Adaptive Fault Tolerance in Real Time Cloud Computing

Sheheryar Malik, Fabrice Huet

2011 IEEE World Congress on Services

Adaptive Fault Tolerance in Real Time Cloud Computing

Sheheryar Malik
Research Team OASIS
INRIA - Sophia Antipolis
06902 Sophia Antipolis, France
Email: sheheryar.malik@inria.fr

Fabrice Huet
Research Team OASIS
INRIA - Sophia Antipolis
06902 Sophia Antipolis, France
Email: fabrice.huet@inria.fr

Abstract -- With the increasing demand and benefits of cloud computing infrastructure, real time computing can be performed on cloud infrastructure. A real time system can take advantage of intensive computing capabilities and scalable virtualized environment of cloud computing to execute real time tasks. In most of the real time cloud applications, processing is done on remote cloud computing nodes. So there are more chances of errors, due to the undetermined latency and loose control over computing node. On the other side, most of the real time systems are also safety critical and should be highly reliable. So there is an increased requirement for fault tolerance to achieve reliability for the real time computing on cloud infrastructure. In this paper, a fault tolerance model for real time cloud computing is proposed. In the proposed model, the system tolerates the faults and makes the decision on the basis of reliability of the processing nodes, i.e. virtual machines. The reliability of the virtual machines is adaptive, which changes after every computing cycle. If a virtual machine manages to produce a correct result within the time limit, its reliability increases. And if it fails to produce the result within time or correct result, its reliability decreases. A metric model is given for the reliability assessment. In the model, decrease in reliability is more than increase. If the node continues to fail, it is removed, and a new node is added. There is also a minimum reliability level. If any processing node does not achieve that level, the systems will perform backward recovery or safety measures. The proposed technique is based on the execution of design diverse variants on multiple virtual machines, and assigning reliability to the results produced by variants. The virtual machine instances can be of same type or of different types. The system provides both the forward and backward recovery mechanism, but main focus is on forward recovery. The main essence of the proposed technique is the adaptive behavior of the reliability weights assigned to each processing node and adding and removing of nodes on the basis of reliability.

Keywords: fault tolerance, reliability, cloud computing

I. INTRODUCTION

Cloud computing is really changing the way, how and where the computing is going to be performed. It has gained a lot of attention to be used as a computing model for a variety of application domains. But the people are still reluctant to use it for real time applications. But now some researchers and cloud vendors are working to give the power of cloud and associated benefits to the real time applications. A few cloud operators has started real time cloud support.

Cloud support for real time system is really important. Because, today we found a lot of real time systems around us. Their applications range from small mobile phones to larger industrial controls and from mini pacemaker to larger nuclear plants. Most of them are also safety critical systems, which should be reliable. In general, real-time system is any information processing system which has to respond to externally generated input stimuli, within a finite and specified period of time [12]. So the correctness depends not only on the logical result, but also the time it was delivered [8]. Failure to respond is as bad as the wrong response [13]. These systems have two main characteristics by which they are separated by other general-purpose systems. These characteristics are timeliness and fault tolerance [6]. By timeliness, we mean that each task in real time system has a time limit in which it has to finish its execution. And by fault tolerance means that it should continue to operate under fault presence [7].

Use of cloud infrastructure for real time applications increases the chances of errors. As the cloud nodes (virtual machines) are far from the transceiver (job submitting node, actuator or sensor). Many real time systems are also safety critical systems, so they require a higher level of fault tolerance [14]. Safety critical real time systems require working properly to avoid failure, which can cause financial loss as well as casualties [10]. So there is an increased need to tolerate the fault for such type of systems to be used with cloud infrastructure. For this purpose we had presented a model for the fault tolerance of real time applications running at cloud infrastructure.

The basic mechanism to achieve the fault tolerance is replication or redundancy [7]. We had performed this replication in form of software variants running on multiple virtual machines. Due to the replication, cost for renting the cloud resources will increase. But it is really required to avoid the catastrophic loss.

