolerance and scheduling workflow applications in computational grids. Kandaswamy et al. [10] described a mechanism for fault tolerant workflow by considering check pointing, migration, and over-provisioning. Liang et al. [11] developed a failure prediction model based on failure analysis of BlueGene/L system. Analysis of failure probability requires traces of failure data about each resource in the environment but often cloud providers do not reveal about their infrastructure and most of the time, this information is hidden from the user.

Methods like replication and resubmission of tasks do not require any history of information. Few techniques provide fault tolerance by making use of replication. All tasks are replicated to their maximum count which provides very good fault tolerance but uses lot of resources. The tasks which can be executed on highly reliable resources are also replicated and hence the resources are wasted. If there is enough number of resources available then this method will give good fault tolerance power to the workflow scheduling system. Most of the times, number of tasks will be very high when compared to the number of resources available and hence this method may lead to task serialization instead of parallel execution [12].

Another technique provides fault tolerance by using resubmission of tasks. Here, tasks are resubmitted to the same or different resource after its failure declaration which will not waste any resource but increases the makespan of a workflow. This may also lead to missing deadline of a workflow and hence gives the less success rate of scheduling when deadline is considered.

To overcome the disadvantages of replication and resubmission, few techniques are discussed which finds the tradeoff between the replication and the resubmission [7]. Tasks are replicated without considering any other parameters except the heuristic metric which may replicate all the tasks even if they are not critical and may lead to resource wastage. Hence, in this paper, prioritization of tasks is considered along with heuristic metric. The tradeoff between replication and resubmission factor helps to find the heuristic metric that indicates the replication count for each task. Priority for the tasks is provided by using parameters like out degree, earliest deadline and high resubmission impact. Prioritization of tasks helps to meet the deadline and reduces resource wastage along with providing fault tolerance for the workflow system.

Here, the objective is to schedule the tasks within its deadline by tolerating the faults that may be present in a cloud computing environment to provide good success rate of scheduling.

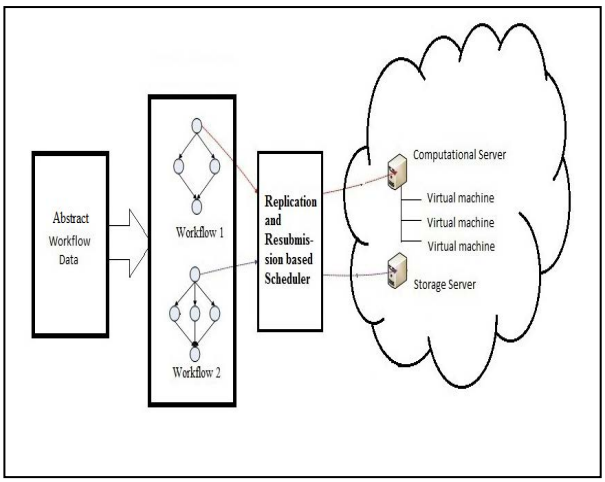

Figure 1. Architecture of FTWS

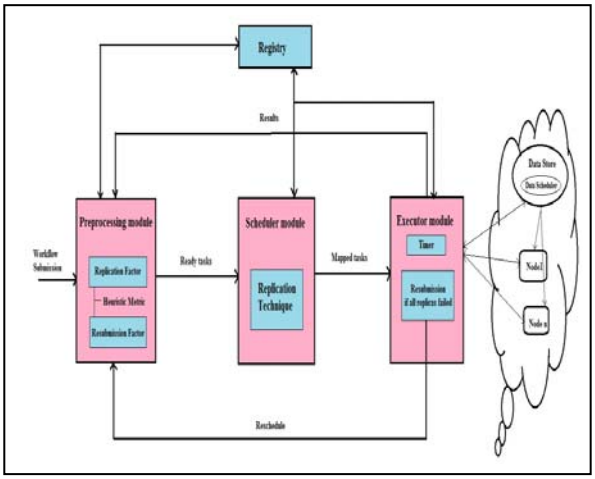

Figure 2. Control flow diagram of modules

III. ARCHITECTURE AND METHODOLOGY

A. *Overview of FTWS*

Fig. 1 shows the architecture of the FTWS in a cloud environment. There are two major types of servers in cloud which are storage server and computational server. Storage server provides the services related to data storage and modification which does not require any mapping of services. Computational server provides the service related to computing resources which requires mapping of services to a task.

