Computer Systems Science & Engineering
DOI:10.32604/csse.2021.015027
Article

Tech Science Press

Fault Aware Dynamic Resource Manager for Fault Recognition and Avoidance in Cloud

Nandhini Jembu Mohanram[1,2,*], Gnanasekaran Thangavel[3] and N. M. Jothi Swaroopan[4]

[1]Information and Communication Engineering Department, Anna University, Chennai, Tamil Nadu, India
[2]Department of Information Technology, Sri Sai Ram Institute of Technology, Chennai, Tamil Nadu, India
[3]RMK Engineering College, Chennai, Tamil Nadu, India
[4]Department of Electrical and Electronics Engineering, RMK Engineering College, Chennai, Tamil Nadu, India
[*]Corresponding Author: Nandhini Jembu Mohanram. Email: nandhiniannauniv.phd@gmail.com
Received: 03 November 2020; Accepted: 16 December 2020

Abstract: Fault tolerance (FT) schemes are intended to work on a minimized and static amount of physical resources. When a host failure occurs, the conventional FT frequently proceeds with the execution on the accessible working hosts. This methodology saves the execution state and applications to complete without disruption. However, the dynamicity of open cloud assets is not seen when taking scheduling choices. Existing optimization techniques are intended in dealing with resource scheduling. This method will be utilized for distributing the approaching tasks to the VMs. However, the dynamic scheduling for this procedure doesn't accomplish the objective of adaptation of internal failure. The scheme prefers jobs in the activity list with the most elevated execution time on resources that can execute in a shorter timeframe, but it suffers with higher makespan; poor resource usage and unbalance load concerns. To overcome the above mentioned issue, Fault Aware Dynamic Resource Manager (FADRM) is proposed that enhances the mechanism to Multi-stage Resilience Manager at an application-level FT arrangement. Proposed FADRM method gives FT a Multi-stage Resilience Manager (MRM) in the client and application layers, and simultaneously decreases the over-head and degradations. It additionally provides safety to the application execution considering the clients, application and framework necessities. Based on experimental evaluations, Proposed Fault Aware Dynamic Resource Manager (FADRM) method 157.5 MakeSpan (MS) time, 0.38 Fault Rate (FR), 0.25 Failure Delay (FD) and improves 5.5 Performance Improvement Ratio (PIR) for 25, 50, 75 and 100 tasks and 475 MakeSpan (MS) time, 0.40 Fault Rate (FR), 1.30 Failure Delay (FD) and improves 6.75 improves Performance Improvement Ratio (PER) for 100, 200, 300 and 500 Tasks compare than existing methodologies.

Keywords: Cloud computing; fault aware dynamic resource manager; fault tolerance; makespan; fault rate; failure delay; performance improvement ratio

1 Introduction

Cloud computing has drastically changed the way web-scale administrators offers types of assistance and oversee infrastructures. The evasion of in-house infrastructures and the acceleration of time to advertise are key parts of this development. This interruption has been basically determined by practical elements derived from the utilization of shared business off-resources arranged in topographically distributed data centers. However, cloud is advancing past money saving advantages as suppliers are elite resource types including process or memory-improved occasions, just as GPU. The advancement as far as execution, along with the range of estimating models, versatility and high-accessibility makes cloud a serious stage for logical processing. Larger cloud environments are inclined to dissatisfactions, by putting the client's application execution in danger. In such situations, the exposed hosts are disturbed during the provisioning of virtualized resources; this expands the likelihood of failures as depicted. Moreover, virtualized resources are straightforwardly influenced by failures on the fundamental physical has just a miss-arrangement or support strategies explicit to the cloud providers. Many parallel distributed applications expand on top MPI, which follows a course fail-stop semantic that terminates the execution if there should be an occurrence of host failure cluster.

Failure tolerance (FT) schemes are intended to work on a reduced and static amount of physical resources. That is, if there is an occurrence of a host failure, customary FT frequently endeavors to proceed with the execution on the staying accessible working hosts? The methodology permits safeguarding the execution state and applications to complete without disturbance. For cloud situations, customary FT arrangements must be updated to use local cloud attributes, for example, the adaptability of virtual resource management for both, security and recuperation tasks of FT. FT designs for cloud which additionally requires capacities of FT schemes for a few clients, executing numerous application on various virtual bunches. For parallel applications actualized with MPI, adaptations to internal failure arrangements are utilized to handle this issue. The FT arrangements in cloud are principally classified as proactive and responsive. Proactive FT continually screens the framework to make disappointments forecast. Then, responsive provisions, performs representations of the framework, that are utilized during the recuperation cycle. The primary aim in proactive methodologies is to source to classify so as to sequence the impacts of the failures before they occur. Despite the fact that, classifications may not be precise, the proactive FT arrangements would not be appropriated when more significant levels of accessibility are pursued.

