CHAPTER 4

ANALYSIS AND IMPLEMENTATION OF ENHANCED CHECKPOINT BASED FAULT IDENTIFICATION AND DUAL CRITERIA SCHEDULING

4.1 INTRODUCTION

CC is a fascinating model in provisioning elastic services and on-demand resources. The cloud model considered in the proposed work is based on PaaS as stated by Armbrust et al (2009), where users can create complex requests (tasks) based on standard web services. Each task is made up of one or more tasks and each task execution needs various types of resources. Each task is executed in a particular VM instances, whose resources (i.e. CPU rate, memory size) are isolated via virtual resource isolation technology with idea of Smith & Nair (2005) and Gupta et al (2006). Therefore in CC, the rate of failure is much greater than in traditional computing. Hence, the fault tolerance and scheduling are the important issues to be considered to achieve reliability, availability and QoS.

The fault tolerance service in cloud infrastructure deals with various types of failures like workflow and resource management. This fault tolerance enables a system to continue its operation at reduced level rather than complete failure. A good fault tolerance mechanism can minimize the failure impact on the system and application execution. Hence the failure to be proactively handled. To handle the failure, proposed research work for the fault identification is performed by checkpointing mechanism namely enhanced checkpoint based fault identification. Let us look after the analysis implementation of fault identification and scheduling separate.

Scheduling is a key element to guarantee the performance in case of distributed application running in a cloud environment. Another purpose of the
scheduler in such environment is the implementation of rigid mechanism to handle the possible faults. In such situations a dedicated rescheduling mechanism is the best solution. The performance of rescheduling mechanism is dynamic in the distributed environment. The proposed research work exhibits a generic rescheduling algorithm. The proposed approach supports fault identification and offers an enhanced mechanism for resource management. The evaluation of proposed scheduling algorithm is performed using CloudSim. In evaluation two parameters resource usage and execution time are considered. Hence it is named as dual criteria scheduling in cloud.

4.2 TYPES OF FAULTS

These faults can be classified as,

**Network fault:** A Fault occur in a network due to network partition, Packet Loss, Packet corruption, destination failure, link failure, etc.

**Physical faults:** This Fault can occur in hardware like CPUs, memory, Storage, etc.

**Media faults:** Fault occurs due to media head crashes.

**Processor faults:** Fault occurs in processor due to operating system crashes etc.

**Process faults:** Fault which occurs due to shortage of resource, software bugs etc.

**Service expiry fault:** The service time of a resource may expire while application is using it. A fault can be categorized on the basis of computing resources and time. A failure occurs during computation on system
resources can be classified as omission failure, timing failure, response failure, and crash failure.

**Permanent:** These failures occur by accidentally cutting a wire, power breakdowns and so on. It is easy to reproduce these failures. These failures can cause major disruptions and some part of the system may not be functioning as desired.

**Intermittent:** These are the failures that appear occasionally. Mostly these failures are ignored while testing the system and only appear when the system goes into operation. Therefore, it is hard to predict the extent of damage these failures can bring to the system.

**Transient:** These failures are caused by some inherent fault in the system. However, these failures are corrected by retrying roll back the system to previous state such as restarting software or resending a message. These failures are very common in computer systems.

### 4.3 FAULT TOLERANT TAXONOMY

Based on faults different fault tolerance techniques and strategies are followed. They are Proactive, Reactive and Adaptive fault tolerance.

**Proactive fault tolerance:** The Proactive fault tolerance policy is to predict the fault, recovers the fault, errors, failure and proactively replace the suspected component before it actually occur. It is a concept that prevents compute node failures from impacting running parallel applications by pre-emptively migrating parts of an application (task, process, or virtual machine) away from nodes that are about to fail.
Reactive fault tolerance: Reactive fault tolerance policies reduce the effort of failures, when the failure occurs. This technique makes system more robust. In other words, it is an On-demand fault tolerance.