FTWS process is designed with four major modules which are Preprocessor Module (PM), Replication based Scheduler Module (RSM), Executor Module (ResEM) with Rescheduling if required and Data Scheduler (DS). The control flow diagram of these modules is shown in Fig. 2.

FTWS replicates and schedules the tasks to meet the deadline of a workflow by using replication and resubmission of tasks. Replication of tasks is performed in the scheduling phase where as resubmission is performed in the execution phase.

Users first submit their workflows with deadline, replication factor and resubmission factor in the form of Abstract data structure format to PM. The PM discovers the services required for those tasks and generates the DAG [3] based on the data and control dependencies between them and divides the tasks based on computation services and storage services. It also generates the threshold which helps in prioritizing the tasks and heuristic metric which indicates replication count. The RSM replicates the tasks based on the priority and heuristic metric. It allocates the particular services to these tasks based on the QWS (QoS based Workflow Scheduling) algorithm [13] in cloud environment. After mapping, ResEM sends the tasks to the mapped servers and starts the timer based on expected execution time. If ResEM receives the successful output from the server within the timer expiry then it activates all the tasks which are dependent on the task. If it fails to receive successful output, then ResEM waits for other replicas. If all replicas fail then it resubmits the task. The DS manages the datasets by replicating them on different sites for the execution of replicated tasks. Detailed explanations of these modules are explained in Section C.

B. *Problem Definition*

Workflow ω_i is represented by a set of four tuples $<T_{i,j}, D_i, Rep_i, Res_i>$ where $T_{i,j}$ is a set of finite tasks $\{ T_{i,1}, T_{i,2}, T_{i,3},\ldots T_{i,j} \}$, D_i is the Deadline of the workflow ω_i before that the workflow has to be executed, Rep_i indicates the replication count of a task in a workflow ω_i and Res_i is the maximum resubmission count of a workflow ω_i.

Each task $T_{i,j}$ has a set of attributes like task-id, deadline, execution time, datasets and services needed, size, etc. Deadline of each task is calculated by distributing the deadline of workflow among tasks in a critical path. Most of the times the execution time of the task depends on the performance of the machine in which job has to be executed [14]. For this reason, execution time is calculated when it is assigned to a node based on its MIPS rate.

Let m be the number of services available in cloud and s_k is the set of services which are capable of executing the task $T_{i,j}$. The FTWS replicates and schedules a set of tasks by mapping each task to suitable s_k by executing tasks within the deadline as in QWS algorithm with few changes required to handle replication. It also reduces the makespan required to execute a workflow. Makespan is calculated as shown in (1).

$$makespan = \max_{Pj \in \tau_n} executiontime(P_j) \quad (1)$$

where τ_n *is the critical paths in the* ω_i

C. Modules Description

1) Preprocessing Module (PM): The main functionality of this module is to calculate all the parameters like heuristic metric, threshold, deadline of each task in a workflow etc., which are required in the process of scheduling workflows.

The PM accepts the data required for the workflow from the user in the form of an abstract data structure template as shown in the following example.

No.of workflows: 1
No.of tasks: 4
*task*1*:{ Datasets:d*1,*d*2,*d*3*, Services:s*1,*s*2 *ctrl_on: Null }*
*task*2*:{ Datasets:d*2,*d*4,*d*5*, Services:s*3,*s*4 *ctrl_on: Null }*
*task*3*:{ datasets:d*3,*d*7,*d*1*, services:s*10,*s*5 *ctrl_on:task*2 *}*
*task*4*:{ datasets:d*5,*d*6,*d*10*, services:s*7,*s*12 *ctrl_on: Null }*
Deadline:Sun May 4 12:53:18 2012
replication factor : 6
resubmission factor : 6

DAG is a directed set of arcs of the form (T_i,T_j) where T_i is called the parent task and T_j is called the child task of T_i. Child task cannot be executed until all its parent tasks complete its execution. PM generates the DAG by finding the data and control dependencies (*Dep* (T_i,T_j)) between the tasks of a workflow using the expression shown below.

$$Dep(T_i,T_j) => (datasets(T_i) \cap datasets(T_j) != \phi) \;||\; (Task\_id(T_i) == ctrl\_on(T_j))\;?\;1:0;$$