Fault Aware Dynamic Resource Manager (FADRM) presents the enhanced model to the Multi-stage Resilience Manager so as to help an application-level FT scheme. It influences FT by consolidating the application-layer checkpoints with a message logs and to deploy the un-coordinated and semi-coordinated mechanisms. This scheme additionally incorporates a unique FT resource manager. Experimental design and assessment is completed by utilizing a Sender-Based Message Logger and ULFM (user level fault tolerant management) augmentation. For parallel applications, Fault Aware Dynamic Resource Manager (FADRM) is a check pointing service for cloud environments. It empowers application check pointing and performs relocation on heterogeneous cloud environments. Fault Aware Dynamic Resource Manager (FADRM) utilizes a unified methodology for the capacity of the checkpoint services. With respect to applications, a structure is proposed, planned in a framework level and particular point of view, to give FT in clouds. It performs VM replication to ensure the execution condition of a client, rather than securing the applications themselves. Proactive FT scheme depends on observing the VMs health status to act on the off chance when a failure is predicted. The method concentrates on offering a FT scheme as assistance; so as to safe equal stateful applications against lasting failures for few users, utilizing different virtual clusters. It is composed of modules to live-relocate VMs, failures classifications and managers. The details of paper contribution are given below:

- To develop a Fault Aware Dynamic Resource Manager (FADRM) for offering the FT with a Multi-stage Resilience Manager (MRM) in the client and application layers, and simultaneously decrease the over-head with degrading.
- To secure the application execution considering the clients, application and framework prerequisites.
- To accelerate fault tolerance segments for logical parallel applications deployed with MPI, utilizing client, application and runtime cloud environment prerequisites.
- To provide Fault Aware Dynamic Resource Manager (FADRM) with high accessibility to the application that can assist it with the process completion of conveying to clients for the expected outcomes.
- To minimize MakeSpan (MS) time, Fault Rate (FR), Failure Delay (FD) and improve the Performance Improvement Ratio (PIR) of proposed system for 25, 50,75 and 100 tasks in comparison with the conventional methodologies.

The remainder sections of paper are organized as follows: Section 2 details about recent work and method related to fault identification during resource management in cloud. Section 3 describes the proposed methodology, workflow, and module work and implementation details. Section 4 discuses about deployment setup, input parameters, and simulation result with comparative analysis. Section 5 summarizes overall fault tolerant work during dynamic resource allocation with future outcome.

2 Literature Work

In [1] explained dynamic clustering league championship algorithm (DCLCA) dynamic grouping class title calculation (DCLCA) scheduling strategy for fault tolerance to address cloud task arrangement which would imitate about the current open asset and decrease the failure of self-sufficient assignments. In [2] studied about extensive outline of adaptation to non-critical failure related issues in cloud computing; underlining upon the massive ideas, structural subtleties, and the state-of-art procedures and techniques. The overview lists a couple of promising strategies that might be utilized for productive solutions and furthermore, recognizes significant research in this field. In [3] focused on describing the repetitive failures in traditional Cloud computing environments, investigating the impacts of failures on client's applications, and reviewing adaptation to internal failure arrangements comparing to each class of disappointments. The technique additionally studied the point of view of offering adaptation to internal failure as a support of client's applications as one of the compelling way to address client's dependability and accessibility concerns. In [4] suffered the issues by utilizing replication and resubmission methods. At that point it reschedules the undertaking once the disappointment happens to the most elevated dependability preparing hub as opposed to repeating this task to every accessible hub. In [5] depicted engineering models which the infrastructure related services are made accessible to the customers in any event, during the faults making the whole cycle of distributed computing solid and viable.

In [6] communicated to decrease the chance of fault occurrences in the framework by a reasonable distribution of client work demands among accessible resources. The framework oversees irregular circumstances that may prompt failure by circulating the approaching position demand dependent on the reliability of processing of handling hubs, i.e., virtual machines (VMs). In [7] upgraded fault tolerance approach where a model is intended to endure deficiencies dependent on the dependability of each process hub (virtual machine) and can be supplanted if the exhibition isn't ideal. Preliminary test of method demonstrates that the pace of increment in pass rate surpasses the reduction in failure rate and it additionally considers forward and in reverse recovery utilizing different programming tools. In [8] inspected meanings of CPS just as the three previously mentioned computing ideal models and afterward shed new light on completely established structures. The work likewise reviews on the application level of Cloud-Fog-Edge Computing in CPS separately and jump into different methods and systems to install

large information applications. In [9] explained pro-active approach for fault tolerant dependent on processing power, memory and network limit to enhance the resources reliability. It estimate the reliability of each VM based on success ration of job deployment and then schedule the job on highly reliable VM. In [10] discussed the fault tolerance scheme or cloud environments, evaluate whether this technique is robust and reliable in cloud.

In [11] described Cluster based Heterogeneous Earliest Finish Time (CHEFT) algorithm to upgrade the scheduling and fault tolerance mechanism for SWf in exceptionally circulated cloud. This method utilizes idle time of the provisioned resources for resubmitting failure clustered jobs for fruitful execution of SWf. In [12] tended to about a model of starting VM fault-tolerant placement for star topological data centers of cloud frameworks is based on various components, including the service level agreement violation rate, resource remaining rate, power utilization rate, failure rate, and adaptation to fault tolerance cost. The service providing VMs are placed by the ant colony algorithms, and the repetitive VMs are put by the conventional methodologies. In [13] communicated virtual cluster allocation method as per the VM attributes to decrease the absolute system resource utilization and total energy consumption in the data center. The determination of the ideal objective PMs is displayed as an enhancement issue that is understood utilizing an improved particle swarm optimization technique. In [14] investigated this exertion and produced a top to taxonomy of them. The framework clarified the ontology of faults and fault-tolerant methods at that point position the current work process management frameworks as for the scientific classifications and the procedures. The framework characterizes different failure models, measurements, tools, and support systems. In [15] tended to RT-PUSH a VM fault detector dependent on timeout and cutoff time for cloud framework running task. The viability of the model is assessed through progress rate and execution drop rate measurements.

