MODELING FOR FAULT TOLERANCE AND SCALABILITY IN CLOUD ENVIRONMENT

Rajeswari Sana[1*], B. Harika[2], E susmitha[3], M. Himabindu[4], Mannuru shalima sulthana[5], P. Santhosh Kumar[6]

[1,3,4,5,6] Department of Computer Science and engineering, RGUKT, RK Valley, IIIT Idupulapaya, Vempalli, Andhra Pradesh 516330, Andrapradesh, India.

[2]Department of Information technology, Government polytechnic college, Obulavaripalli, Kadapa.

*Corresponding author: **srajeswari@rguktrkv.ac.in**

Abstract:
Cloud computing is in-demand service evolution in computing paradigm of huge scale distributed computing. It is a versatile technology to furnish integration of resources and software which are highly scalable and regularly are prone to failure. Fault Tolerance is involved with all the methods to permit the system to tolerate software and additionally hardware faults left over in a system after its advancement. Fault Tolerance is the capacity to tolerate the faults that are performed by the user by mistake. The scalability is the capacity to scale on-demand services and facilities when recommended by the user. The scaling is beyond the limits. Cloud middleware is designed on the principle of scalability through different dimensions like performance, size and load, size and performance. The cloud middleware deals a huge number of users and resources which depend on the cloud to achieve premises without affording the maintenance and administrative costs. So the capability to tolerate is common for offering optimized and efficient system. It is challenging in cloud computing to improve highly scalable tolerance systems that will offer competitive performance. The main objective of this paper is to investigate the fault tolerance model for cloud computing and the fault tolerance techniques in a cloud environment. Achieves low end-to-end latency and offering application transparency to the cloud middleware maintains strong replica consistency.

Keywords: Fault Tolerance, Fault tolerance, Model, Reliability, Scalability

1. INTRODUCTION

Cloud computing main objective is to give reliable services inside data centers that include networks, storage, and servers. The services are supply to the consumers transparently without their requirement to know the fundamental hardware and software. Major difficulties of cloud computing are to ensure that the applications execute without a hiatus in the services they give to the consumers. The LLFT (low latency fault tolerance) middleware gives fault tolerance for distributed applications deployed inside a data center environment or cloud computing, as a service provided by the owners of the cloud. The low latency of false tolerance replicates the process of the applications, using the follower/leader replication method [1]. The replicas of a process constitute a process group and the process group that give a service to the consumers constitute a service group, as shown in Figure 1.

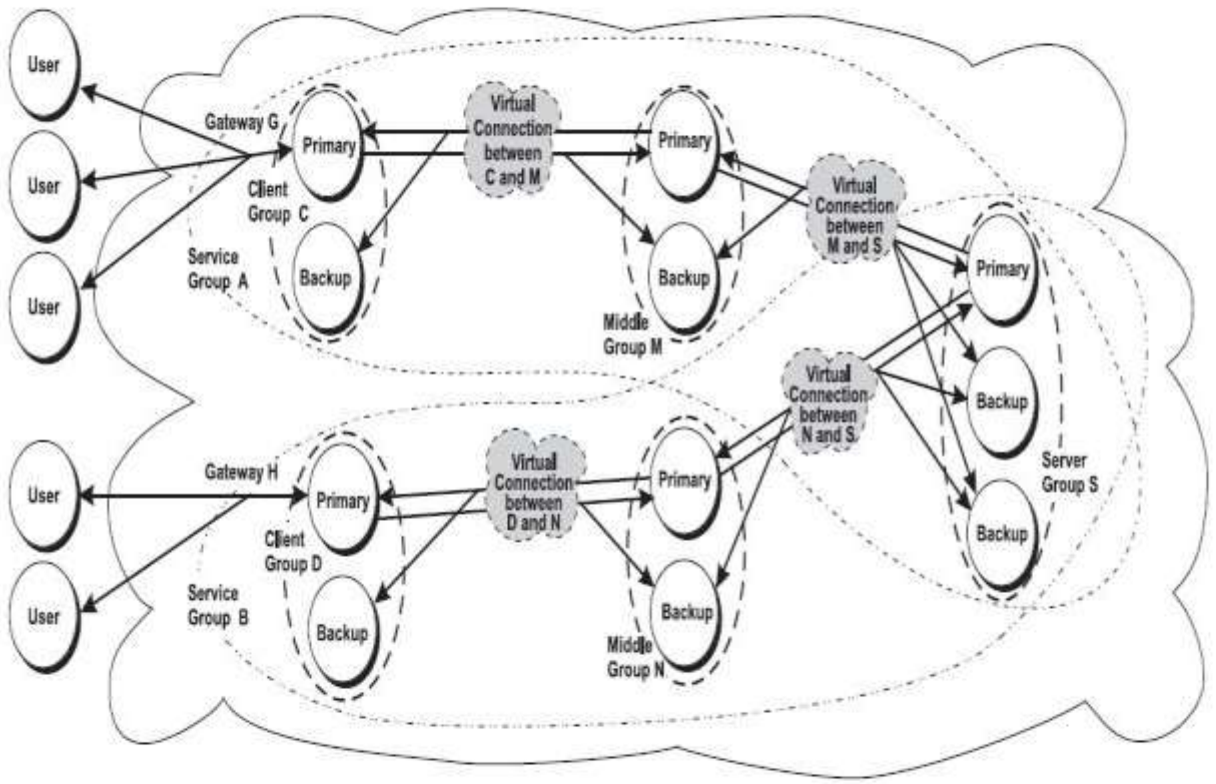

Fig.1. Multiple process groups interacting over virtual connections in a cloud.