Adaptive fault tolerance: The fault-tolerance needs of an application change depending on its current position in its state space and the range of control inputs that can be applied. So, in adaptive fault tolerance all procedures are done automatically according to the situation. Adaptive fault tolerance can assure adequate reliability of critical modules, under temporal and resources constraints, by allocating less critical modules by elegantly reducing their resource requirement.

4.4 EXISTING FAULT TOLERANCE TECHNIQUES IN CLOUD

Several fault tolerance techniques that are existing currently in clouds based on Sweta & Ajay (2013) are as follows:

Self-Healing: In this method divide and conquer technique is used, in which a huge task is distributed into several parts. This division is done for better performance. In this, various instances of an application are running on various virtual machines and failure of all this individual instances are handled automatically.

Task Migration: Sometimes it happens that due to some reason a particular machine fails and cannot execute task. On such a failure, a task is migrated to working machine using HA-Proxy. Also, there are algorithms that automatically determines the fault and migrates batch applications within a cloud of multiple datacenters.
**Check Pointing:** It is a proficient task level fault tolerance technique for large applications. In this method, check pointing is done in system. When a task fails, instead of initiating from beginning it is restarted from the recently checked pointed state. Check pointing is carried out periodically i.e., checkpoints are kept and process is executed from the recent check point, once system governs the fault.

**Replication:** Replication means copying. Several replicas of tasks are created and they are run on different resources, for effective execution and for getting the desired result. Hadoop, HA-Proxy, Amazon EC2 like tools are there on which replication can be implemented. Also, there are mainly three different types of replication schemes such as Active Replication, Semi-Active Replication and Passive Replication.

**Task Resubmission:** Many times it happens that due to high network traffic or due to heavy work load, a task may fail, whenever such failed task is detected, at runtime the task is resubmitted either to the same or different working resource for execution. For these, certain algorithms are designed, which assigns task to resources on the basis of certain properties.

**Masking:** After occupation of error recovery the new state needs to be identified as a transformed state. If this process applied systematically even in the absence of effective error provide the user error masking.

**Resource Co-allocation:** In refers to the process of allocating resources for further execution of task. Many algorithms are designed to deal with resource allocation depending on the properties of VM such as workload, type of task, capacity of VM, energy awareness etc.

**Timing Check:** This is deployed with the help of watch dog. It is a simplest technique with time as a critical function. It keeps the track of task
execution, whether the task has been completed in required amount of time or not. Depending on which further action for fault tolerance is taken.

**Rescue Workflow:** A workflow consists of a sequence of connected steps where each step follows without delay or gap and ends just before the subsequent step may begin. In this technique, it allows the workflow to carry on until it becomes unimaginable to move forward without catering the failed task.

**User Specific (defined) Exception Handling:** In this case, whenever fault is detected, action is predefined by the user, i.e. user defines the particular treatment for a task on its failure.

### 4.5 PROPOSED ENHANCED CHECKPOINT BASED FAULT IDENTIFICATION METHOD

![Figure 4.1 Introducing checkpoint on Cloud Environment](image)
There were many algorithms which can help to make the system fault tolerant but no technique is implemented to deal with the situations of faults or failures. Hence to deal with such situations to make the system reliable and to resolve the problem of fault at the time of load balancing a new fault identification technique named as enhanced checkpoint based fault identification technique, is designed and implemented. It is used to identify and predict the fault or failure. Although both aim to improve system performance represented in Figure 4.1 the presence of failure, their effectiveness largely depends on tuning runtime parameters like starvation time, execution time of task and resources mapping towards task. The proposed method provides fault identification during task scheduling process.

4.5.1 System overview

Let us consider a DAG representation of the system. Consider a cloud host (CH) consists of set of tasks \( T = \{ Ch1, Ch2, ..., ChM \} \), and the set of resources or nodes \( CR= \{ r1, r2, ..., rN \} \), where \( N, M \) are the number of nodes and tasks respectively. Every processor is associated with group of resource having the trust value indicating the reliability of that node, where \( 0 \leq r_j \leq 1, j = 1,2, ..., N \), and initialized as 1 at the beginning. Let us assume that the simulated job submission data includes a mix of jobs that vary according to checkpoint ability half were checkpointable, half had to start over upon node failure, checkpoint periods and runtime.