It is essential that each task should be completed before a deadline so that the workflow meets the deadline D_i given by the user. So the deadline of the whole workflow is distributed in to sub-deadlines among the tasks in a critical path based on their size[15]. Backtracking algorithm as shown below is used to find the critical paths in a workflow.

```
void backtrack( )
{
      get DAG of workflow
      critical_path(DAG)
}

void critical_path(DAG)
{
       /*start –> root node
          end –>terminating node in DAG*/
       for(each set of start and end tasks in ω_i)
             allpaths(DAG,start,end);
}
```

```
void allpaths(DAG,current,end)
{
        /* current –>node from which path has to be found */
        push(current);
        if (current==end)
                store each task in critical path set τ of ω_i;
        for (i from 0 to number of neibours of i)
                allpaths(DAG,i,end);
}
```

The PM finds the threshold which is used as one of the parameter to prioritize the tasks during replication in scheduling phase. The threshold (T_h) is calculated by finding the average number of children for a task in a workflow as shown in (2).

$$T_h = \sum_{i=1}^{n} child\ (t_i)\ /\ n \qquad (2)$$

Where n indicates number of workflows

The heuristic metric is developed by using the algorithm shown below. Here the make span of a workflow is calculated by making use of a QWS algorithm which was designed in our previous work [13]. In the heuristic metric calculation, workflow ω data is copied to workflow $\omega^{/}$. Number of instructions required to execute a task in workflow $\omega^{/}$ is multiplied with resubmission factor which gives the worst case number of instructions to execute a particular task. Then, the difference δ_i between the makespan of ω and $\omega^{/}$ is calculated which helps in analyzing the impact of resubmission on a workflow with respect to one particular task T_i. Resubmission Impact (RI) is calculated by normalizing the difference. RI is one of the parameter which helps in prioritizing tasks for replication. The heuristic metric is calculated by multiplying the RI fraction with the replication factor mentioned by user for each workflow which gives the replication count for each task in a workflow ω.

$$\omega^{/} = \omega$$

$$\omega^{/} \Leftarrow \forall T_i,\ \eta(T_i) * res\_factor$$

$$\delta_i = makespan(\omega^{/}) - makespan(\omega)$$

normalize RI_i

$$RI_i = \frac{\delta_i}{\max_{j=0..n} \delta_j}$$

$$rep_i = RI_i * rep\_factor$$

New rep_i *is given to* ω

Tasks can be divided into dependent tasks and independent tasks. Tasks which are independent from all other tasks are said to be ready tasks. PM finds ready tasks using (3) in all workflows which are independent tasks whose predecessor tasks executed successfully or they are the start tasks of workflows i.e., root nodes of all DAGs and it sends these tasks to the scheduler module by placing them in to a ready queue.

$$\sum_{k=0}^{n} \sum_{l=0}^{k} DAG_i(T_k, T_l) == 0$$

then T_k *is ready task* (3)

where n is total number tasks in a workflow

2) Replication based Scheduling module(RSM): This module sorts all tasks in the ready queue as in QWS algorithm based on:

- Instructions_time_ratio
- Number of services

Instructions_time_ratio is the ratio between the number of instructions in the task and the deadline of the task. The tasks with less instructions_time_ratio are scheduled first.

If a task requires more number of services, it may block the tasks which require fewer services thereby increasing the waiting time of these tasks. So the task with fewer services is scheduled first.

After sorting the tasks in ready queue, it checks whether the task requires computation service or storage service. If the task is related to storage service, then it maps the task to the storage server. If it is related to the computational service, then it replicates the task T_i as shown in following snippet.

$$
\begin{aligned}
&if((no.\ of\ children\ (T_{i,j}) \\
&\qquad >= Threshold(\omega_i)\ \&\&\ all\ children\ of\ T_{i,j}\ are\ feasible) \\
&\quad || (\text{Re}submission\_impact\ \ for\ T_{i,j} > 0.9) \\
&\quad || (deadline\ (T_{i,j}) - current\_time \\
&\qquad\qquad - estimated\_execution\_time(T_{i,j}) \\
&\qquad\qquad <= \text{exp}ected\_commn\_time)) \\
&then \\
&\qquad replicate\ \ task\ \ based\ \ on \\
&\qquad\qquad the\ \ heuristic\ \ metric\ \ value \\
&endif
\end{aligned}
$$

Here the condition for replication is based on task priority which is calculated by any of the following three factors.