II. RELATED WORK

The use of cloud infrastructure for real time computing is quite new. Most of the real time applications require the

978-0-7695-4461-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SERVICES.2011.108

fault tolerance capability to be provided. A lot of work has been done in the area of fault tolerance for standard real time systems. But there is lot of research room available in fault tolerance of real time application running on cloud infrastructure. Cloud infrastructure has introduced some new issues related to real time computing. In cloud, the latency of nodes (virtual machines) is unknown. Even if one determines the latency, it can change over the period of time [5]. Generally, IP is not associated to a particular instance [2]. Node is virtual and computation can be migrated from one virtual machine instance to another by keeping the same IP [1]. The user of real time cloud applications has loose control over the nodes. He does not know where his application is going to be processed [6]. But on the brighter side, cloud has a facility to scale up dynamically. So the faulty node (node represents itself and the communication link) can be removed. A new node can be added, if required. These characteristics are different from the existing traditional distributed real time systems.

X. Kong et. al. [1, 5] presented a model for virtual infrastructure performance and fault tolerance. But it is not well suited for the fault tolerance of real time cloud applications.

For the non-cloud applications, a baseline model for distributed RTS is, distributed recovery block [11] proposed by K. H. Kim which is very basic in nature. Another model, "A formal approach for the fault tolerance of distributed real time system (RTS)" is being proposed by J. Coenen and J. Hooman [7]. Another model is "time stamped fault tolerance of distributed RTS" [9], which is proposed by S. Malik and M. J. Rehman. This model incorporated the concept of time stamping with the outputs. All of these models were defined for the real time systems based on standard computing architecture. No one has introduced the fault tolerance on the basis of reliability of node and reliability assessment of node.

Our proposed model is based upon adaptive reliability assessment of virtual machines in cloud environment and fault tolerance of real time applications running on those VMs. This fault tolerance has to be done on the basis of the reliability of virtual machines.

III. PROPOSED MODEL (AFTRC)

A scheme is devised here which is for the fault tolerance of real time applications running on cloud infrastructure. The model name is **A**daptive **F**ault **T**olerance in **R**eal-time **C**loud computing (AFTRC). This scheme tolerates the faults on the basis of reliability of each computing node, i.e. virtual machine. A virtual machine is selected for computation on the basis of its reliability and can be removed, if does not perform well for real time applications. The model is shown in figure 1.

In this model, we have two main types of node. One type is a set of virtual machines, running on cloud infrastructure, and the other is the adjudication node. Virtual machine contains the real time application algorithm and an acceptance test for its logical validity. On the adjudicator, we have the time checker, reliability assessor and decision mechanism modules. The location of adjudication node depends on the type of the real time applications and the scenario in which they are used. It can be a part of the cloud infrastructure or can be a part of the user infrastructure. Generally, it is placed near to the sensors, actuators, submission node.

In practice, we have N nodes on cloud to execute real time applications. All of these nodes are virtual machines. These virtual machines run the invariant real time application algorithms. These invariant algorithms try to mask the coincident faults. The diverse algorithms running on different virtual machines have approximately equal computation time. Generally, the virtual machines are replicas of each other, but they can be a invariant. Variant virtual machines will provide a higher level of immunity to coincident faults. It is more affordable and comparatively lesser cumbersome to use different environments in cloud as compare to non-virtualized infrastructure. This scheme provides forward recovery as well as optional backward recovery.

In this scheme, we have 'N' virtual machine, which run the 'N' variant algorithms. Algorithm 'X_1' runs on 'Virtual machine-1', 'X_2' runs on 'Virtual machine-2', up till 'X_m', which runs on 'Virtual machine-m'. Then we have AT module which is responsible for the verification of output result of each node. The outputs are then passed to TC module which checks the timing of each result. On the basis of the timing the RA module calculates and reassigns the reliability of each module. Then all the results are forwarded to DM module which selects the output on the basis of best reliability. The output of a node with highest reliability is selected as the system cycle output.

A. Working

As stated earlier, this technique has M nodes (virtual machines). Each node is taking input data from the input buffer. This input is concurrently passed to all the virtual machines, which run diverse software. The reason to run diverse algorithm on each module is to mask the chances of coincident faults. Each node takes the input, executes the application algorithm and produces result. These results are passed to AT module. AT then passes these results to the adjudication node for result decision and reliability assessment. The different modules in the system have the given responsibility.

Real time application Algorithm is the application logic to perform the real time computations. There are variant algorithms running on different virtual machines. These algorithms can be different from each other either by implementation language, or software engineering process, or by functional logic. The algorithm passes the result to the acceptance test for verification.

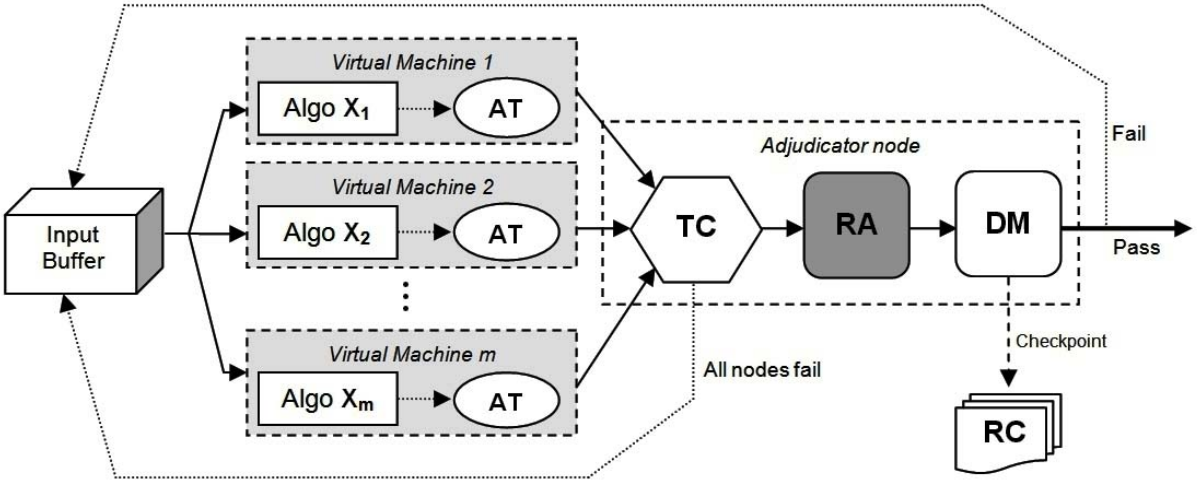

Figure-1: Proposed System Model

Acceptance test (AT) module is provided to each virtual machine. All the AT module are replica of each other. It performs the acceptance test for each cycle output. It verifies the correctness of the result produced by an algorithm running on the same virtual machine. If the result is correct then it passes the result to TC module. If the result is incorrect then it does not pass the result to the TC module. To indicate the failure of result, AT sends a validity exception signal to TC.

Time checker (TC) module is a time checking sensor. It contains a watch dog timer (WDT) which monitors the timing of result produced by each module. TC module passes the results to RA (reliability assessor) module. It only passes the correct results of those nodes which produces the result before deadline time. TC module raises the signal of time-overrun for those modules which produces results after deadline time. If all the nodes fail to produce the result then TC module performs the backward recovery. It also informs the RA module to compute the new reliabilities of all the nodes.

Reliability assessor (RA) module assesses the reliability for each virtual machine. It is the main core module of the proposed system. As the proposed system tolerates the faults and makes the decision on the basis of reliability of the processing nodes (i.e. virtual machine). The reliability of the virtual machine is adaptive, which changes after every computing cycle. In the beginning the reliability of each virtual machine is 100%. If a processing node manages to produce a correct result within the time limit, its reliability increases. And if the processing node fails to produce the correct result or result within time, its reliability decreases. The reliability assessment algorithm is more convergent towards failure conditions. It means that decrease in reliability is more than increase. This comparison is also shown in figure 2 and figure 3. There is also a minimum and maximum reliability level. If reliability of any processing node falls below minimum reliability level, the RA stops that node to work further and removes it. It then adds a new node in place of the removed node. Handling of removal and addition of node is not the responsibility of RA. It only asks to some other responsible system module (resource manager or scheduler, i.e. ProActive resource manager in our case). If reliability of all the nodes fall below minimum reliability level, the system either perform the backward recovery or stop the system to work further. If a system is a safety critical system then it is better to stop the system, otherwise it may permit to perform backward recovery. The outputs are further moved to DM module.

Decision mechanism (DM) selects the final output for a computing cycle. It selects the output of the node which has highest reliability among all the competing nodes. Competing nodes are those nodes which have produced the results within time. We have included only the competing nodes, because a node which fails to produce the result can have higher existing reliability. If two nodes have the same highest reliability level then the output of the node with smaller IP address is selected as computing cycle output. There is a SRL (system reliability level). It is the minimum reliability level to be achieved to pass the result. DM compares the best reliability with the system reliability level. Best reliability level should be greater than or equal to system reliability level. If the node with best reliability does not achieve the SRL the DM raises the failure signal for the computing cycle. In this case, backward recovery is performed. Backward is performed by the help of check point established in recovery cache. DM also request to some responsible module (resource manager or scheduler) to remove one node with minimum reliability and add a new node.

Recovery cache (RC) is a repository area to hold the checkpoints. At the end of each computing cycle DM makes checkpoint in it. In case of a complete failure, backward recovery is performed with the help of checkpoints

maintained in this recovery cache. In our experiment we have done the communication induced checkpoint (CIC) [15, 16]. The CIC perform the check pointing at the end of every cycle to maintain a global state.

This scheme provides an automatic forward recovery. If a node fail to produce output or produce out put after time overrun the system will not fail. It will continue to operate with remaining nodes. This mechanism will produce output until all the nodes fail.

B. Fault Tolerance Mechanism

We apply reliability assessment algorithm for each node (virtual machine) one by one. Initially ***reliability*** of a node is set to 1. There is an adaptability factor ***n***, which controls the adaptability of reliability assessment. The value of *n* is always greater than 0. The algorithm takes input of three factors ***RF***, ***minReliability*** and ***maxReliability*** from configuration file. *RF* is a reliability factor which increases or decreases the reliability of the node. It decreases the reliability of the node more quickly as compare to the increase in reliability. It is due to its multiplication with the adaptability factor *n*. *minReliability* is the minimum reliability level. If a node reaches to this level, it is stopped to perform further operations. *maxReliability* is the maximum reliability level. Node reliability cannot be more than this level. It is really important in a situation, where a initially produces correct results in consecutive cycles, but then fails again and again. So its reliability should not be high enough to make the reliability difficult to decrease and converge towards lower reliability.

The algorithm is normally more convergent to failures in near past. So if there are two nodes and both of them have 10 passes and 10 failures in total 20 cycles. But the node, who have more failures in near past has more chances to have lesser reliability than the other. This factor is really in accordance to latency issues, where initially node latency was good, but then it becomes high. So this node tends to more node failures by failing to produce the results in time.

The values of variables (*RF, minReliability, maxReliability, SRL*) depend on the real time applications. User has to decide how much be the value for each variable. Calculation of these variables is not within the scope of this research.

Reliability Assessment Algorithm:

Begin
Initially reliability:=1, n :=1
Input *from configuration* RF, maxReliability, minReliability
Input nodestatus
if nodeStatus =Pass **then**
reliability := reliability + (reliability * RF)
if n > 1 **then**
n := n-1;
else if nodeStatus = Fail **then**
reliability := reliability – (reliability * RF * n)
n := n+1;

if reliability >= maxReliability **then**
reliability := maxReliability
if reliability < minReliability **then**
nodeStatus :=dead
call_proc: remove_this_node
call_proc: add_new_node
End

Decision Mechanism Algorithm

Begin
Initially reliability:=1, n :=1
Input *from RA* nodeReliability, numCandNodes
Input *from configuration* SRL
bestReliability := find_reliability of node with highest reliability
if bestReliability >= SRL
status := success
else
perform_backward_recovery
call_proc: remove_node_minReliability
call_proc: add_new_node
End

C. Reliability Assessment Impact Analysis

A metric analysis is given for the reliability assessment impact analysis. Here we have analyzed the reliability adaptation for different scenarios for a single virtual machine. We have assumed that the value of reliability factor (RF) is 3% (i.e. 0.03) and total computing cycles are 10. In the beginning, the reliability is 1. In the table-1, a comparison is provided between pass and fail scenario. This comparison is done for 10 computing cycle. In these cycles a node continuously passed and another node continuously failed. The increase in reliability after 10 cycles for VM-1 is 0.3439, whereas decrease in reliability for VM-2 is 0.8439. Here we can see that decrease due to failure is more than increase. And this decrease continues to decrease faster if the node continues to fail.

Table-1: Comparison of Pass & Fail

Cycle	VM-1		VM-2	
	Status	Reliability	Status	Reliability
Start	-	1.0000	-	1.0000
1	Pass	1.0300	Fail	0.9700
2	Pass	1.0609	Fail	0.9118
3	Pass	1.0927	Fail	0.8297
4	Pass	1.1255	Fail	0.7302
5	Pass	1.1593	Fail	0.6206
6	Pass	1.1941	Fail	0.5089
7	Pass	1.2299	Fail	0.4021
8	Pass	1.2668	Fail	0.3056
9	Pass	1.3048	Fail	0.2231
10	Pass	1.3439	Fail	0.1561

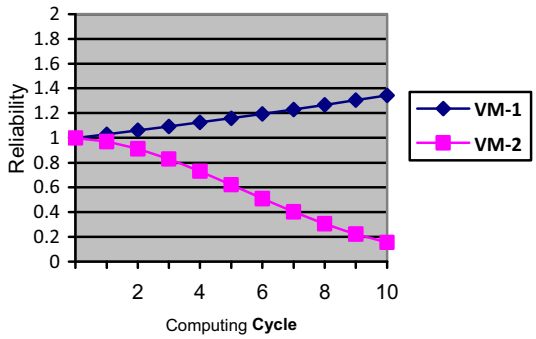

Figure-2: Comparison of two virtual machine nodes

In the table-2 and figure-3 a scenario for a single node is provided. In this a node continues to be successful for first 5 cycles and then fail for 5 times. Here we can see that convergence towards decrement in reliability is much higher.

Table-2: Pass to Fail shifting

Cycle	VM Status	Reliability
Start	-	1.0000
1	Pass	1.0300
2	Pass	1.0609
3	Pass	1.0927
4	Pass	1.1255
5	Pass	1.1593
6	Fail	1.1245
7	Fail	1.0570
8	Fail	0.9619
9	Fail	0.8465
10	Fail	0.7195

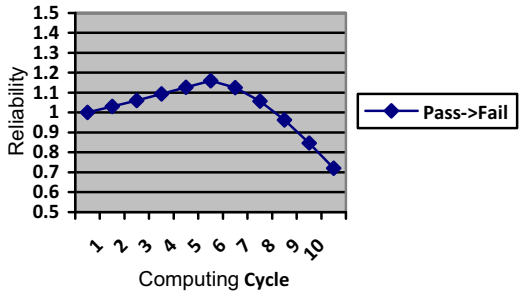

Figure-3: Change in reliability for a single node

D. *Different Scenarios*

Due to the diversity in software and timing constraints, there could be different scenarios for success and failure of results.

1) Complete Failure Free Scenario

All the algorithms on each virtual machine produce the result. Acceptance test pass the results. All the results are produced before time overrun. So TC also clears the results. RA computes and assigns the new reliability weights to each virtual machine. Decision mechanism selects an output from the VM with maximum reliability. No failure or exception occurs in any VM in this case.

2) Partial Failure Scenario – All AT pass, TC passes some Nodes

All the virtual machines produce the correct result. Some results are produced within time and some after time-overrun. All AT pass the results and forward them to the time checker. TC receives result of some virtual machines before time-overrun. It passes them to RM, which assesses their reliability. RM forwards the produced result to the decision mechanism for adjudication. Decision mechanism selects an output from the VM with maximum reliability.

In this scenario, system will continue to operate with forward recovery. Adjudicator selects the output from the subset of the nodes that have produced the correct output within time limit.

3) Partial Failure Scenario – Some AT pass, TC also pass

Some of the virtual machines produce the correct result. These results are produced within time limit. Acceptance test pass only the correct results to the TC. For failed virtual machines, it generates an error signal to TC. Time checker receives result of passed virtual machines before time-overrun. It passes them to RM, which assesses their reliability. RM forwards the produced result to the decision mechanism for adjudication. Decision mechanism selects an output from the VM with maximum reliability. In this scenario, system will continue to operate with forward recovery. Adjudicator selects the output from the subset of the nodes that have produced the correct output within time limit.

4) Failure Scenario – ATs fail, TC fail

In this scenario, either all the AT rejects the result of the algorithms or some AT passes but TC fails to find a single output within time limit. In this case, the cycle fails and TC informs the adjudicator to perform backward recovery. Now backward recovery will be done with the help of checkpoints stored in recovery cache.

5) Failure Scenario – ATs pass, TC pass, DM fail

In this scenario, all or some of the AT passes the results. TC also finds the output within time limit. Reliability assessor computes and assigns the reliability to the virtual machines. But the VM with highest reliability could not achieve the system reliability level (SRL). In this case, DM raises the failure signal for the whole computing

cycle and backward recovery is performed with the help of checkpoint stored in recovery cache.

IV. EXPERIMENTS & RESULTS

We have conducted an experiment using ProActive grid interface to Amazon EC2 cloud. In this experiment, we created three virtual machines and then run our real time algorithm on them. The algorithm has 10 computing cycles. The algorithms consist of a series of tasks. Each of these tasks is performed in one computing cycle. Each virtual machine node runs a diverse algorithm. VirtualMachine-1 runs algorithm X_1, VirtualMachine-2 runs algorithm X_2, and VirtualMachine-3 runs algorithm X_3. The algorithm is for real time financial analysis. Then we have an adjudication node, which is sending the input data and receiving the results from the virtual machines to be processed. This adjudicator is placed at our research center. This node is working both as a adjudicator and a tester for our experiment.

We have done the implementation using ProActive parallel suite's active object model [15]. In this implementation, each virtual machine is behaving like an active object. We have done the check pointing using CIC protocol [16].

Our experiment has the following description of implementation for the proposed model.

Virtual Machine 1:

- Algorithm 'X_1' implementation
- Acceptance Test-I

Virtual Machine 2:

- Algorithm 'X_2' implementation
- Acceptance Test-II

Virtual Machine 3:

- Algorithm 'X_3' implementation
- Acceptance Test-III

Adjudication Node:

There is an adjudicator node, which contains the following modules:

- Input Buffer
- Time Checker;
- Reliability Assessor;
- Decision mechanism;
- Recovery Cache;

Environmental variable:

Values of environmental variables are following;
reliability =1, n = 1, RF = 0.2, SRL = 0.8, maxReliability = 1.2, minReliability = 0.7

In this experiment, we have an input buffer at adjudication or tester node. This input buffer provides the inputs to all the algorithms. At VirtualMachine-1 algorithm 'X_1' started performing the first task. The result of the task is passed to the acceptance test, which verifies the correctness of its result. If result is correct, it was passed to the time checker for the verification of timeliness. The process runs for 10 cycles.

At VirtualMachine-2, same procedure was applied on a design diverse algorithm 'X_2' produced results for the task in each computing cycle. The result of the task was passed to acceptance test at VirtualMachine-2, which verify the correctness of its result. If it found the result to be correct, it was passed to the time checker for the verification of timeliness.

At VirtualMachine-3, same procedure was applied on a design diverse algorithm 'X_3' produced results for the task in each computing cycle. The result of the task was passed to acceptance test at VirtualMachine-3, which verify the correctness of its result and upon success signal forwarded it to the time checker for the verification of timeliness.

Time checker module checked the timeliness of each virtual machine result and then passed the result to the reliability assessor module. RA after assessing and updating reliability of three modules passed the result to the decision mechanism which selected the result with best reliability among all. DM determines the success or failure of a task result in a computing cycle. It also stored checkpoint in case of success. Some experimental details are given in table-3.

Evaluation

In the first cycle, both VirtualMacine-1 and VirtualMachine-3 have the same reliability, but the result of VM-1 has been selected as it has a lower IP address. VM-3 output was selected by DM from cycle 2 to 4, as it has the highest reliability among competing virtual machines. In cycle 5 VirtualMachine-3 still has the highest reliability, but it is not selected. Because its result was not passed by AT and TC, so consequently, it was not among competing virtual machines.

In the case of both AT and TC failure, time is given for the instance, TC has received the error signal from AT. TC status 'fail' does not means that it has received the result after time overrun. If an AT fails to produce a correct result then TC status is also 'fail'. E.g. VirtualMachine-1 has failed in acceptance test in cycle 10. So it has generated an error signal to TC, which received it before time limit. But as there is no result produced, so the status of TC is also 'fail'.

Table-3: Experiment Results for the Reliability Assessment														
Cycle	Real Time Limit (ms)	Virtual Machine-1				Virtual Machine-2				Virtual Machine-3				DM Selected Node
		AT	TC	Time	Relia-bility	AT	TC	Time	Relia-bility	AT	TC	Time	Relia-bility	
Start		-	-	-	1	-	-		1	-	-		1	-
1	2500	Pass	Pass	2174	1.020	Fail	Fail	2997	0.980	Pass	Pass	2238	1.020	VM-1
2	3700	Pass	Fail	3901	1.000	Pass	Fail	-	0.941	Pass	Pass	3599	1.040	VM-3
3	2150	Pass	Fail	3477	0.960	Pass	Pass	2101	0.960	Pass	Pass	2084	1.061	VM-3
4	6300	Pass	Fail	-	0.902	Pass	Pass	5732	0.979	Pass	Pass	6113	1.082	VM-3
5	1950	Fail	Fail	-	0.830	Pass	Pass	1906	0.998	Fail	Fail	1892	1.061	VM-2
6	2800	Pass	Pass	2791	0.846	Pass	Pass	2682	1.018	Fail	Fail	2653	1.018	VM-2
7	2350	Pass	Pass	2269	0.863	Pass	Pass	2312	1.039	Pass	Fail	2771	0.957	VM-2
8	3200	Pass	Pass	3075	0.881	Pass	Pass	3102	1.059	Pass	Fail	-	0.881	VM-2
9	3900	Pass	Pass	3618	0.898	Pass	Fail	3949	1.038	Pass	Pass	3772	0.898	VM-1
10	2650	Fail	Fail	2589	0.880	Pass	Pass	2601	1.059	Pass	Fail	3110	0.826	VM-2

V. DISCUSSION & CONCLUSION

The proposed scheme is a good option to be used as a fault tolerance mechanism for real time computing on cloud infrastructure. It has all the advantages of forward recovery mechanism. It has a dynamic behavior of reliability configuration. The scheme is highly fault tolerant. The reason behind adaptive reliability is that the scheme can take advantage of dynamic scalability of cloud infrastructure. This system takes the full advantage of using diverse software. In our experiment, we have used three virtual machines. It utilizes all of three virtual machines in parallel. This scheme has incorporated the concept of fault tolerance on the basis of VM algorithm reliability. Decision mechanism shows convergence towards the result of the algorithm which has highest reliability.

Probability of failure is very less in our devised scheme. This scheme works for forward recovery until all the nodes fail to produce the result. The system assures the reliability by providing the backward recovery at two levels. First backward recovery point is TC. Here if all the nodes fail to produce the result, it performs backward recovery. Second backward recovery point is DM. It performs the backward recovery if the node with best reliability could not achieve the SRL. There is another big advantage of this scheme. It does not suffer from domino effect as check pointing is made in the end when all the nodes have produced the result.