The simulation is done based on their execution on resources whose availability was determined from the trace data. The ECFI designed and tested with a simple prediction-based scheduler that chooses resources based on their predicted failure rate during the application’s execution interval, current CPU load and their CPU speed.
Algorithm for Enhanced Checkpoint based fault identification

Step 1: Start
Step 2: M: = number of Task cloudlets
Step 3: N: = number of Resource cloudlet
Step 4: Discovermappedresource: = 0 (i.e. Ch (m tasks) <> Cr (n Resources))
Step 5: Set Checkpoint= 0 (task or resource failure occurs).
Step 6: Do (Formation of load balancing using scheduling algorithms)
   a: Input processing cloudlet status
Step 7: If input processingcloudletstatus = pass then
   Discovermappedresource = discovermappedresource +1
   Set checkpoint
Step 8: While (all tasks mapped resources)
   a: If (failure = yes) then
   b: If (discovermappedresource<1) then
   c: Roll back the process
   d: Else if (discovermappedresource>1) then
Step 9: Check for the next task and available resources
Step 10: Update value the availability, time counter
Step 11: Update starting time, processinginital and completion time
Step 12: Compute starvation time = processinginital – starting time
Step 13: Compute execution time = completion time – starvation time
   a: Else
   b: Raise flag towards failure of cloudlet
Step 14: End
4.5.2 Working model

The ECFI algorithm will help to make the CE more reliable. In the proposed ECFI, whenever a task request arises, triggering is initiated to check task dependency. If the resource can be allocated without any deadlock then resource is allotted, otherwise request is transferred to waiting state. Each task is mapped towards resources in terms of many to many relationship. Each state of resource and task node availability is recognized and flag is raised. At the same time, timer is initiated when the task was assigned. Periodically the tasks are queued for processing and waiting for resources. Here in above ECFI algorithm every time anew task is mapped with resources is checked, the value is stored in discover mapped resource. This collects information about the each task completion and waiting for resource. If any task cloudlet encounters fault then the flag raises to highlight the failure.

![Diagram](image.png)

Figure 4.2 Workflow of task cloudlet mapping towards resources

Cloudlets
When the flag is raised to indicate the failure the fault management system will be redirected towards the scheduling methodologies to recover. Each task requires more than one resource then each resource availability towards task is split into subtasks and their subtasks are verified. When FIFO scheduling is implemented as shown in Figure 4.2 by considering number of resources to be mapped to M number of task at a time only. Now for first schedule, First Come First Serve (FCFS) scheduler will schedule task of Ch1, Ch2 … ChM need of resource cloudlet (Cr) in parallel, so it will take X time to execute M tasks. During this schedule, remaining cloud resources remains idle and Cloud Host (CH) with tasks remains idle is waiting due to Starvation. So in this manner resources are not utilized properly and tasks are not scheduled properly may result in increase of execution time or failure of task to complete the whole workflow execution.

4.6 PERFORMANCE EVALUATION

This ECFI algorithm is simulated using CloudSim, a toolkit for modeling and simulation of virtualized cloud based datacenter environment including dedicated management interfaces for virtual machines, memory and bandwidth.

4.6.1 Simulation Setup

Table 4.1 Task Failure identified through ECFI with FIFO of total job

<table>
<thead>
<tr>
<th>No. of Job</th>
<th>Resource Failure</th>
<th>Network Failure</th>
<th>Total failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>8</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>40</td>
<td>12</td>
<td>11</td>
<td>23</td>
</tr>
<tr>
<td>60</td>
<td>17</td>
<td>9</td>
<td>26</td>
</tr>
<tr>
<td>80</td>
<td>29</td>
<td>8</td>
<td>37</td>
</tr>
<tr>
<td>100</td>
<td>19</td>
<td>12</td>
<td>31</td>
</tr>
</tbody>
</table>
The simulation work is conducted on an Intel Core i3 CPU machine having configuration: 3.20 GHz and 2 GB of RAM running a Windows 7 Ultimate edition and JDK 1.6 along with CloudSim toolkit. For the simulation hundred cloudlets are considered and the maximum size of the resources is limited to M/5, where M is the total number of task cloudlets used in the cloud environment. If the unscheduled tasks are more than the limit, dynamically the size or number of resources needed is changed. In this simulation, it is assumed to have an N resources need to be mapped towards M task cloudlets in heterogeneous distributed computing environments.