- No. of children
- Resubmission Impact
- Deadline

If the number of children of T_i is more than the threshold (T_h), then the task becomes critical because number of tasks waiting on its result is more. Since waiting time for children increases due to the failure, T_i is replicated. If the children of T_i depends on the other tasks in addition to T_i, then the replication of T_i only wastes the resources hence T_i will not be replicated.

Resubmission impact describes the overhead involved in the scheduling process if the task T_i fails. Hence if this RI factor is more than 90% then the task T_i is replicated.

Even in failure of above two conditions, if the expected completion time of task T_i is very close to the deadline then task T_i is replicated.

After replication, the replicated tasks are placed next to the original task in the ready queue. Here the tasks are adjusted based on the services required by the replicated tasks (*Services(T_{rep})*) and other tasks (*Services(T_{other})*) in the queue.

$$
\begin{aligned}
&Services(T_{rep}) \cap Services(T_{other}) == \varphi\ ?\ no\ adjustement\ : \\
&\qquad number\ (aviable\ \ resources\ ) > number\ (T_{rep} + T_{other})\ ? \\
&\qquad\qquad no\ \ adjustment\ \ :\ \ adjustment\ \ required\ ;
\end{aligned}
$$

This adjustment is performed because if the available services are able to execute only replicated tasks, then all other tasks may get delayed because of replication which may lead to miss deadline. This also helps in avoiding serialization. If the adjustment is required, then tasks in ready queue are checked against deadline and readjusted based on earliest deadline first. This rearrangement may reduce the number of replications but ensures meeting deadline without increasing the makespan.

The tasks in ready queue are mapped with the available services in a data center by using the information present in the registry. If the services available in a datacenter are busy then it checks for other datacenters and assigns to the next free server. If all are busy then it will be mapped to the wait queue of the data center which has lesser load compared to others. The algorithm for mapping and scheduling tasks on services is shown below:

```
void schedule(t,ω_i )
{
    /* t –> task
       s –>service*/
    while ( ready_queue not empty )
    {
        t← first task in ready_queue
        s← getservice(t)
        send task to the data center by placing in run queue
        dequeue( ready_queue)
        make s as busy
    }
}
int getservice(t)
{
    select service S such that
        executiontime(t)< deadline(t)
    return S
}
```

All available data centers will be registered in the registry with the attributes MIPS rate, available memory, services it can provide, etc.

3) Resubmission based Executor module: ResEM sends all mapped tasks to the respective data centers and also waits for the acceptance and reply from the data center. The data center can accept the task or reject the task based on surplus information as shown in (4).

$$T(t_{ready\_queue} + t_{wait\_queue} + t_i + resolve(t_i)) <= Deadline(t_i) \quad (4)$$

Where t_{ready_queue} –> time required to execute tasks in ready queue

t_{wait_queue} –> time required to execute tasks in wait queue

t_i –> time required to execute the incoming task

$resolve(t_i)$ –> time required to get datasets used by task t_i

If the datacenter accepts the task, then ResEM sends the task to the data center by assinging a unique version identity to the task and then waits for the result. While sending the task it also starts a timer by spawning the thread with the expected time to execute the task as shown below.

```
pthread_create ( &timer_var, NULL,
        timer, &lc_expected_etime );
```

ResEM waits till the timer expiry or till the result comes back. Once ResEM gets the result, it checks for the correctness of a task by checking the given version id and the obtained version id. If the completed task is correct then, it sends the signal to the node where the task was executed to update the datasets in data store and also to stop all the other replicas. ResEM sends the result to preprocessor which masks the dependencies of all its child tasks.