VI. FUTURE WORK

We are working on some new enhancements to this scheme so that our system should be more fault-tolerant. Major focus in future will be on reliability factor inclusion in more effective decision making.

We are also working on a module name Resource Awareness Module (RAM). It is aimed to help the cloud scheduler for the scheduling decisions on the basis of certain network and infrastructure characteristics. This fault tolerance mechanism is going to be a part of RAM. Initially, RAM is targeted to be integrated with ProActive scheduler. So after this integration ProActive scheduler will do the scheduling on a virtual machine node on the basis of node reliability.

REFERENCES

[1] X. Kong, J. Huang, C. Lin, P. D. Ungsunan, "Performance, Fault-tolerance and Scalability Analysis of Virtual Infrastructure Management System", 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications, Chengdu, China, August 9-12, 2009

[2] J. Barr, A. Narin, "Building Fault-Tolerant applications on AWS", *Amazon Web Services,* http://media.amazonwebservices.com/AWS_Building_Fault_Tolerant_Applications.pdf

[3] Z. Dai, F. Viale, X. Chi, D. Caromel, Z. Lu, "A Task-Based Fault-Tolerance Mechanism to Hierarchical Master/Worker with Divisible Tasks", *2009 11th IEEE International Conference on High Performance Computing and Communication,* Seoul Korea, June 25-27 2009

[4] D. Caromel, C. Delbe, A. D. Costanzo, *Peer-to-Peer and fault-tolerance: Towards deployment-based technical services,* Vol 23, No. 7, August 2007, pp. 879-887

[5] X. Kong, J. Huang, C. Lin, "Comprehensive Analysis of Performance, Fault-tolerance and Scalability in Grid Resource Management System", *2009 Eighth International Conference on Grid and Cooperative Computing*, Lanzhou, China, August 27-29, 2009

[6] W. T. Tsai, Q. Shao, X. Sun, J. Elston, "Real Time Service-Oriented Cloud Computing", *School of Computing, Informatics and Decision System Engineering Arizona State University USA*, http://www.public.asu.edu/~qshao1/doc/RTSOA.pdf

[7] J .Coenen, J. Hooman, "A Formal Approach to Fault Tolerance in Distributed Real-Time Systems", *Department of Mathematics and Computing Science, Eindhoven University of Technology, Nether land*

[8] O. S. Unsal, I. Koren, M. Krishna, “Towards Energy-Aware Software Based Fault Tolerance in Real Time Systems” *ISLPED ’02*, Monterey, California, USA, August 12-14, 2002,

[9] S. Malik, M. J. Rehman, “Time Stamped Fault Tolerance in Distributed Real Time Systems”; *IEEE International Multi-topic Conference, Karachi, Pakistan,* 2005

[10] L. L. Pullum, “Software Fault Tolerance and Implementation” Artech House, Boston, London, United Kingdom, 2001

[11] K. H. Kim, “Structuring DRB Computing Stations in Highly Decentralized Systems,: *Proceedings International Symposium on Autonomous Decentralized Systems*, Kawasaki, 1993, pp. 305-314

[12] J. Stankovic, “Misconceptions About Real-Time Computing,” *IEEE Computer*, Vol. 21, No. 10, October 1988, pp. 10-19.

[13] K. H. Kim, “Towards Integration of Major Design Techniques for Real-Time Fault-Tolerant Computer System”, *Society for Design & Process Science*, USA, 2002

[14] K. H. Kim, “Distributed Execution of Recovery Blocks: An Approach to Uniform Treatment of Hardware & Software faults.” *Proceeding fourth International Conference on Distributed Computing Systems*, 1984, pp. 526-532

[15] D. Caromel, C. Delbe, A. D. Costanzo, M. Leyton, *ProActive: An Integrated Platform for Programming and Running Applications on Grids and P2P Systems,* Computational Methods in Science and Technology 12(1), pp 69-77, 2006

[16] “ProActive Service and Advanced Feature”, INRIA – University of Nice Sophia Antipolis, http://proactive.inria.fr/trunk/Programming/AdvancedFeatures/multiple_html/FaultTolerance.html