Figure 4.3  Task Failure identified through ECFI during FIFO
For the simulation it is assumed to have a 100 tasks assigned with utilization of 200 resources and categorized into subtask in CE. Each 20 task with increment of 20 task is considered for evaluation of task failures identified and representing failure occurred due to resource and network is shown in the Figure 4.3 based on Table 4.1. The result shows that the identification of failure in resource is 11.65% worse than the network failure in cloud workflow.

4.6.2 Simulation Result

The given algorithm will continuously allocate the resources as per the requests generated by the user. Before the allocation, this algorithm checks that the system is in safe state or not before any allocation.

4.7 WORKFLOW OF SCHEDULING STRATEGY

![Figure 4.4 Through Checkpoint resource utilization analyzed in CE](image-url)
At first look, the reason for maintaining the best schedule based on task completion time is not clear. But after observation the entire scheduling process tends to yield good schedules in terms of resource usage throughout the application. However, in this scheme as highlighted in Figure 4.4 the issue raise is that schedules are less attractive to task completion time. This observation has led to incorporation of a relaxed schedule selection scheme as an effective one to reduce the task completion time and resource usage. The primary objective of this effort is to minimize application completion time, at the expense of redundant resource usage. This scheme indirectly takes resource usage into consideration.

There might be delay in submitting the tasks during the actual task dispatch process. So, it inefficiently takes a long time to complete the execution in turn rescheduling the events. In other words, the late completion of a task does not disturb the task completion time of a workflow application as long as the time of completion is not later than the actual latest completion time of the task. Hence, the delay is acceptable.

4.8 PROPOSED DUAL CRITERIA SCHEDULING ALGORITHM TO MANAGE WORKFLOW

The flow diagram of the Dual Criteria Scheduling algorithm is depicted in Figure 4.5. Workflow applications takes advantage of a cloud platform such as, interjob dependencies, resource heterogeneity and dynamism, to force a great burden on scheduling. In the proposed system, scheduling of workflow applications is done using dual criteria scheduling algorithm. The fault is identified by using enhanced checkpoint based fault identification techniques which is used during the scheduling process as discussed in the beginning of this Chapter 5.
Figure 4.5  Working Mechanism of DCS Algorithm

By implementing these two methods together it minimizes the application completion time and resource usage. The principle of fault identification is as soon as a resource failure or task failure is discovered during scheduling, all jobs submitted to the failed resource are rescheduled to the available resource. In each scheduling round, not only newly arrived jobs are considered for submission, but also all jobs distributed to failed nodes. The benefits of proposed system are the good quality of output schedules to increase the performance of CE with less resource usage and completion time with adaptability to performance fluctuations on resources.

4.8.1 System Overview

Let us consider a DAG G= (V, E) representation of the system. Where V represents the vertex or node and E represents the edges connecting two nodes. Consider a cloud host (CH) consists of set of tasks T = {Ch1 , Ch2 ,..., ch_M },and the set of resources or nodes CR = { r1 , r2 ,..., r_N } ,where N,M are the number of nodes and tasks respectively. Every processor
is associated with group of resources having the trust value indicating the reliability of that node, where \(0 \leq r_j \leq 1, j = 1,2,..., N\), and initialized as 1 at the beginning. Let us assume that the simulated job submission data includes a mix of jobs that vary according to checkpoint ability half were checkpoint able, half had to start over upon node failure, checkpoint periods and runtime.