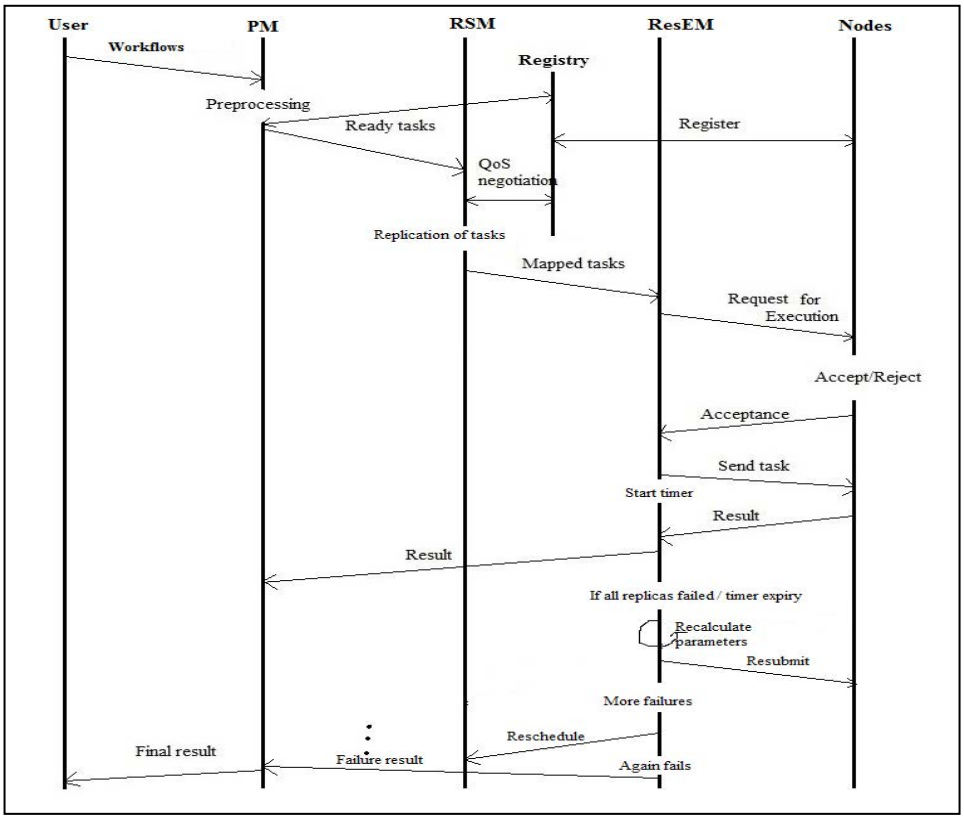

Figure 3. Process of FTWS

If the completed task does not meet the requirements or timer expired before getting the result, then ResEM waits for the result of other replicas. If all the replicas fail then it resubmits the task by calculating its parameters again and the task is scheduled on the same node with a new set of parameters. If the task fails again then it is resubmitted till it reaches maximum failure factor. Failure factor is a variable and in this work it is assumed to be 75% of the maximum number of resubmission count. If the failure of the task had reached the failure factor, it is considered as a new task and re-scheduled on a different node. If the task fails and reaches the maximum resubmission count on the new node then ResEM sends an error signal to preprocessor which displays message to the user.

The complete process of the FTWS is shown in Fig. 3. This shows how the tasks can move from initiation state to completion state.

4) Data Scheduler: Tasks may be replicated in different nodes so all those nodes may require accessing the same set of datasets. Hence the data sets are also replicated. The management of these data sets is done by the DS. The strategy followed here is to maintain two copies of datasets. They are primary copy and local copy. After getting the datasets from the data store to the node, the copy of datasets in node becomes a local copy. All the changes done by the node are modified locally. Once the signal comes from the ResEM, the updated datasets are sent to the data store and also DS invalidates all the local copies in other nodes. DS maintains a table to keep track of all the replica information which helps in invalidating the datasets. The process of data replication and updation of datasets is shown in Fig. 4.

IV. EXPERIMENTAL RESULTS

In this section, the experimental results of the proposed FTWS model are discussed. The Cloud environment is simulated using VMware virtual machines. Communication between them is achieved by using SSH programming. Different types of faults that can occur in a cloud environment are also simulated.

There are around 25 services in cloud environment which are scattered in different computational data centers and storage servers. To evaluate the performance of multiple workflows ranging from 5 to 50 with a minimum of 8 tasks in each workflow with different set of parameters are generated randomly using random generation algorithm with uniform distribution.

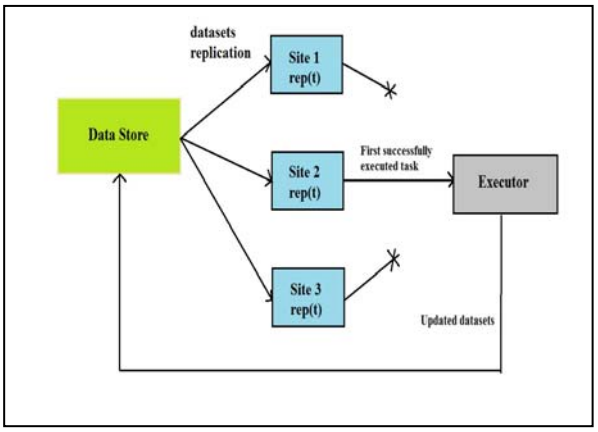

Figure 4. Datasets Maintenance

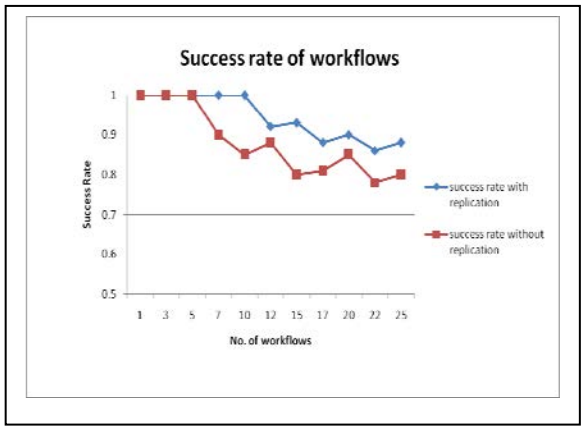

Figure 5. Success Rate of Scheduling

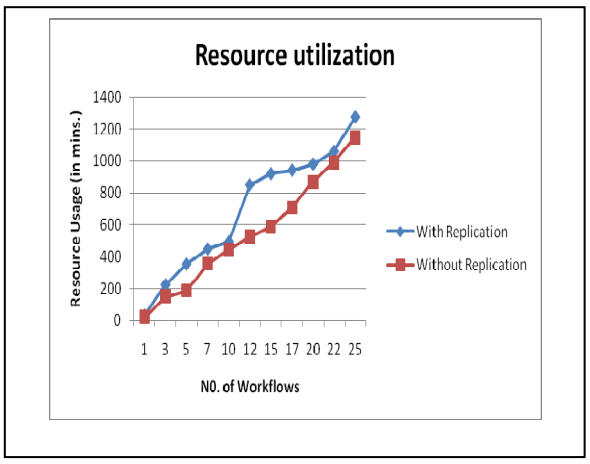

Figure 6. Resource Utilization

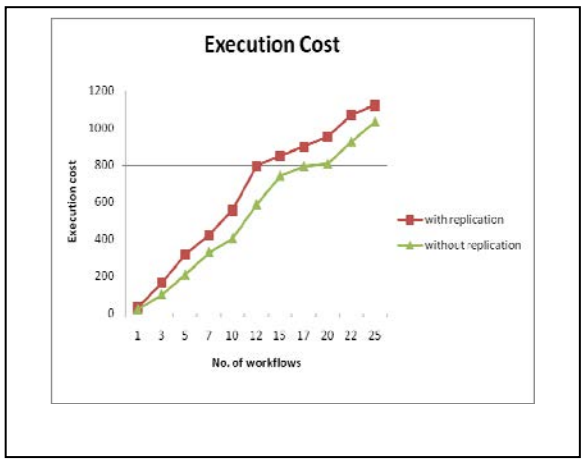

Figure 7. Cost Comparison

The most common method for generating the random sequence $\{r_1, r_2, r_3, \ldots\ldots, r_k\}$ over the [n, m] is the linear congruent method. This method multiplies the previous random number r_i-1 by the constant 'n' and adding with constant 'c' to it, then the modulus of the result is taken by dividing it by 'm' which gives r_i as shown in (5). This helps in distributing the values over [n, m] uniformly. '1' is added in the expression (5) to avoid generation of zero.

$$r_i = (nr_{i-1} + c) \bmod m + 1 \quad (5)$$

The services are chosen for each task randomly in the set $\{S_1, S_2, S_3, \ldots\ldots, S_{25}\}$. Then this information is sent to PM which starts the process of FTWS as explained in section III.

Various faults generated in nodes are link failure, malicious code attack, deleting results, request loss, no route to data store etc. Randomly we generate the MTBF (Mean Time Between Failures) and MTTR (Mean Time To Repair) values in each node. The timer thread is started using the value of MTBF. After the expiry of MTBF thread, new thread with MTTR is started and till the expiry of MTTR thread different faults are generated like deleting the results generated by the node, deleting the request, changing the execution time to maximum or minimum or deleting the communication address for the data store etc. After the MTTR timer expiry i.e., after repair it activates the MTBF thread, the node works properly till expiry of MTBF thread and the cycle continues.

The proposed FTWS algorithm was run to evaluate its performance for various test cases with different number of workflows with different deadlines and with various types of faults. The experiments were repeated with different replication and resubmission factors and averages of different results were found.

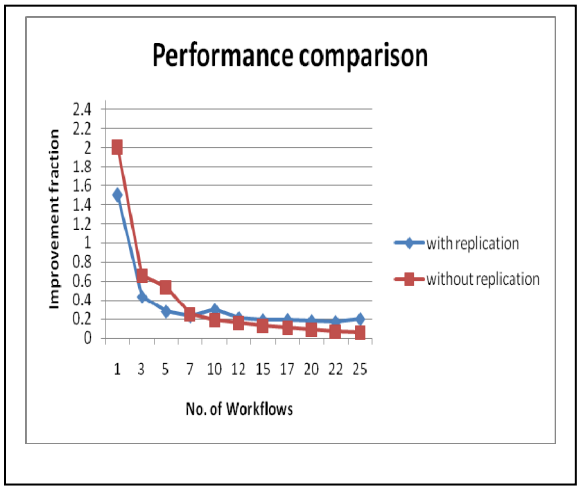

Figure 8. Performance Comparison

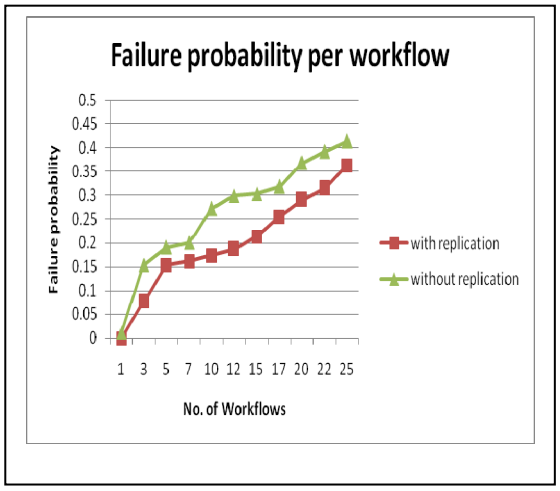

Figure 9. Failure Probability

The graph for comparison between success rate with replication and success rate without replication is shown in Fig. 5. Since the tasks are replicated on different resources, the usage of resources may be more compared to scheduling without replication. Fig. 6 shows the graph for resource usage. Cost may also increase because of replication which is showed in Fig. 7. Fig. 8 shows the ratio between success rate and the resource usage and also the failure probability is compared between scheduling with replication and without replication as shown in Fig. 9.

From the analysis we found that, the success rate of the scheduling is better than QWS (which is the scheduling algorithm without any consideration of fault tolerance) i.e., the success rate of scheduling with replication is better than the success rate of scheduling without replication with little compensation of resource usage. This work is an enhancement of the previous work [13].

V. CONCLUSION AND FUTURE WORK

Many failures may occur while scheduling workflows in Cloud Environment. Hence the workflow scheduling application should be fault tolerant for failures. The main goal is to schedule workflows and execute these workflows within the deadline in-spite of many failures that occur in the environment. Many existing systems have addressed for providing fault tolerance but without considering the user given deadline.

The proposed work, Fault tolerant workflow scheduling (FTWS), allows users to execute their workflows by satisfying deadline in-spite of different failures that occur in the environment.

Experiments were conducted to test FTWS with random generation of workflows in a simulated cloud computing environment with simulated faults. The results of these experiments were compared with replication and without replication. This showed that FTWS algorithm produced good success rate of scheduling even after considering different failures that occur in the environment.

In future, other failures like data center shutdowns, network failures, etc., may be added to the faults. Mapping of tasks and resources may be improved by maintaining valid datasets at various nodes, so that the transmission overhead of datasets can be reduced.

REFERENCES

[1] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Breandic, “Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility”, Future Generation Computer Systems, Elsevier Science, Amsterdam, June 2009, Volume 25, Number 6, pp. 599-616.

[2] C. Germain-Renaud and O.Rana, “The Convergence of Clouds, Grids and Autonomics”, IEEE Internet Computing, p. 9, 2009

[3] Goverdhan P.V and S.Y.Kulkarni, “Error detection in Grid Computing”, Second International Conference on Computer and Electrical Engineering, 2009.

[4] Z. Shi and J.J.Donagarra, “Scheduling workflow applications on processors with different capabilities,” Future Gen. Comupter systems 22, pp.665-675, 2006.

[5] K.Plankensteiner, R.Prodan, T.Fahringer, A.Kertesz, and P.Kacsuk, “Fault-tolerant behavior in state-of-the-art grid workflow management systems”, Technical Report TR-0091, Institute on Grid Information, Resource and Workflow Monitoring Services, Core GRID-Network of Excellence, October 2007.

[6] Zhifeng Yu, Chenjia Wang and Weisong Shi, "FLAW: FaiLure-Aware Workflow Scheduling in High Performance Computing Systems," Journal of Cluster Computing, Kluwer Academic Publishers Hingham, MA, USA, Vol. 13 Issue 4, pp. 421-434, December 2010.

[7] Kassian plankensteiner, Radu Prodan, and Thomas Fahringer, "A New Fault Tolerance Heuristic for Scientific Workflows in Highly Distributed Environments based on Resubmission Impact", Fifth IEEE International Conference on e-Science, 2009.

[8] Raul Sirvent, Rosa M. Badia and Jesus Labarta, "Graph-based task replication for workflow application", 11th IEEE International Conference on High Performance Computing and Communications, 2009.

[9] Y. Zhang, A. Mandal, and K. Cooper, "Combined fault tolerance and scheduling techiques for workflow applications on computational grids," in International Symposium on Cluster Computing and the Grid. IEEE Computer Sociery, 2009, pp. 244-251.

[10] G. Kandaswamy, A. Mandal, and D. A. Reed, "Fault tolerance and recovery of scientific workflows on computational grids," Cluster Computing and the Grid, IEEE International Symposium on, vol. 0, pp. 777-782, 2008.

[11] Y. Liang, A. Sivasubramaniam, and J. Moreira, "Filtering failure logs for a bluegene/l prototype," in Proceeding of 2005 International Conference on Dependable Systems and Networks (DSN'05). Washington, DC, USA: IEEE Computer Society, 2005, pp. 476-485.

[12] R. Prodan and T. Fahringer. Overhead analysis of scientific workflows in Grid environments. IEEE Transactions on Parallel and Distributed Systems, 19(3):378-393, mar 2008.

[13] Jayadivya S. K. and S. Mary Saira Bhanu, "QoS based Workflow Scheduling in Cloud Computing," in Proceeding of International Conference on Cloud Comuting ICCC-2012, Interscience Research Netwrok, 2012, in Press.

[14] H. Topcuouglu, S. Hariri and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Transactions on Parallel and Distribution Systems, vol. 13, no. 3, pp. 260-274, 2002.

[15] Jia Yu, Rajkumar Buyya and Chen Khong Tham, "Cost-based Scheduling of Scientific Workflows Applications on Utility Grids", In 1st IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, Dec. 5-8,2005.

AUTHORS PROFILE

Jayadivya S K received the B.E Degree in Computer Science & Engineering from Visveshwaraiah Technological University, Bangalore, Karnataka. Currently, she is pursuing the MTech Degree in Computer Science from National Institute of Technology, Tiruchirappalli, India. Her areas of interests include cloud computing, Grid Computing and Real Time Operating Systems.

S. Jaya Nirmala received the B.E Degree in Computer Science and Engineering from Periyar University, Salem and the M.E Degree in Computer Science and Engineering from Anna University, Chennai. Currently, she is pursuing her doctoral degree in the area of Service Discovery in Cloud Computing in the Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli, India . She is an Assistant Professor in National Institute of Technology, Tiruchirappalli, India. Her areas of interest include Cloud Computing, Cryptography and Operating Systems.

Mary Saira Bhanu S received the B.E Degree in Electronics and communication from Madurai Kamaraj University, the M.E Degree in Computer Science from Bharathidasan University and the Ph.D. Degree from National Institute of Technology, Tiruchirappalli. Currently, she is an Associate Professor in the Department of Computer Science and Engineering in National Institute of Technology, Tiruchirappalli, India. Her research interests include OS, Real Time Systems, Distributed Computing, Grid Computing and Cloud Computing.