Within a process group, one replica is designated as the primary, and the other replicas are the backups. The primary process group multicasts messages to a destination process group over a virtual connection. The primary in the destination process group orders the messages performs the operations and produces ordering information for non-deterministic operations, which it supplies to the backups. The Cloud model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) is. The Cloud As a new computing model was created with the aim to provide reliable, customized and quality of service (QoS) Calculation of dynamic environments for end-users to have secured. In general, the cloud computing New aspects in the management of computing resources for future brings Infinite computing resources available on demand for end-user perspective, zero commitment, front view Members of the clouds; and short-term use of the computational resource of the high - end. Because of the nature of the application, the application is important to secure clouds. The fundamental principle of cloud computing is that the user information is not stored locally but the Internet is stored in the data centre. Thus the need to protect data among processes is unreliable.

Users can store data at any time using the application programming interface (API) by the cloud providers can access from any terminal equipment connected to the Internet. Not only is the storage services provided by hardware and software services to the public vote and is available in the business market. Services provided by the service providers can be everything from infrastructure, platform and software resources of thebe. Each of these services to the Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS) is called. Hence, fault tolerance and scheduling are two important parameters for reliable system nodes. Due to the vast complexities of cloud, to provide a comprehensive solution for cloud security will be difficult. Cloud computing environment in two basic characteristics timeliness and fault tolerance is able to work. With timeliness, we mean that any work in real time, a time limit is That Run it will end and fault tolerance means that work must continue to error. Cloud support is really important for real-time systems. In general, a real-time system is an information processing system in response to external stimuli input production, in a limited and specified period of time. In cloud computing, the use of processor nodes, and also because it raises the probability of error in terms of safety-critical real-time systems should increase their reliability and hence the demand for achieving fault tolerant systems in real-time systems to enhance.

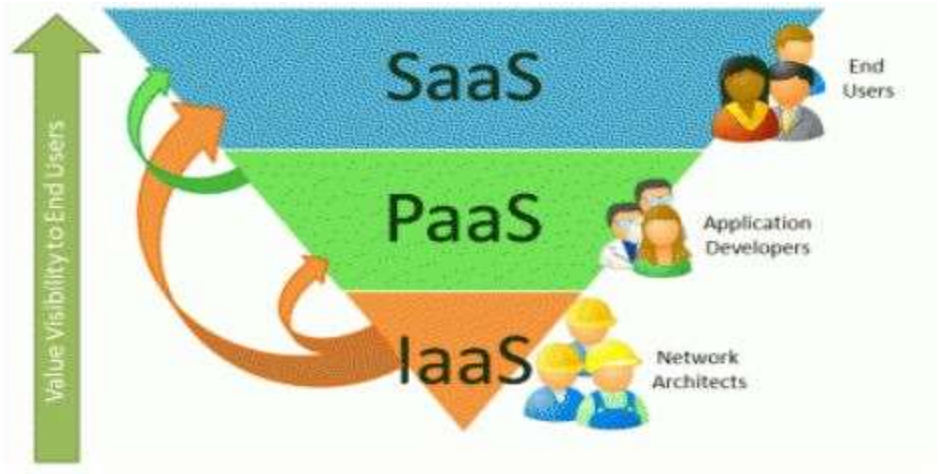

Fig.2. Types of cloud computing

Cloud computing is used for accessing, configuring and manipulating the software and hardware resources from a remote location. It tenders application, infrastructure, and online data storage. Cloud computing communicates which we can access the applications as utilities over the internet. It is used to customize, configure, and create business applications online. Fault tolerance is a considerable problem to guarantee the reliability and availability of critical services and also execution and application processing. In order to the failure impact on the application execution and system, the failure mode should be handled anticipated and proactively. Fault tolerance methods are objective to predetermine these failures and take proper action before failures actually occur. In this paper explain the overviews of the current fault tolerance methods in cloud computing based on their policies, research challenges, and tools used.

2. Related works

The low latency fault tolerance is a software related method to fault tolerance such as Delta-4 system. With low latency fault tolerance, middleware such as Web Services or applications programmed using TCP socket APIs can be replicated with strong replica consistency and without modification. Low latency fault tolerance is created for a distributed application deployed inside a cloud computing circle, rather than above the wide area internet. Other research works are given fault tolerance for web services occur. Like the hypervisor system and TFT, low latency fault tolerance middleware is transparent to both the operating system and applications. However, these systems vary from low latency fault tolerance in the path in which they attain transparency. The Hypervisor system needs a hardware instruction counter. The TFT system needs application object coding editing. Low latency fault tolerance uses the more flexible library interposition method. Using operating system facilities to low latency fault tolerance middleware can be dynamically inserted inside a running application process. Depends on virtual synchrony low latency fault tolerance middleware gives strong replica constituency, even if the backup fails or primary using the piggybacking mechanisms and Virtual Determinize Framework. Instead of piggybacking, The Delta-4 system uses an individual message to transmit ordering information from the primary to the backups.

Thus the primary must wait till the backups acknowledged the ordering notification before it sends a message, which may decrease the performance of the system. Atomic multicast protocols that distribute the messages in the same total order and constantly, like Totem, Amoeba, and Isis have been used to keep strong replica uniformity in fault-tolerant distributed systems. However, those procedures introduce delays in delivering or sending a message. The latency fault tolerance messaging procedures do not introduce such delays, because the primary in a group creates the results on the order in which the processes are executed and in the group ordering information has reflected the backups in the group.

Cloud infrastructure must be fault tolerant. Compare to traditional information processing system, Calculations the cloud is increasing popularity for the processing and storage of the data is very large. This model of computing in the artificial new data center's with thousands of servers interconnected with the capability to host a lot of applications has been made. In majority cases, out of the users and these virtual data center's computing resources on-demand services on the Internet asset in the way of VMs (virtual machines) with modifiable have been advanced.

Yabandeh and Guerraoui have forecasted that the database and resource graph is essential for the service provider to ensure the manner of the fault tolerance mechanism. A method is established by Jhawar et al. in which the fault tolerance mechanism is evaluated that uses virtualization to transparently increase the availability and reliability of cloud infrastructure. Malhotra et al explained the function of Virtualization in cloud computing. Important reasons for different standards of networking technologies, ideas of software-defined networking and virtualization are described.

3. Basic Concepts of LLFT middleware

The low latency fault tolerance middleware replicates application process to protect the application against several types of faults like crash fault and timing fault. If the processor does not generate a result within an appropriate time constraint, it is known as timing fault. If a processor generates any more results, it is known as a crash result. The low latency middleware does not handle Byzantine faults. The low latency fault tolerance middleware supports two kinds of follower replications which are semi-passive replication and semi-active replication. In semi-active replication, the primary orders the messages it receives performs the corresponding operations and gives ordering information for the non-deterministic operation to the backups. A backup log and receives incoming messages execute the operations based on the ordering information supplied by the logs and primary outgoing messages. However, does not send outgoing messages. In semi-passive replication, primary orders first messages receive and executes the corresponding operations. Then contributes ordering information for nondeterministic operations to the backups. Additionally primary communicate state updates to the backup, also database and file updates. A backup log and receives incoming messages and update it states but does not generate outgoing messages and does not executes operations. When compared to semi-active replication, semi-passive one using fewer processing resources. But if the primary fails, it incurs greater latency for recovery and reconfiguration. Set of interacting process groups known as a service group and it offers a service to the user. Set of replicas of a process called as a process group. Generally, the different processor is used to run different replicas in a process group. Process groups have two members one is primary and another one is backups. Each primary and backups have a group membership and include replicas of the process. A membership modifies that introduces a new primary constitutes a new primary view sequence number. Each member of a process, the group must know the primary of its group. There is no necessity for the members of a sending process group to know which member of a destination process group is the primary. The low latency fault tolerance (LLFT) middleware introduces a novel elegant idea of a virtual connection, which is a natural extension of the point-to-point connection of TCP.

A connection which occurs between the two endpoints is known as virtual connection, where each endpoint has a process group and over which messages are communicated between the two group endpoints. A virtual connection (VC) is a full duplex many to many communication channel b/w the two endpoints. Over the virtual connection, a sender uses UDP multicast to a destination group. A virtual port recognizes the source group from which the messages are delivered over the virtual connection (VC). Members of various process groups listen on different virtual ports and every member of a process group listen on the same virtual port. The process groups essential to know the process groups virtual ports with which they are interacting, just as with TCP. Each two service groups has a port, service group offered the service and it is accessed by the user over the gateway into the cloud. Generally, the process group is an endpoint of more than one virtual connection (VC) and participants also as shown in Figure 1.

4. LLFT MESSAGING PROTOCOL

For the application messages, LLFT (low latency fault tolerance) messaging tools offers total ordering services and reliable delivery. In total ordering, the same sequence of the messages delivered to the application by every group members in a group. In a reliable delivery, each message receives by every member that is multicast to the group on a connection. The low latency fault tolerance (LLFT) messaging tool transforms the unreliable message delivery service of UDP multicast into a reliable. Like LLFT, TCP converts IP unicast. In LLFT conversion totally ordered message delivery service between two group endpoints and in TCP conversion totally ordered message delivery service between two individual endpoints. If a fault occurs, the LLFT Messaging Protocol delivers messages to the application while maintaining virtual synchrony [2]. The LLFT Messaging Protocol incorporates flow control mechanisms same as TCP mechanism.

5. LLFT MEMBERSHIP PROTOCOL

The LLFT (low latency fault tolerance) Membership Protocol ensures that the members of a process group have a consistent view of the membership set and of the primary in the group. When compared to a multi-round consensus protocol, LLFT membership protocol is faster one because it is made by the deterministic choice of the primary. Based on rank and precedence, the primary determines the addition of the backups to the group. Order in which the member joins the group to determined precedence of a member of a group. If subsequently restarts and member fails, it is assigned new precedence and considered a new member. Due to the precedence numbers increase monotonically leads to a member that joins later has higher precedence and no two members have the same precedence in the history of a group membership. When a primary adds a new backup to the membership, it assigns the next precedence in sequence to that backup. However, the precedence of the members is not essentially consecutive, because a member might have been removed from the group. If the current primary fails, the precedence of the members determines the order of succession to become the new primary.

The rank of the backups are 2, 3…. and rank of a primary member of a group is 1. Primary assigns a rank in the order of their precedence when primary adds a new backup to the membership. The ranks are used to identify the failure of a backup or a primary. Only membership changes when changes of the primary constitute a new view, it is known as a primary view change. Every new primary view has a primary view sequence number. Adjusts the members' ranks and the proposed new primary resets the message sequence number to one when the primary view changes. It is significant for the backups to modify the primary view at accurately the same virtual synchrony point as the primary. To this end, just like other ordering information, the new primary multicasts that entry to the backups and the new primary generate an ordering information entry for the primary view change. Before the virtual synchrony point, backup changes to the new primary view when it has achieved all of the operations that were ordered.

6. LLFT VIRTUAL DETERMINIZER FRAMEWORK

Applications deployed in a cloud computing environment usually involve several sources of non-determinism. Necessary to mask or sanitize such sources of non-determinism to maintain strong replica consistency, for example, to render the application virtually deterministic. For sanitizing the sources of non-determinism, LLFT (low latency fault tolerance) virtual determinizer introduces a novel generic algorithm. We have instantiated the generic algorithm for the following kinds of non-deterministic operations like time-related operations, socket communication, and multi-threading. The state of an application process is determined by data that are shared by a thread and among the thread. Each thread changed and managed the specific data locally. Each thread inside a process has a specific thread identifier. Multiple threads shared a data item and it is protected by a mutex. The mutexes and threads can be deleted and created dynamically.
The generic algorithm records the return value information and ordering information of library calls/ deterministic system at the primary, to ensure that the backups achieve the same results in the same order as the primary.

The generic algorithm records the information of thread identifier for each non-deterministic operation. The identifier of the thread that is carrying out the operation. Operation identifier is defined as an identifier that represents data items. Data items might change on completion of the operation or at the time of the operation. The number of the operation carried out by a thread for a given operation identifier is known as operation count. The operation metadata means data returned from the library/system. It is consist of the error code, the return value of the call, and out parameters.

At the primary the algorithm keeps a queue, the order info queue four-tuples (O, D, T, N), where thread T has executed a call with operation identifier O and with metadata recorded in D, and this is the Nth time in its execution sequence that thread T has executed a nondeterministic call of interest. The order info queue spans different operations and different threads. At the primary, the algorithm appends an entry (O, D, N, and T) to the order info queue on the return of the operation O. The entries are transmitted to the backups reliably and in FIFO order, using the piggybacking mechanism of the LLFT Messaging Protocol. At a backup, for each operation O, the algorithm maintains an O. Order info queue of three- tuples (T, N, D), in the order in which the primary created them. After an entry is appended to the queue, the algorithm awakens an application thread if it is blocked. At a backup, when thread T tries to execute the operation O as its Nth execution in the sequence, if (N, T, D) is not the first entry in the O. Order Info queue, the algorithm suspends the calling thread T. It resumes a thread T that was suspended in the order in which (T, N, D) occurs in the O. Order Info queue, operating system scheduler determined in an order or , rather than in the order in which the thread was suspended. It eliminates an entry (D, N, and T) from the O. Order Info queue immediately before it returns control to the calling thread T after its Nth execution in the sequence. The algorithm needs the ordering of all related operations like the release of mutexes and claims.

The CMTS (Consistent Multi-Threading Service) is an exact instantiation of the generic algorithm, with a mutex M being used as the operation identifier. For the normal mutex claim call (pthread_ mutex_lock() library call), If the call is successful , metadata can be empty. However, if the normal mutex claim call returns an error code and for the non-blocking mutex claim call (pthread_mutex_trylock() library call), at the primary, the mechanism store the return value as the metadata in the recorded ordering information. When it is time to process a mutex ordering information entry at the backup, after that Consistent Multi-Threading Service (CMTS) mechanism study the metadata. If the metadata have an error code, without performing the call they return control to the calling thread with an identical error status. Otherwise, they delegate the mutex claim operation to the original library call offered by the operating system. If the mutex is not currently held by another thread, the calling thread attains the mutex instantly. Otherwise, the calling thread is subsequently resumed and suspended by the operating system when the thread that owned mutex releases it. While maintaining strong replica consistency, The Regular Multi-Threading Service permits concurrency of threads that do not simultaneously acquire the same mutex. While maintaining strong replica consistency, it reaches the maximum degree of concurrency possible.

7. IMPLEMENTATION AND PERFORMANCE

FOR Linux operating system, LLFT (low latency fault tolerance) middleware is applied in the C++ programming language. The library inter positioning method is used to control and capture the application's connection with the runtime environment. The application state is restored and checkpointed using facilities offered by memory mapped checkpoint library. The execution is compiled into a shared library, which is placed into the application address space at startup time through the LD_PRELOAD features of the operating system. , LLFT (low latency fault tolerance) middleware does not require relinking or recompilation of the program and it is transparent to the application. The experimental testbed contains the Linux Ubuntu 9.04 operating system, on a 1 Gbps Ethernet, 14 HP blade servers, each equipped with two 2 GHz Intel Xeon processors. A two-tier application that resembles a networked file system server/server was used to benchmark the LLFT application.

The analysis involved three types of workloads such as 0/x, x/0 and 0/0 where x denotes a variable reply or request and 0 denotes a small reply or request. The 0/0 workload characterizes operations on the 0/x workload characterizes read operations, the x/0 workload characterizes write operations in networked file systems and on file data. The end-to-end latency measurement results for semi-active replication are shown in Figure 3. The graph on the left represents the dependency of the end-to-end latency on message size (replies or request), without and with replication. The 0/x and 0/x workloads represent no observable difference in end-to-end latency for the similar x value. As can be seen in the figure, the LLFT messaging Protocol incurs very moderate overhead, ranging from around 55% overhead for small messages and 15% for large messages. The overhead is primarily due to the piggybacking of ordering information at the time of normal operation.

For large messages, which need fragmentation in user space, additional context switches are incurred. LLFT provides outstanding fault scalability due to its design. Example the performance does not degrade as the replication degree is increased (so that the system can tolerate larger numbers of concurrent faults), as shown in the graph on the right of Figure 3.

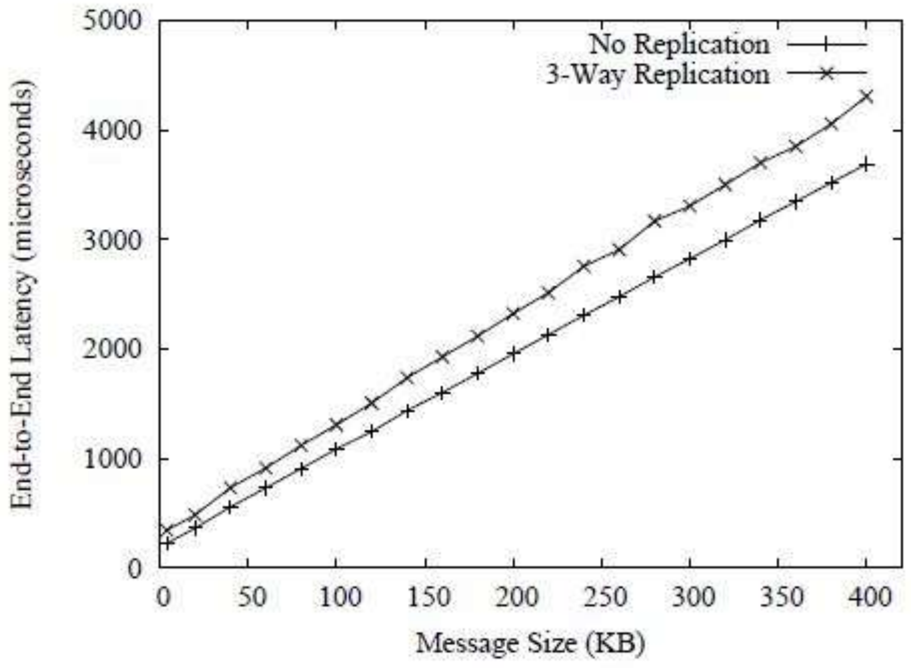

Fig.3 end-to-end latency with respect to various message sizes, with and without replication.

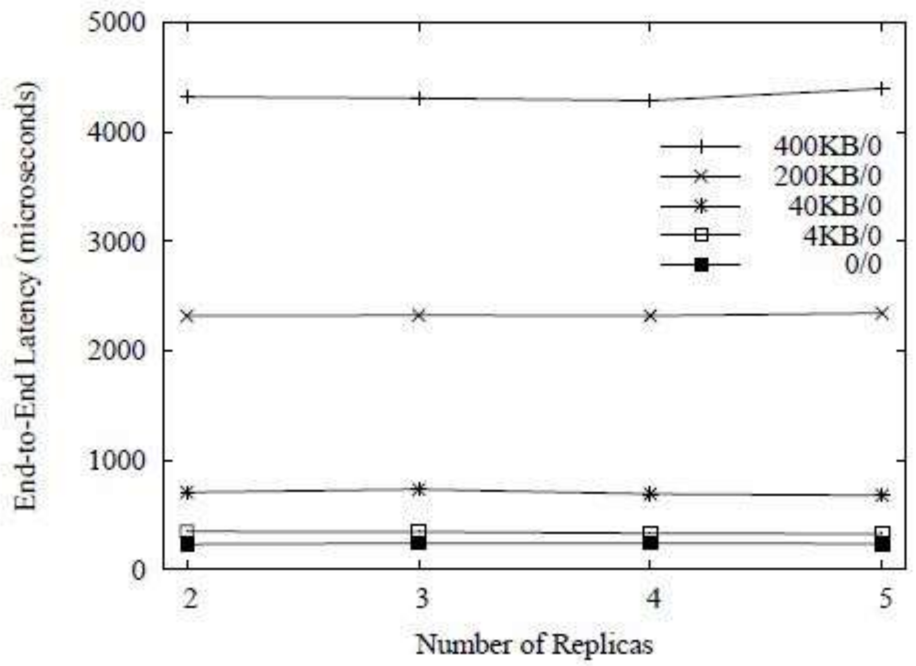

Fig.4. end-to-end latency with respect to various replication degrees

In addition to the above experiments, we analyzed the influence of system load on the end-to-end latency of Low latency fault tolerance. Without think time the load was introduced in the way of multiple concurrent clients issuing results to the server continuously. At each client, we observed end-to-end latency and run up to 10 concurrent clients because of the inadequate numbers of nodes in our test bed. The end-to-end latency remains identical to the value for the single-client condition and insensitive to the load. To analyze the capability of LLFT at the time of fault recovery, we focus on three scenarios: a) addition of a replica, b) recovery from back failure, Recovery from primary failure. Because the LLFT Membership Protocol is very effective the recovery, time from primary failure is dependent on fault detection time, it is around 10-30ms in our analysis and is importantly end-to-end latency at a client. The primary eliminates the backup from the membership and at a client no negative influence on the end-to-end latency when the backup fails. The replica introduced inside a group, it is joined as a backup. Even though a state transfer is essential to take the new replica up-to-date, the state transfer takes place off the critical path and, thus, the impact on the end-to-end latency at a client is minimal.

7. THE PROPOSED MODEL FOR FAULT TOLERANCE CLOUD COMPUTING (FTMC)

The model is designed for fault-tolerant real-time applications running on a cloud infrastructure. An adaptive real-time fault-tolerant model called cloud computing (AFTRC High). These models are shown in Figure 5. Model of systems that deal with the mechanism of fault tolerance is called fault-tolerant model for Cloud model (FTMC). It is applicable to evaluate the reliability of the computing nodes as a virtual machine (VM) in a cloud environment where fault tolerance in real-time applications are running on VMs. The validity of VM-based computing has been selected and can be removed if performance is not well for real-time applications. In this, “ N “ Virtual machine with “ N “ The type of algorithm, “ X1 “ Virtual Machine 1 Runs with “ X 2 “ and “ m” Virtual Machine 2" runs up with “ X m ” virtual machines m. Then an ACCEPTER to verify the resulting output node. The output TIME reach time the Czech will be transmitted. Reliability is assigned to each module based on the timing of RELIABILITY ASSESSOR and calculated again. Then all results are sent to DECISION MAKER for the reliability output and the output node is selected with the highest reliability.

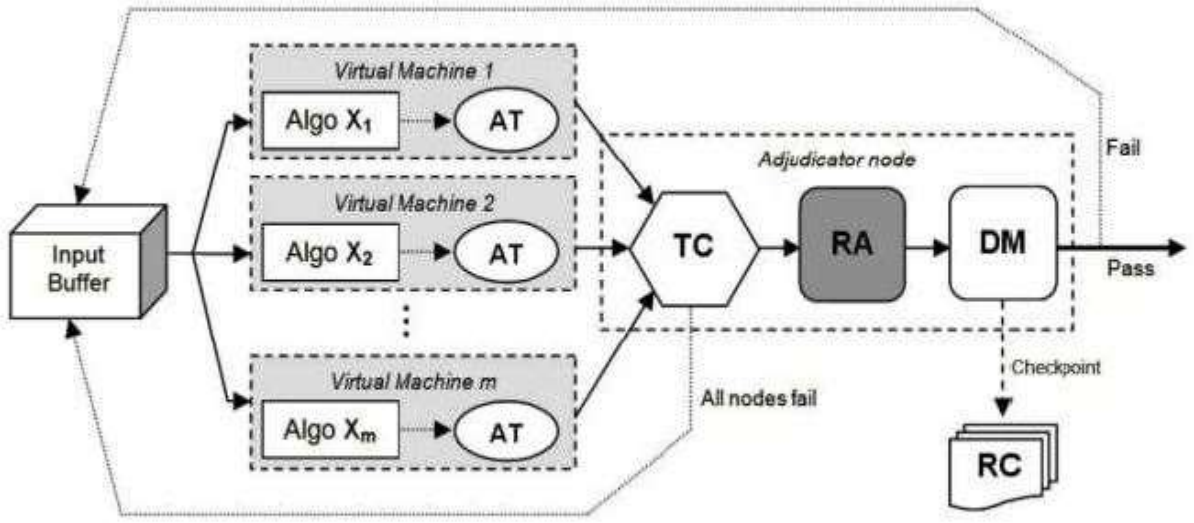

Fig.5 Model of the system

8. MECHANISMS OF TOLERANCE ERROR

Reliability evaluation algorithm on each node (VM) is a practical model with the first feature that ensures a node to ‘1’, an adjustment factor ‘n’ controls the reliability (n always greater than 0), the algorithm input factors ‘RF’ and ‘min Reliability’ and ‘max Reliability’ is the configuration file. RF Factor is reliability and this reliability of nodes increases or decreases as denoted by min Reliability, for the least reliable. If a node B reaches this level, further operations are stopped. The maximum level of reliability is denoted by max Reliability and reliability nodes cannot exceed this level. If there are two nodes and they both broke 10 passing and 10 should have a total of 20 cycles. But the biggest failure of a node occurs when the past has more chances of reliability than other nodes present. Variable amounts of (RF, min Reliability, max Reliability, SRL) depends on real-time applications. The user can decide the value of each variable.

Reliability evaluation algorithm:

```
Start
Initialized reliability: = 1, n: = 1
Input RF, ma x Reliability, min Reliability
Input processing node status
If processing node Status = Pass then
reliability: = reliability + (reliability * RF)
```

```
if n> 1 then
n: = n-1 ;
else if processing node Status = Fail then
reliability: = reliability - (reliability * RF * n)
n: = n +1 ;
if reliability> = max Reliability then
Reliability: = max Reliability
if reliability <min Reliability then
Processing node Status: = dead
Call Add new node ();
Stop.
```

Decision Maker Algorithm

```
Start
Initialized reliability: = 1, n: = 1.
input from RA processing node Reliability.
Input SRL.
Best Reliability: = find reliability of node with
highest
reliability.
if best Reliability> = SRL
status: = success
else
status: = assign to input buffer.
Calls remove processing node min Reliability
Call add new processing node
Stop.
```

The model shows the fault tolerance in cloud computing.

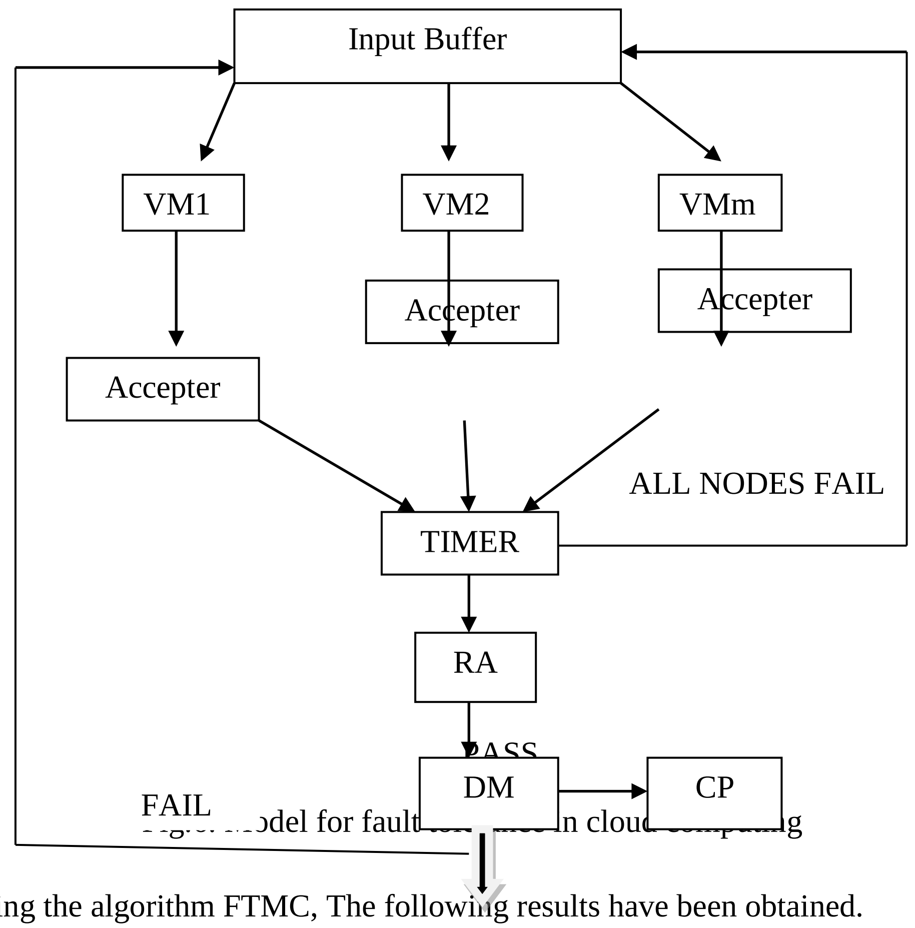

Fig. Model for fault tolerance in cloud computing

After applying the algorithm FTMC, The following results have been obtained.

TABLE I: RESULTS OF THE FTMC

Cycle	Reliability	VM1	Reliability	VM2
1	0.72	Fail	0.67	Pass
2	0.68	Fail	0.62	Pass
3	0.64	Fail	0.60	Fail
4	0.74	Pass	0.70	Pass
5	0.70	Fail	0.63	Pass

From the above table 1, it can be seen that after 5 cycles, the CPU Status of VM1 in the last cycle ‘Fails'. After 5 cycles, VM2 for the last two periods results in ‘Pass’. This is due to the presence of fewer cycles without failure of CPU. Further, the cloud environment tolerance can be done with additional fault tolerance by including additional parameters to produce a decision module.

9. CONCLUSION

The LLFT (Low Latency Fault Tolerance) middleware offers fault tolerance for disturbing applications deployed inside a data center environment or cloud computing. Using the LLFT middleware to middleware like web services or Applications programmed using TCP socket

APIs can be replicated with strong replica consistency without any change to the applications. For the application, performance measurements indicate that the LLFT middleware attains membership and low latency message delivery. Its generality, low overhead and application transparency form it proper for a distributed application deployed inside a data center environment or cloud computing. In the concept of fault tolerance and reliability of the algorithm, VM is incorporated to decide the convergence of the algorithm that shows the highest reliability. Probability of failure of this work is much less. These forward recovery plans are successful until all nodes are able to produce results and provide flexible scalability and high effective fault-tolerant capability. System behavior will become results in failure or faults and unexpected because of the dynamic nature of the cloud environment. Present work emphasized the analysis of several fault tolerance methods, issuing checkpoints to identify fault prevention and faults. In the cloud infrastructure, a Current plan is a good option as a mechanism for fault tolerance for real-time computing.

REFERENCES

1. D. Powell, Delta-4: A Generic Architecture for Dependable Distributed Computing. Springer-Verlag, 1991.
2. K. P. Birman and R. van Rennesse, Reliable Distributed Computing Using the Isis Toolkit. IEEE Computer Society Press, 1994.
3. W. R. Dieter and J. J. E. Lumpp, “User-level check pointing for Linux Threads programs,” in Proceedings of the 2001 USENIX Technical Conference, June 2001, Boston, MA, June 2001, pp.81–92.
4. K. Birman, R. van Renesse, and W. Vogels, “Adding high availability and autonomic behavior to Web Services,” in Proceedings of the 26th International Conference on Software Engineering, Edinburgh, Scotland, 2004, pp. 17–26.
5. K. K. Lau and C. Tran, “Server-side exception handling by composite Web Services,” in Proceedings of the 3rd Workshop on Emerging Web Services Technology, Dublin, Ireland, November 2008, pp. 30–44.
6. S. Malik and F. Huet, "Adaptive Fault Tolerance in Real Time Cloud Computing" IEEE World Congress on Services, pp. 280-287, 2011 .
7. Kevin Curran, Mervyn Adams, "Security Issues in Cloud Computing", chapter 14,2012
8. M. Armbrust, A. Fox, and et al., "Above the clouds: A Berkeley view of cloud computing," UC Berkeley, Tech. Rep. UCB/EECS-2009-28, February 2009.
9. K. Birman, G. Chockler, and R. van Renesse, "Toward a cloud computing research agenda," SIGACT News, vol. 40, no. 2, pp. 68-80, 2009.
10. Pa. Sadeghzade, D. Bahrepour and Pe. Sadeghzade, "Analysis of the security challenges in cloud computing", The 8 th Symposium on Advances in Science and Technology (8 th SASTech), Mashhad, Iran
11. J. Stankovic, "Misconceptions About Real-Time Computing," IEEE Computer, Vol. 21, No. 10, October 1988, pp. 10-19.

12. M. Rezazade, "Evaluate the tolerability of various forms of cloud computing", National work shop on cloud computing, 2012
13. L.L. Pullum. “Software fault tolerance and implementation Artech House, Boston, London,UK 2001.
14. S.Lakshmi Sudha. “Fault tolerance in cloud computing” IJESR International Journal of Engineering Sciences Research, Vol 04, 2013.
15. Jhawar, Ravi, Vincenzo Piuri & Marco Santambrogio, “Fault tolerance management in cloud computing”. A system level perspective” Systems Journal. IEEE 7.2(2013): 288-297.
16. R. Guerraoui and M. Yabandeh, “Independent faults in the cloud” in Proc. 4th international workshop Large scale distributed system, Middleware, no 6.2010, pp12-17.
17. Jhawar, Ravi, Vincenzo Piuri and Marco Santambrogio “Fault Tolerance Management In IAAS cloud”. IEEE 2012.
18. L.Malhotra, D. Agarwal & A.Jaiswal, Virtualization in cloud computing, JITSE, Volume 4, Issue 2,ISSN: 2165-7866.
19. R.Jain & S. Paul (2013) Network virtualization and software defined networking for cloud computing: A survey IEEE 24-31.
20. F.Lombardi, R.Di Pietro, Secure virtualization for cloud computing, Journal of network and computer applications, Elsevier 2010.