4.8.2 Working Model

![Flow chart of Working on Dual Criteria Scheduling](image)

Figure 4.6 Flow chart of Working on Dual Criteria Scheduling
Let us assume that each processor has an infinite buffer size to store tasks waiting for execution. So this assumption will eliminate the dropping of a task due to unavailability of buffer space. Task size is assumed to have normal and it includes both the program and data sizes. Generating a random schedule, the initial solution undergoes the manipulation process of algorithm repeatedly for further improvement in make-duration and/or resource utilization. This schedule manipulation involves a branch-and-bound-style technique and two types of mutation (swap and point). Each task in the initial random schedule is tried on each cloudlet to check whether any of these matches shortens the current completion time. If one or more matches better than the original match is identified, the cloudlet on which the make-duration is reduced is selected.

The proposed method will keep track of the two best schedules (strict and relaxed) possibly the same schedule based on completion time and resource usage, respectively. It is observed from Figure 4.6 that the application of such strict criteria throughout the entire scheduling process tends to produce very good schedules in terms of resource usage; however, this strict schedule selection scheme raises the issue that schedules are less attractive, as far as completion time is concerned.

### 4.9 ALGORITHM FOR DUAL CRITERIA SCHEDULING

**Input:** A workflow application CE (CH, CR), a set CH of cH cloudlets in cloud CR, #iteration g

**Output:** A schedule of CG onto CH

Step 1: Compute all the task assigned for CH found in CE

Step 2: Sort tasks in decreasing order based on priority

Step 3: Generate a random schedule, S

Step 4: Set random completion time (md), best completion time (md’), best resource usage (ru’) = \( \infty \)
Step 5: While $g^{th}$ iteration is not reached do
   a. for all the tasks in $C_G$ do
      i. try each task on for all $c_r$ needed for $C_H$
      ii. Select the best cloudlet $C_{h^*}$ on which completion time, $md$ of $S$ is the smallest
      iii. Let $S^* = S$ if $md < md'$
      iv. Let $S' = S$ if $md < md'$ and resource usage $ru$ of $S < ru'$
      v. Let $S = S'$
   b. end for
   c. if no improvement on $S^*$ made then
      i. Mutate $S^*$ using either point or swap mutation with a probability of 0.5
      ii. Let $S = the mutated schedule (S^*)'$
   d. end if
Step 6: end while
Step 7: Let $S = S^*$
Step 8: Compute the tangible latest completion time TLCT of all the task assigned for $C_H$
Step 9: while for all the tasks are scheduled with resources do
   a. Dispatch all, ready tasks in $S$
   b. Remove the dispatched tasks from $V$
   c. Wait until any task $n$ to finish
   d. If $TCT(n) > TLCT(n)$ then
      i. Let $S^*$ and $S' = S$
      ii. goto Step 5
   e. end if
Step 10: end while

This observation has led to the incorporation of a loosened schedule selection scheme ($S^*$) as an effective solution to this issue. This scheme helps
shorten the completion time noticeably, as the result of a compromise between completion time and reason for maintaining the best schedule is based only on make-duration because DCS determines the quality of schedule based on both completion time and resource usage.

The scheme indirectly takes resource usage into account. Specifically, the resource usage of the best schedule kept in S" is guarded to remain at a reasonable level by S', which can be seen as the quality guarantor of DCS for resource usage. S" is a slight modification of S', and the degree of sacrifice made to the resource usage of S" is minimal. At the end of each iteration, mutation is considered if no improvement on S" is made during the current iteration. DCS randomly chooses a mutation method between point and swap mutations and mutates each task in S" with a probability of 0.5 sufficient to generate substantially different schedules. Mutation occurs only with unscheduled tasks during rescheduling. The mutated schedule is then used as the current schedule (S) and the strict best schedule (S') in the next iteration. If there have been some improvements on S" in the current iteration, S (i.e., S') is passed onto the next iteration for further improvements.