In [16] considered an investigation and assessment is additionally performed relating to the fault tolerance and fault detection systems. The overview uncovered that versatile and intelligent fault identification issue and resilience methods can improve the reliability of grid working environments. In [17] explained job scheduling with fault tolerance in grid computing utilizing ant colony optimization to guarantee that positions are executed effectively during resource fault? The method is utilized the utilization of resource failure rate, just as checkpoint-based roll back recovery technique. Check-pointing minimizes the measure of work that is lost upon fault of the framework by promptly sparing the condition of the framework. In [18] researched to give a prevalent perception of various QOS based assistance scheduling which worked to upgrade the execution in fog computing, and moreover review on different fault tolerant based methods associated with fog computing. In [19] discussed with the comprehension of fault tolerance methods in cloud and examination with different models on different parameters have been completed. Fault tolerance method is concentrated with the assistance of programming language brought together for demonstrating language and state outlines are structured and approved through the concept of limited state machine. In [20] focused on the standard fault tolerant in cloud computing. Cloud computing is another field of examination contrasted with different innovations, a ton of exploration works are being completed, particularly in building up an independent fault tolerance technique.

3 Proposed Methodology

Fault Aware Dynamic Resource Manager (FADRM) is introduced for composed conventions, despite the fact that the engineering configuration is extensible, and conceivable to help uncoordinated and semi-coordinated rollback-recovery conventions. The framework used in the FARDM is at application-level, off programmed and straightforward scheme for recouping applications in the event of failures which plays out any activity when failure is identified. The scheme utilizes semi-coordinated and uncoordinated rollback-rollback-recovery conventions. FARDM integrates the application-level checkpoints with a

sender-based message logs for discovery and recovery purposes. An effective fault aware based dynamic resource manager is included, which involves in observing of primary memory utilization for the logger benefits, permitting and distinguishing when its utilization reaches the boundaries. With this information, it calls customized designated checkpoints, in an optimistic way, which grants liberating memory underpins used for the message logs to evade keep down or slow down of the application execution. FARDM scheme limits the customer's application layer from failures. It is reasonable the execution circumstance with a fault aware dynamic resource manager, which manages failure recovery of the customer's application execution when fault is perceived. The scheme is described in Fig. 1. The fault aware dynamic resource manager is made out of sender-based message logs, which integrated with the application-level uncoordinated and semi-coordinated checkpoints, maintains the application during fault-free executions. Fault aware dynamic resource manager is appended, which screens and oversees FARDM resource utilization for FT assurance.

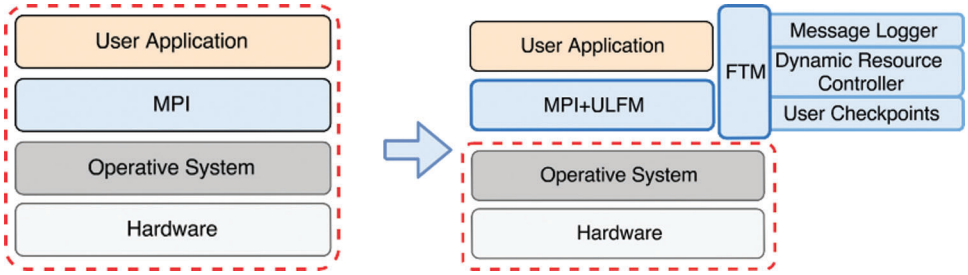

Figure 1: Process flow of proposed FARDM in the Application-Level

4 Results and Discussion

4.1 Protection

FARDM follows both the uncoordinated and semi-coordinated rollback-recovery conventions. The application state is spared utilizing application-level checkpoints and the swapped messages are saved in a sender-based message logs. FARDM explains the development of the Fault Tolerance (FT) in the application-level to give high accessibility to the client's applications in a programmed and straightforward way. The fault tolerance assurance is finished by describing the checkpoints of the application measures and putting away swapped the messages between measures in a sender-based message logs, during the application execution

4.2 Application Checkpoints

The checkpoints are obtained from the function-level, the check pointing activity is started by the application, and henceforth adjustments in the applications source are fundamental for the checkpoint incantation, despite the fact that the application method remains neutral. The checkpoints incantation is embedded during regular synchronization of the application measures. The checkpoints stock up structures containing just vital data to reestablish execution if there should be raised an occurrence of failures, maintaining a strategic distance from the need to store the specific data.

4.3 Message Logging

During the application execution, messages are put away into a message logs facility, so as to replay them to the cycles that are failures. The uncoordinated and semi-coordinated conventions have the preferred position that just fault measures restart execution utilizing the last solid checkpoint accessible,

permitting different cycles to proceed with their execution, reducing the computing paradigm. After the cycles are restarted, during the re-execution, they straightforwardly consume messages from the message logs facility. FARDM in the application-level uses a pessimistic sender-based message logs. For the uncoordinated methodology, all swapped messages between the application measures are stored. The semi-coordinated approach store just swapped messages between the application measures that are in particular hubs.

4.4 Fault Detection, Reconfiguration and Recovery

When, fault identified a component of location is expected to begin the recovery procedure. For the FARDM in the application-level, it is utilized to identify faults. FARDM executes mistake handler, which is conventions by the ULFM (user level fault management) recognition scheme to recuperation the application execution. The brought forth measure, recognizes that it has been re-propelled and load the checkpoint record. After the checkpoint record is stacked, the factors in the process are set as they were in the last checkpoint earlier faults. The system proceeds to the right execution line so as to proceed with the application execution. The messages are expended from the message logs to complete the re-execution. Then, fault free cycles can proceed with their execution.

4.5 Fault Aware Dynamic Resource Manager

Fault Aware Dynamic Resource Manager (FADRM) presents the enhanced model to the Multi-stage Resilience Manager so as to help an application-level FT scheme. It influences FT consolidating the application-layer checkpoints with a message logs to deploy the un-coordinated and semi-coordinated mechanism. FT security requires resources and it accompanies overhead for the clients applications. The uncoordinated and semi-coordinated conventions restrict a strategic distance from the restart of the apparent multitude of uses measures when a failure recognized. In spite of the fact, they require logs to replay messages to reestablished measures. To reduce fault free overhead, the logs utilizes primary memory to store measures messages, when enough memory is accessible, as it frequently gives higher speed access. The accessible memory can quickly ran out because of FT security jobs. The effect of running out of primary memory can bring about the application execution become slowed down.

FADRM plays out the security of the application execution, storing both: checkpoints, and messages of the application measures. Despite the fact that there is not control performed in term of the resource usage during FT safety so as to restrict free memory ran out due to FADRM security jobs; a fault aware dynamic resource manager is tended to with FADRM. It works at every hub of the application execution. The fault aware dynamic resource manager continually checks at presently assigned memory of the variety of records structure utilized for message logging purposes. At the point when it identifies that utilization is arriving at as far as possible, it triggers a programmed checkpoint invocation, which thus stores the condition of the application and free and utilized memory supports for logging purposes, giving the application. FADRM security and simultaneously abstain from obstructing the applications memory usage. Intermission based checkpoints are not affected and when they are prepared, the memory are delivered. As the checking is performed simultaneously as the message storage into the message logs, no extra overhead is predicted. At the point, programmed checkpoint is raised to free the memory utilization. The application execution period, the memory may run out significance the lost as far as evaluation of the application execution. The fault aware dynamic resource manager distinguishes it and consequently invokes a checkpoint formation of every application cycle, when they arrive at the following regular synchronization point, permitting freeing memory utilization by the logging benefit, along these lines keeping away from the application deployment stall due to the lacking of main memory.

The FADRM handler for fault identification, reconfiguration and recovery is described. For straightforwardness, one cycle for each hub is expected, utilizing the uncoordinated convention with a

critical sender-based message logs. The hub running P3 comes up failures, and P2 is the principal cycle which recognizes fault. It is distinguished by ULFM (user level fault mismanagement), causing the revocation of the worldwide communicator utilizing MPI_Comm_revoke. After the revocation, all outstanding cycles are informed and they are disappeared the worldwide communicator, taking out the failure measures utilizing the MPI_Comm_shrink call. Fault identification, reconfiguration and recovery are continued methods for one failure. Each message is represented as: m(i,j,k) where I = source measure, j = destination measure, k = transmission order. Finally, cycles generate the communicator utilizing a powerfully launching cycle of the application. The cycle is re-launching utilizing MPI_Comm_spawn primal. These tasks are assemblages, subsequently relies upon the MPI library execution and its presentation are connected on the size of the applications measures.

There is one cycle running for each hub P(0…3), predefined checkpoint invocation are arranged too, called default checkpoints. The FADRM continually screens n the message logs utilization. During the application execution, P3 distinguishes that the buffering arrives at as far as possible, henceforth plan a programmed checkpoint invocation. The checkpoint is done in the following regular synchronization purpose of the application execution. To explain usefulness of the Sender-Based Message Logger, the semi-coordinated convention execution is utilized. The logging technique is done as follows: Each cycle (P1..PN) of the equal application produces m messages, explained as: m (i,j,k), where I = source measure, j = destination measure and k = is the send grouping number. The message starts a variety of connected records in the hub unstable memory. The structure permits saving each extrovert message from the cycles. For instance, P2 first send m (2, 1, 1) to P1, and the logs stores it in linked list of array of the exhibit V1. When P2 sends m (2, 1, 2), it is likewise included the linked list of array V1, placing the message in the tail of the linked list etc. for each extrovert message. It is represented that message m (2, 3, 1) and m (2, 3, 2) are not storing in the logs since they are between hub correspondences of hub Ni. Each time a recovery line is acquired, a memory cleaning is executed. A FADRM is needed to continually screen FT resource use. For e.g., application cycles might be failures because of high memory utilization during FT assurance, causing the cycles stall. The activities are performed during the failure discovery, reconfiguration and recovery. The spawned cycle, recognizes that it has been re-launched and load the checkpoint record. After the checkpoint record is stacked, the factors in the process are set as they were in the last checkpoint earlier faults. The cycle hops to the right execution line so as to proceed with the application execution. The messages are expended from the message logs to complete the re-deployment. Then, non-failure cycles can proceed with their execution. The proposed FADRM pseudo code is explained in detail:

Input: Jobs, resource (VM, memory, file size, bandwidth, datacenter) and node

Output: Fault prevented Node (FN), Makespan (MS), Fault Rate (FR), Failure Delay (FD),

Performance Improvement rate (PIR)

Procedure:

Initiate VM;

Allocate the tasks;

Select the available resource;

Apply FADRM method;

Execute the task;

Wait for an expected period of time;

Start function to verify

If a process is running task and node

Verify running (node) and task status;
Connect to node
Return node is working condition and task is executed
End function
While tasks running = true do
Wait upto expected time (frequently monitoring)
Executing = verify_running(node) and task status
If (execution /= true) then
F N = neighbor_node identified;
Return F N;
MakeSpan(MS), Fault Rate(FR), Failure delay(FD), Performance ImprovementRate(PIR)
End if
End while

4 Results and Discussions

4.1 Deployment Setup

The deployment setup is executed on Intel Core i5 (7th generation), 8GB RAM of DDR4 500 GB memory with windows 8 operating systems and CPU 2.70GHz. The proposed algorithm prototype model is developed JAVA programming languages', JDK (Java Development Kit) 1.8, NetBeans 8.0.2 and MSQL 5.6 integrated database.

4.2.1 Input Configurations

The deployment setups are appropriated to execute to the analysis for assessing the proficiency of proposed methodologies whose input details are given in Tab. 1.

4.2 Simulation Results

This section presents experimental outcomes obtained applying FADRM to give Fault Tolerance in the application-level. The outcome shows its programmed usefulness and confirms the usefulness of the Fault Aware Dynamic Resource Manager (FADRM) in real time deployment environments. The deployments were created in controlled environments utilizing and infusing faults. Executions are estimated in any event 3 to 5 multiple times, except if it is unequivocally determined in an unexpected way, and estimations are taken utilizing the time, system devices.

4.2.1 Makespan Time (MS)

Makespan is determined by a virtual machine as total time consumption to finish the complete task as a fixed schedule. Makespan chooses the resources for the total machine (virtual machine, RAM, bandwidth capacity, and memory) which pursues finishing all jobs execution. Makespan is described in Eq. (1) and (2)

$$MakeSpan = \max\{MS_{rj}\} \tag{1}$$

$$MS_{rj} = \sum\nolimits_{T_i \in \theta_{rj}} EET_{T_i r_j} \tag{2}$$

Table 1: Cloud experimental details

Parameters	Value
Number of Jobs	10–100 & 50–500
Virtual Machine	15–25
Cloud users	5–10
Brokers	5–10
Processing of virtual resources	10,000–20,000MIPS
Tasks data Size	800,000–900,000 MI
Datacenter	02DC(1-3hosts,-02-2hosts)-05DC(each-03,taols-15 hosts)
File Size	600–1000
RAM	512–2048
Bandwidth	10,000 MBPS
Data Centre VM	Xen
No. of Process Machine	4
No. of Running Deployment	60
DC VM Policy	Time Shared
DC OS	Linux
VM Memory	1,000,000
DC Architecture	X86

4.2.2 Fault Rate (FR)

The fault rate (FR) is the proportion of aggregate of failure jobs in the proposed technique to the total amount of failure tasks in the other scheduling method. The proposed FADRM method will be improved if the estimation of FR turns out to minimal value other existing method which is derived in Eq. (3).

$$FR = \frac{\sum_{0}^{n} Total\_Fault\_FADRM\_method}{\sum_{0}^{n} Total\_existing\_method} \times 100 \tag{3}$$

4.2.3 Failure Delay (FD)

Failure delay (FD) is explained as the proportion of time deferral or disruption as outcome, because of fault to-fault free job execution time, mean over the total amount of other existing method of tasks. The FD of the proposed Fault Aware Dynamic Resource Manager (FADRM) should be minimal of other existing scheduling methods for the evaluation and estimated which derived in Eq. (4)

$$FD = \frac{Time_{Fault\_to\_fault-free\_task\_execution\_time}}{Overall\_Total\_Task} \tag{4}$$

4.2.4 Performance Improvement Rate (PIR)

Performance improvement rate (PIR) is characterized as the level of execution improvement (or decrease in makespan) for the proposed technique with respect to different techniques and is determined utilizing condition 5

$$PIR = Makespan(Exisitng\_Method) - Makespan(Proposed\_Method) \times \frac{100}{Makepan(Proposed\_Method)} \quad (5)$$

Fault Aware Dynamic Resource Manager (FADRM) presents enhancements model to the Multi-stage Resilience Manager so as for helping the application-level FT scheme. Tab. 2 displays with MTCT(Min–min based time and cost exchange of) [01], MINMAX [01], ACO (Ant colony optimization) [01] NSGAII [01], and DCLCA (dynamic clustering league championship Algorithm) [01] existing techniques on MakeSpan (MS) time, Fault Rate (FR), Failure Delay (FD) and Performance Improvement Ratio (PIR) for 25, 50, 75 and 100 Tasks. Where, FADRM method is evaluated DCLCA method for fault tolerance awareness to address cloud task deployment which should replicate the current accessible resource and minimize the premature fault of autonomous tasks. However, the dynamicity of accessible cloud resource isn't viewed as when integrating on scheduling choices. Where, Fault Aware Dynamic Resource Manager (FADRM) influences FTM consolidating the application-layer checkpoints with a message logs to execute the uncoordinated and semi-coordinated conventions. The scheme additionally incorporates a unique FT resource manager. The deployment and resources of proposed method is finished utilizing a Sender-Based Message Logger and ULFM (user level fault management) augmentation. For parallel applications, proposed a checkpointing services for cloud environments. Based on tabular result, Fault Aware Dynamic Resource Manager (FADRM) reduce on 157.5 MakeSpan (MS) time, 0.38 Fault Rate (FR), 0.25 Failure Delay (FD) and improves 5.5 Performance Improvement Ratio(PIR) for 25, 50, 75 and 100 tasks compare than existing methodologies.

Table 2: MakeSpan (MS) time, Fault Rate (FR), Failure Delay (FD) and Performance Improvement Ratio (PIR) for 25, 50, 75 and 100 Tasks

	25				50				75				100			
Technique	MS	FR	FD	PER	MS	FR	FD	PER	MS	FR	FD	PER	MS	FR	FD	PER
MTCT	850	0.49	0.29	27	1300	0.40	0.33	40	2200	0.30	0.42	52	3100	0.25	0.65	75
MAXMIN	800	0.45	0.27	29	1100	0.35	0.32	43	1900	0.28	0.40	59	2500	0.23	0.45	80
ACO	750	0.40	0.25	33	900	0.34	0.33	51	1750	0.27	0.35	64	2000	0.21	0.40	85
NSGA-II	610	0.37	0.23	35	800	0.32	0.28	53	1570	0.24	0.30	72	1700	0.20	0.37	90
DCLCA	570	0.31	0.20	38	750	0.30	0.26	60	900	0.23	0.28	80	1000	0.18	0.35	94
FADRM	490	0.25	0.17	41	600	0.25	0.23	65	700	0.21	0.25	90	800	0.16	0.32	98

According to Figs. 2–5, Fault Aware Dynamic Resource Manager (FADRM) performs MakeSpan (MS) time, Fault Rate (FR), Failure Delay (FD) and improves the Performance Improvement Ratio(PIR) for 100, 200, 300 and 500 Tasks compare than existing methodologies. Fault Aware Dynamic Resource Manager (FADRM) is estimated with MTCT [01], MINMAX [01], ACO [01] NSGAII [01], and DCLCA [01] existing methodologies. Fault Aware Dynamic Resource Manager (FADRM) is evaluated with MTCT [01], MINMAX [01], ACO (Ant colony optimization) [01] NSGAII [01], and DCLCA [01] existing methodology. Where, nearest competitor is DCLCA [01] that is presented for fault tolerance awareness to address cloud task deployment which should replicate the current accessible resource and minimize the premature fault of autonomous tasks. However, the dynamicity of accessible cloud resource isn't viewed as when integrating on scheduling choices. Where, Fault Aware Dynamic Resource Manager (FADRM) influences FTM consolidating the application-layer checkpoints with a message logs to execute the uncoordinated and semi-coordinated conventions. The scheme additionally incorporates a unique FT resource manager. The deployment and resources of proposed method is finished utilizing a Sender-Based

Message Logger and ULFM (user level fault management) augmentation It is conceivable to see, that the time and memory utilization becomes because of the accessibility of the resources, and when a checkpoint is taken, the memory buffers are delivered. NSGA-II [01] is addressed to interpret the adaptation of fault tolerance issues. The NSGA-II method depends on the Pareto predominance relationship, giving no particular optimal outcome. However, a lot of results are not subject on to each other during dynamic resource changes. ACO [1] represented intelligent optimization technique for dealing with the approaching mapping tasks and resources. It is arbitrary optimization search approach that will be utilized for assigning the mapping tasks to the VMs. However, the dynamic scheduling for this method isn't a need in accomplishing the objective of fault tolerance. Max-Min [01] is resource distribution and scheduling method which is utilized in cloud and in grid computing to reduce the MakeSpan, cost and maximizes benefit and resource usage. This is finished by choosing a task in the task list with the highest execution time on a resource that can deploy it inside a minimum time period. But, a Max-Min technique endures with the production of higher makespan; poor resource utilization and unbalance load issues. MTCT [01] explained for multi-target work process scheduling to help fault recovery in cloud. The MTCT method was evaluated by use of various true logical work processes with organization measure. MTCT technique is significant for genuine work process when both of the two optimization targets are impressive. But, being a multi-objective algorithm, the MTCT is inherently inclined to improve thought into different parameters.

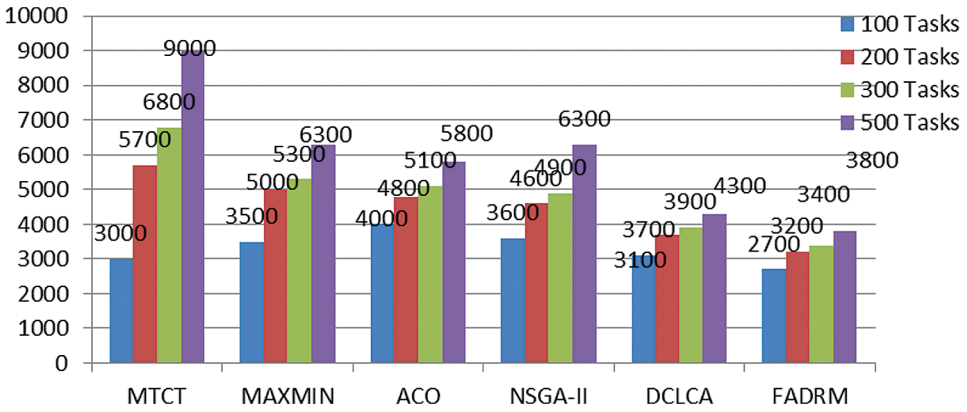

Figure 2: MakeSpan(MS) time for 100, 200, 300 and 500 Tasks

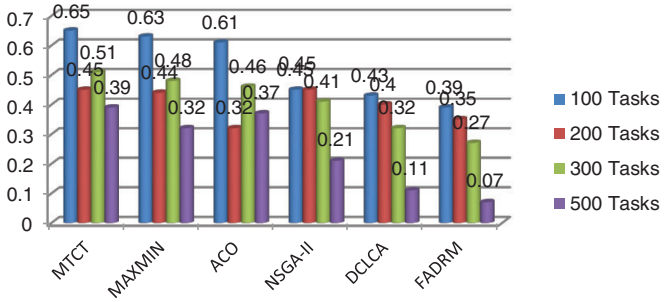

Figure 3: Failure Ratio (FR) for 100, 200, 300 and 500 Tasks

Fault Aware Dynamic Resource Manager (FADRM) continually screens the memory accessibility, and in this specific case, plays out no activities, since it recognizes that there is sufficient memory for the logs, prioritizing server performance for the FT security. When the deployment without the Fault Aware Dynamic

Resource Manager (FADRM) begins utilizing SWAP memory zone of the framework, the Performance improvement rate (PIR) definitely drops, making the entire application crash. Simultaneously, Fault Aware Dynamic Resource Manager (FADRM), hopefully invoke the checkpoints, and after their execution, the memory buffers that is utilized for the logger facility. It is delivered for permitting the constant execution of the application by the principle memory accessible for the application measures. The scenario is intended to utilize the low resource cluster and an application with high FT resource request, to imagine the conduct of the application with and without the Fault Aware Dynamic Resource Manager (FADRM). It is one reason for a few checkpoints creation by the dynamic resource manager, so as to proper deployment, and conveys the application results. FADRM doesn't involve in the user characterized checkpoint schedule, permit the user the control of the checkpoint schedule. However, it might perform hopeful checkpoints, to free resource for the application. So as to maintain a strategic distance from free memory ran out due to FADRM assurance task. Fault Aware Dynamic Resource Manager (FADRM) is tended to with FT. It deals with every hub of the application execution. FADRM continually checks the as of now distributed memory of the variety of records structure utilized for message logging purposes. Proposed scheme permits the application to proceed with the execution. According to graphical results, Fault Aware Dynamic Resource Manager (FADRM) minimizes on 475 MakeSpan (MS) time, 0.40 Fault Rate (FR), 1.30 Failure Delay (FD) and improves 6.75 Performance Improvement Ratio (PIR) for 100, 200, 300 and 500 tasks than conventional methodologies.

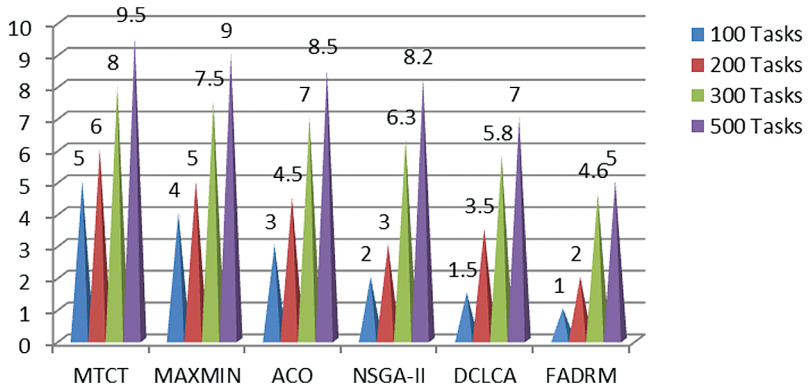

Figure 4: Failure Delay (FD) for 100, 200, 300 and 500 Tasks

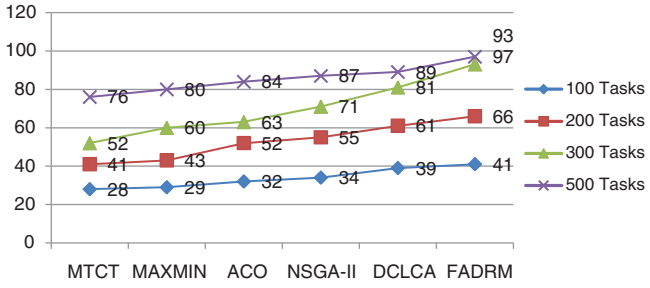

Figure 5: Performance Improvement Rate (PIR) for 100, 200, 300 and 500 Tasks

5 Conclusion

The article presented Fault Aware Dynamic Resource Manager (FADRM) for enhancing the model to the Multi-stage Resilience Manager so as to help an application-level FT scheme. It influences FT

consolidating the application-layer checkpoints with a message logs to deploy the un-coordinated and semi-Coordinated mechanisms. FADRM use the FT to application-level, off programmed and straightforward systems for recovery applications if there should arise an occurrence of faults which plays out any activity when fault is identified. The scheme utilizes semi-coordinated and rollback-recovery conventions. FADRM integrates application-level checkpoints with a sender-based message logs for discovery and recovery purposes. FADRM is included, which plays out the observing of principle memory usage for the message logs, permitting distinguishing when its utilization is arriving at a limit. With this information, it invokes modified designated checkpoints, in an optimistic way, which grants cradle memory upholds used for the message logs for preventing the slowdown stall of the application deployments. FADRM gives novel mechanisms in the application-layer; allowing users to derive only the essential confirm information for their applications.

Besides, a Fault Aware Dynamic Resource Manager (FADRM) is fit to FTM, which screens FT resources usage and perform activities when the utilization arrives at limits. Proposed FADRM minimizes 157.5 MakeSpan (MS) time, 0.38 Fault Rate (FR), 0.25 Failure Delay (FD) and improves 5.5 Performance Improvement Ratio (PIR) for 25, 50, 75 and 100 tasks and 475 MakeSpan (MS) time, 0.40 Fault Rate (FR), 1.30 Failure Delay (FD) and improves 6.75 improves Performance Improvement Ratio (PIR) for 100, 200, 300 and 500 Tasks compare than existing methodologies.

In future, Fault Aware Dynamic Resource Manager (FADRM) can be applied in fog computing environments. Where, there are hues number faults are found in nodes and routes during data transmission. Once, resource is updated dynamically then there is not fully control on data transmission reliability.

Acknowledgement: The authors like to thank the Doctoral panel members from Anna University, Chennai, for their important info and input. Interim, the authors like to extend out their generous gratitude to the in-charge of Research Center, RMK Engineering College too for yielding the assets.

Funding Statement: The authors have not received explicit funding for this research work.

Conflicts of Interest: The creators proclaim that they do not have any conflicts of interest to report with respect to the current study.

References

[1] A. Latiff, M. Shafie, S. H. H. Madni and M. Abdullahi, "Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm," *Neural Computing and Applications*, vol. 29, no. 1, pp. 279–293, 2018.

[2] K. Priti and P. Kaur, "A survey of fault tolerance in cloud computing," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 10, pp. 1–12, 2018.

[3] R. Jhawar and V. Piuri, "Fault tolerance and resilience in cloud computing environment," *Computer and Information Security Handbook*, 2nd Edition, J. Vacca (ed.), Morgan Kaufmann 2013, pp. 165–181, 2017.

[4] A. Eman, M. Elkawkagy and A. El-Sisi, "A reactive fault tolerance approach for cloud computing," in *13th Int. Computer Engineering Conf. (ICENCO)*. IEEE, pp. 190–194, 2017.

[5] J. K. R. Sastry, K. Sai Abhigna, R. Samuel and D. B. K. Kamesh, "Architectural models for fault tolerance within clouds at infrastructure level," *ARPN Journal of Engineering and Applied Sciences*, vol. 12, no. 11, pp. 3463–3469, 2017.

[6] A. Hamid, C. Pahl, G. Estrada, A. Samir and F. Fowley, "A fuzzy load balancer for adaptive fault tolerance management in cloud platforms," in *European Conf. on Service-Oriented and Cloud Computing*. Cham: Springer, pp. 109–124, 2017.

[7] M. Bashir, M. Kiran, I. U. Awan and K. M. Maiyama, "Optimising Fault Tolerance in Real-Time Cloud Computing IaaS Environment," in *2016 IEEE 4th Int. Conf. on Future Internet of Things and Cloud (FiCloud)*. IEEE, pp. 363–370, 2016.

[8] X. Zhanyang, Y. Zhang, H. Li, W. Yang and Q. Qi, "Dynamic resource provisioning for cyber-physical systems in cloud-fog-edge computing," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 1–16, 2020.

[9] C. T. Jonathan and G. C. Hua, "Resource reliability using fault tolerance in cloud computing," in *2nd Int. Conf. on Next Generation Computing Technologies (NGCT)*. IEEE, pp. 65–71, 2016.

[10] G. Sam and A. Bhardwaj, "Efficient fault tolerance on cloud environments," *Int. Journal of Cloud Applications and Computing (IJCAC)*, vol. 8, no. 3, pp. 20–31, 2018.

[11] K. Vinay and S. M. D. Kumar, "Fault-tolerant scheduling for scientific workflows in cloud environments," in *IEEE 7th Int. Advance Computing Conf. (IACC)*. IEEE, pp. 150–155, 2017.

[12] Z. Wei, X. Chen and J. Jiang, "A multi-objective optimization method of initial virtual machine fault-tolerant placement for star topological data centers of cloud systems," *Tsinghua Science and Technology*, vol. 26, no. 1, pp. 95–111, 2020.

[13] L. Jialei, S. Wang, A. Zhou, S. A. P. Kumar, F. Yang *et al.,* "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Trans. on Cloud Computing*, vol. 6, no. 4, pp. 1191–1202, 2016.

[14] P. Deepak, M. A. Salehi, K. Ramamohanarao and R. Buyya, "A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments," in *Software Architecture for Big Data and the Cloud*. Morgan Kaufmann, pp. 285–320, 2017.

[15] S. Sampa, B. Sahoo and A. K. Turuk, "RT-PUSH: a VM fault detector for deadline-based tasks in cloud," in *Proc. of the 3rd Int. Conf. on Communication and Information Processing*, pp. 196–201, 2017.

[16] H. Sajjad and B. Nazir, "Fault tolerance in computational grids: Perspectives, challenges, and issues," *SpringerPlus*, vol. 5, no. 1, pp. 1–20, 2016.

[17] I. Hajara, A. E. Ezugwu, S. B. Junaidu and A. O. Adewumi, "An improved ant colony optimization algorithm with fault tolerance for job scheduling in grid computing systems," *PLoS One*, vol. 12, no. 5, pp. e0177567, 2017.

[18] R. M. Sri and P. Chawla, "A survey on QOS and fault tolerance based service scheduling techniques in fog computing environment, " in *7th Int. Conf. on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, pp. 365–372, 2018.

[19] T. Z. Rampratap, "Modeling for fault tolerance in cloud computing environment," *Journal of Computer Sciences and Applications*, vol. 4, no. 1, pp. 9–13, 2016.

[20] R. Archana, "A survey of fault tolerance in cloud computing," *International Journal of Arts, Science and Humanities*, vol. 6, no. S1, pp. 98–104, 2018.