This is because changes made to tasks with low b-level values (low-priority tasks) may enable better task-cloudlet matches, in the next iteration, for unchanged high-priority tasks leading to improvement in the quality of the current schedule (S) as well as the best schedules (S" and S'). Schedule manipulation process repeats for a predefined number of iterations. Now, tasks in the best schedule (S") are dispatched to their assigned cloudlets, as they become ready, i.e., their predecessor tasks have finished. During this actual task dispatch process, there might be cases in which some tasks are delayed to complete their execution. Such trigger situation rescheduling events. Here, it is very crucial how to define unacceptably long. For each task n_i in V, DCS
computes the Tangible Latest Completion time (TLCT (n_i)), which is a modification of Latest Completion time (LCT).

The modification is made because DCS uses task insertion. Their parameters are listed below,

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH</td>
<td>Cloud Hosts with Tasks</td>
</tr>
<tr>
<td>CR</td>
<td>Cloud Resources</td>
</tr>
<tr>
<td>CG</td>
<td>Unscheduled Task in Hosts</td>
</tr>
<tr>
<td>CE</td>
<td>Cloud Environment</td>
</tr>
<tr>
<td>S</td>
<td>Schedule</td>
</tr>
<tr>
<td>S*</td>
<td>Loosened schedule selection scheme</td>
</tr>
<tr>
<td>S’</td>
<td>Strict best schedule</td>
</tr>
<tr>
<td>Wij</td>
<td>Waiting time of Chi for particular Crj</td>
</tr>
<tr>
<td>LCT</td>
<td>Latest Completion Time</td>
</tr>
<tr>
<td>TLCT</td>
<td>Tangible Latest Completion Time</td>
</tr>
<tr>
<td>ru</td>
<td>Resource Usage</td>
</tr>
<tr>
<td>ru’</td>
<td>Best resource Usage</td>
</tr>
</tbody>
</table>

4.10 PERFORMANCE AND BOUND OF DCS

Through Checkpointing method in DCS algorithm’s tangible latest start and completion time of a task c_hi on a cloudlet c_hj are defined as,

\[
\text{TLSCT} (c_{hi}, C_{hj}) = \text{TLCT} (c_{hi}, C_{hj}) - w_{ij} \tag{4.1}
\]

Where TLSCT (c_{hi}, C_{hj}) calculated through equation (4.1) is the actual latest start time of the next task scheduled after c_hi on the same cloudlet C_{hj}. The TLCT of a task is an indicator of whether the delay in the completion of the task is acceptable. In other words, the late completion of a task does not affect the make-duration of a given workflow application as long as the time of the
completion is no later than the actual latest completion time of the task; hence, the delay is acceptable.

4.10.1 Simulation setup

Dual Criteria Scheduling and FIFO Scheduling in Cloud Environment were simulated using CloudSim. Datacenter is usually composed of a set of hosts, each of which represents a physical computing node in the cloud. In simulation, 30 cloudlets with multiple tasks are created with heterogeneous configuration characteristics randomly given in Table 4.2. To model cloud users, create application tasks that contain information related to execution details such as task processing requirements, disk I/O operations and the size of input files. Simulate 16 users in a cloud system, and each user with an exponentially distributed number of tasks. Each task information consists of request time, starting time, bandwidth and other resources requirements.

<table>
<thead>
<tr>
<th>Table 4.2 Resource Characteristics for Cloud Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristics</td>
</tr>
<tr>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Machine architecture</td>
</tr>
<tr>
<td>Operating system</td>
</tr>
<tr>
<td>Virtual machine monitor</td>
</tr>
<tr>
<td>Number of Processing Environment</td>
</tr>
<tr>
<td>MIPS rating per Processing Environment</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Storage</td>
</tr>
<tr>
<td>Bandwidth</td>
</tr>
</tbody>
</table>
4.10.1.1 Evaluation of task scheduling in cloud

Table 4.3 Task Graph Properties

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tasks, $v$</td>
<td>{20, 40, 60, 20, 40, 60, 20}</td>
</tr>
<tr>
<td>Communication-to-Computation Ratio</td>
<td>{0.1, 0.2, 0.3, 0.1, 0.1, 0.2, 0.8}</td>
</tr>
<tr>
<td>Out degree of a node</td>
<td>U(1, 10)</td>
</tr>
</tbody>
</table>

Task graph set used in simulations consists of 42 base task graphs generated with combinations of seven graph sizes and six Communication-to-Computation Ratio (CCR) as in Table 4.3. For each combination, 20 variant task graphs are randomly generated, retaining the characteristics of the base task graph.

Table 4.4 Tangible Latest Completion time among FIFO and DCS

<table>
<thead>
<tr>
<th>Task List</th>
<th>CCR</th>
<th>FIFO (Sec)</th>
<th>DCS (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>424603</td>
<td>364600</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>424595</td>
<td>364570</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
<td>424586</td>
<td>364581</td>
</tr>
<tr>
<td>4</td>
<td>0.1</td>
<td>424562</td>
<td>364551</td>
</tr>
<tr>
<td>5</td>
<td>0.1</td>
<td>424560</td>
<td>364553</td>
</tr>
<tr>
<td>6</td>
<td>0.2</td>
<td>424567</td>
<td>364557</td>
</tr>
<tr>
<td>7</td>
<td>0.8</td>
<td>424540</td>
<td>364545</td>
</tr>
</tbody>
</table>

The computation and communication times of the tasks in each task graph were randomly selected from a uniform distribution with the mean equal to the chosen average computation and communication times. The out degree of each node in a task graph was random and uniformly distributed between 0 and 10. Specifically, for a node, its out degree of one indicates that it has only one successor task.
Figure 4.7  Graphical Representation of Tangible Latest Completion time among FIFO and DCS

Trials performed with seven different CCRs as stated in Table 4.4, and the experimental results obtained with two significant CCRs (0.1 and 1.0) are presented. For instance, the test result acquired from the task graphs with CCR 5.0 does not show a significant difference from the test result acquired from the task graphs with CCR 10.0. Tests performed with seven different CCRs as stated in Table 4.4 and their Tangible Latest Completion time for First in First out Scheduling and Dual Criteria Scheduling was represented in Figure 4.7 based on Table 4.4. DCS provides average of 18.75% efficient completion time than FIFO scheduling.

4.10.1.2 Evaluating of resource failure in cloud

Figure 4.8 represents the task failure due to resource failure in FIFO and DCS with communication-to-computation ratio maintaining normalized resource usage based on Table 4.5.
Figure 4.8 Task Failure due to Resource Failure in FIFO and DCS

Resource utilization are considered for task’s completion time collection on each. Through DCS 11.45% of resource failures is reduced than FIFO scheduling due to iterative checking of resources availability.

Table 4.5 Task Failure due to Resource Failure in FIFO and DCS

<table>
<thead>
<tr>
<th>Task List</th>
<th>Resource Failure FIFO</th>
<th>Resource Failure DCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>12</td>
</tr>
</tbody>
</table>
4.11 SUMMARY

There are many algorithms which can help to make the system fault tolerant but no technique is implemented to deal with the situations of faults or failures. There is no any fault identification algorithm that can be used along with the scheduling algorithm. Hence this ECFI algorithm tried to deal with such situations to make the system reliable. It also resolves the problem when fault occurs at the time of load balancing. The ECFI is implemented with various scheduling algorithms such as Dual Criteria Scheduling and Particle Swarm Optimization in the following proposed work. Each scheduling needs to be verified through checkpoints in order to classify the resource failure to improve the performance of the cloud environment. Through above mentioned scheduling strategies, faults are identified and task is scheduled based on resource availability.

Developing a workflow mechanism, to schedule a task in Cloud environment is performed without considering resources and in turn it suffers from starvation time before execution starts in DCS. Hence the total completion time increases due to less resource usage. So in order to overcome this problem the above scheduling is considered with maximum resource allocation to improve quality of service along with efficient completion time and minimized starvation time. Resource allocation in cloud environment processed through reservation cluster can therefore be formulated as a dual criteria optimization problem. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure.