Journal of Computer Sciences and Applications, 2016, Vol. 4, No. 1, 9-13
Available online at http://pubs.sciepub.com/jcsa/4/1/2
© Science and Education Publishing
DOI:10.12691/jcsa-4-1-2

Modeling for Fault Tolerance in Cloud Computing Environment

Taskeen Zaidi, Rampratap*

Faculty of Computer Science & Engineering, Shri Ramswaroop Memorial University, Deva Road, Lucknow, India
*Corresponding author: taskeenzaidi867@gmail.com

Abstract Due to emergence of cloud computing, many traditional issues have been eliminated or reduced. Cloud computing is on demand computing in which processing is done on remote computer hence the chances of errors is more due to communication delay, latency or loss of control over computing node. These are the important issue that's why cloud computing infrastructure should be fault tolerant as well as designed to schedule tasks must be properly handled. The present paper deals with the understanding of fault tolerance techniques in cloud environments and comparison with various models on various parameters have been done. Fault tolerance technique is studied with the help of well known object-oriented language Unified Modeling Language and state diagrams are designed and validated through the concepts of Finite State Machine.

Keywords: *UML, cloud computing, fault tolerance, UML, FSM, test cases, round robin*

Cite This Article: Taskeen Zaidi, and Rampratap, "Modeling for Fault Tolerance in Cloud Computing Environment." *Journal of Computer Sciences and Applications*, vol. 4, no. 1 (2016): 9-13. doi: 10.12691/jcsa-4-1-2.

1. Introduction

Cloud computing is used for manipulating, configuring, and accessing the hardware and software resources from remote location. It tenders online data storage, infrastructure, and application. Cloud computing imparts by which we can access the applications as utilities over the internet. It is used to create, configure, and customize the business applications online. Fault tolerance is a major issue to guarantee the availability and reliability of critical services as well as application processing and execution. In order to depreciate failure impact on the system and application execution, failures should be anticipated and proactively handled. Fault tolerance techniques are aiming to predict these failures and take an appropriate action before failures actually occur. The present paper overviews the existing fault tolerance techniques in cloud computing based on their policies, tools used and research challenges.

Malik and Fabrice [1] have proposed the fault tolerance model for real time cloud computing and analyzed on the basis of reliability of processing nodes. The systems tolerate the faults and it is predicted that the reliability of virtual machines changes frequently and it may be of adaptive nature. Pullum [2] has described the concepts of software fault tolerance techniques, fault removal, fault forecasting, checkpoints, and data diverse software fault tolerant techniques. The fault tolerance techniques with more emphasis on reliability factor are well studied in [3]. An innovative method for creating and managing fault tolerance is proposed by Jhawar et al. [4]. A generic fault tolerance mechanism is for independent module which validates the users' requirements. Various factors like performance, fault tolerance, scalability of virtual management system with three structure centralized, hierarchical and peer to peer are well studied by Kong et al. [5]. Guerraoui and Yabandeh [6] have predicted that resource graph and database are necessary for service provider to ensure the behavior of fault tolerance mechanism. A approach is developed by Jhawar et al. [7] in which fault tolerance mechanism is evaluated that uses virtualization to transparently increase the reliability and availability of cloud infrastructure. Role of virtualization in cloud computing is well explained in Malhotra et al. [8]. Key reasons for virtualization and various standards of networking technologies and concepts of software defined networking are described in [9]. Lombardi and Pietro [10] have explained that security requirements are based on configuration management and vulnerability assessment. In the present work, Unified Modeling Language (UML) is also used to create a model. Let us describe some of the important review on UML which is a general purpose modeling language that is used to model various kinds of the research problem. It is widely accepted by the software professionals and developed by Object Management Group (OMG) [11,12]. Modeling with real time characterization in UML, approach and structure, viewpoint, UML viewpoint, concurrency modeling, performance modeling, real time CORBA applications are well explained in [13]. UML diagrams in mathematical analyzable format and transformed into queuing network based on intermediate textual representation has been represented in [14]. Pllana and Fahringer [15,16] have also used the UML modeling language for the performance oriented distributed and parallel architecture applications.

In the present work, we have suggested different fault tolerance techniques and proposed a fault tolerance technique using round robin scheduling scheme in which failures are minimized on system through checkpoints when a fault occurs on system. Object oriented language UML is used to design a model in terms of state transition diagram for the systems processing and updating of local and global checkpoints and model is validated by the various test cases.

2. Background

2.1. Fault Tolerance Techniques

There are two different types of fault tolerance techniques which are described below in brief:

2.1.1. Reactive Fault Tolerance

It is used to reduce effect of failures on system when actually failure occurs. It consists of the followings:

1. Checkpointing

If the task failed then it is restarted from recent checkpoint in spite of beginning. It is efficient when implemented for large application.

2. Replications

It is used to maintain execution speed for different replicas of task run on different resources until the replicated task not crashed. Various techniques like Haddop, HAP proxy and Amazon EC2 are used to implement replicas.

3. Job Migration

It is used when failure occurs then job is migrated to a new machine.

4. Retry

It is simplest technique in which user resubmits task to same cloud resource.

5. Exception Handling

In this method user defines the specific action of task failures for workflows.

6. Task Resubmission

Failed task is submitted either on same machine or on another machine when it was operated.

2.1.2. Proactive Fault Tolerance

This technique predicts fault proactively and replaces the suspended source of faults by other working components and then avoids recovery from faults or errors. It consists of the followings:

1. Self-Healing

If the instance of an application running or multiple virtual machines fails then it is controlled automatically.

2. Software Rejuvenation

System starts with new state every time and also planned for periodic reboots.

3. Preemptive Migration

The application is constantly observed and analyzed and it depends upon feedback loop control mechanism.

2.1.3. Hardware Fault Tolerance

Fault tolerant techniques are divided towards building computer that automatically recover from random faults occurring in hardware components. By partitioning of computer system into pieces, the protective redundancy is used. Major approaches to recover hardware failures are described below:

1. **Fault Masking**

It is structural redundancy technique that completely masks fault in redundant modules. Triple Modular Redundancy (TMR) is used for fault masking in which circuitry triplicated and voted. The individual voter failures are corrected by voting process.

2. Dynamic Recovery

Dynamic recovery is useful when only one copy of computation is running at a time and it involves automated self-repair. In this approach, spiral mechanism is required to detect failures in modules, switch out a faulty module, instigate actions to restore and continue completion. Using this approach, computational delays the fault recovery, fault coverage is lowered.

2.1.4. Software Fault Tolerance

Software that tolerates the software designs faults use the static and dynamic redundancy approach. The approaches like N-version programming and design diversity are used to implement fault tolerant system.

2.2. Round Robin Scheduling

Round robin scheduling is a preemptive scheduling algorithm used in time-sharing or multitasking or timesharing systems that give each process a unit of time slice or quantum for execution on central processing unit and then next process moves to ready queue, it continues until all processes completed. This scheduling requires use of timer interrupts and typically take time quantum between 10 to 100 milliseconds. Context switching is used to save states of process. The advantage of this scheduling is that it provides equal share of central processing unit to every process and it is easy to implement if number of process on run queue are known and worst-case response time for a process can be computed.

3. Proposed Algorithm

On the basis of above concepts, the following algorithms are proposed for global and local check points:

Global Checkpoint

Let us define global check point in which simulator assigns threads to the requested processes and allocates sub clouds in round robin manner and global checkpoint will be updated. It also tells the information about time of entry of job, number of processors required, serial time, thread time, etc. The algorithm is given below:

1. Virtual Machine (VM's) submits their request to cloud service provider;

2. Cloud Service Provider(CSP) assigns threads to request and allocates minimum loaded sub clouds in round robin way;

3. After allocation of subclouds global checkpoint will be updated periodically;

4. By reading checkpoint cloud service provider will check whether any subclouds has failed or no failure occurs. If no failures occur then global checkpoint will be updated but if failure occurs then job will be migrated to secondary node in round robin manner and then global checkpoint will be updated.

The above steps are represented by the use of UML in the form of state transition model:

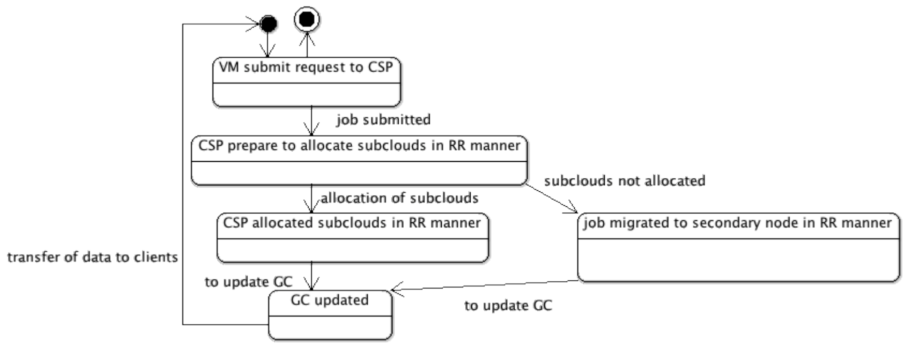
VM submit request to CSP
job submitted
CSP prepare to allocate subclouds in RR manner
subclouds not allocated
allocation of subclouds
CSP allocated subclouds in RR manner
job migrated to secondary node in RR manner
transfer of data to clients
to update GC
GC updated
to update GC

Figure 1. UML State Transition Diagram for Global Checkpoint

Local checkpoint

It tells information about server status, job status, server name and remaining time of threads. The proposed algorithm is given below:

1. Threads arrive on subclouds;
2. Subclouds check whether node is active or inactive. If inactive then a message is generated that subcloud is not responding;
3. If active then subcloud allocates minimum loaded nodes to the thread in round robin way to balance the load;
4. Global checkpoint will run periodically and a new save point will be created every time;
5. By reading checkpoint cloud service provider checks whether any node is failed or recover from failure;
6. If any node is crashed or failed then subcloud load will be shifted to another node in round robin manner in such a way that the load will be balanced and local checkpoint will be updated.

The above steps are represented by the use of UML in the form of state transition model:

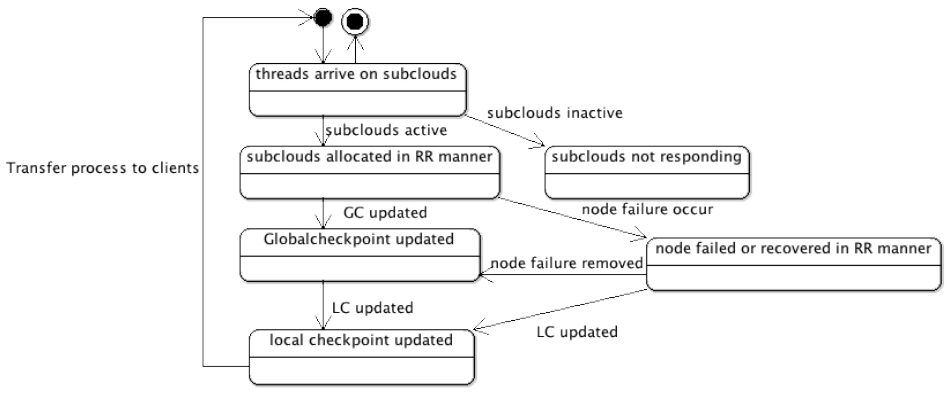

Figure 2. UML State Transition Diagram for Local Checkpoint

4. Validation of Models Through Test Cases

Authors have designed two UML models one for global check point and another for local check point which are validated by the concept of the finite state machine. Table 1 shows the list of events occurred for transition from one state to another state. Total events taken from Figure 1, Figure 2 are a,a ′’b,b’c, c',d, and e as mentioned in table.

By taking above events, equivalent state transition diagram is represented in Figure 3, Figure 4.

Table 1. List of Events for Transition Table in LC

a	subclouds active and allocated
a’	Subclouds inactive
b	Update GC
b'	Node Failure occurs
c	To update GC after recovery
d'	To update LC after recovery
d	LC updated
e	Transfer of data to user

The total states are q_0, q_1, q_2, q_3, and q_4 with number of events as represented in above table. Transition functions and equivalent grammar are generated from Figure 3, Figure 4, which are given below:

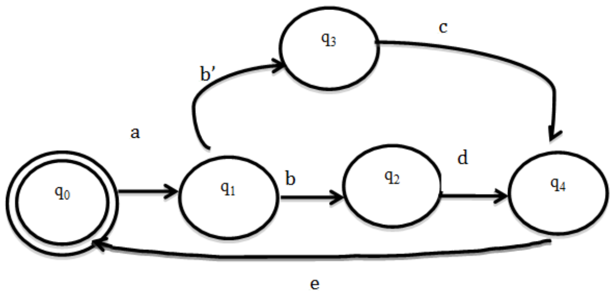

Figure 3. FSM for Global Checkpoint

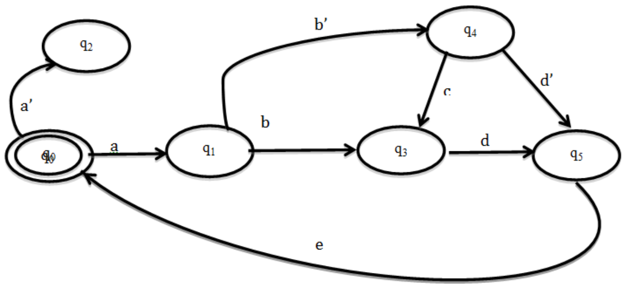

Figure 4. FSM for Local Checkpoint

Equivalent Grammar for Global Checkpoint

$\delta(q_0,a) = q_1$	⇒	$q_0 \rightarrow aq_1$
$\delta(q_1,b)=q_2$	⇒	$q_1 \rightarrow bq_2$
$\delta(q_1,b')=q_3$	⇒	$q_1 \rightarrow b'q_3$
$\delta(q_3,c)= q_4$	⇒	$q_3 \rightarrow cq_4$
$\delta(q_2,d) = q_4$	⇒	$q_2 \rightarrow dq_4$
$\delta(q_4,e) = q_0$	⇒	$q_4 \rightarrow eq_0$

Table 2. List of Events for Transition Table in GC

a	Job submitted
b	Allocation of subclouds
b’	Subclouds not allocated
c	GC updated after migration
d	Update GC
e	Transfer data to clients

Equivalent Grammar for Local Checkpoint

$\delta(q_0,a) = q_1$	⇒	$q_0 \rightarrow aq_1$
$\delta(q_1,a') = q_2$	⇒	$q_1 \rightarrow a'q_2$
$\delta(q_1,b')= q_4$	⇒	$q_1 \rightarrow b'q_4$
$\delta(q_1,b) = q_3$	⇒	$q_1 \rightarrow bq_3$
$\delta(q_4,c) = q_3$	⇒	$q_4 \rightarrow cq_3$
$\delta(q_3,d) = q_5$	⇒	$q_3 \rightarrow dq_5$
$\delta(q_4,d')= q_5$	⇒	$q_4 \rightarrow d'q_5$
$\delta(q_5,e) = q_0$	⇒	$q_5 \rightarrow eq_0$

2.3.1. Test case for Local Checkpoint Updating:

(i) Sub clouds allocation

$q_0 \rightarrow aq_1$

$q_0 \rightarrow a'q_2$

By replacing non terminals on RHS of production we can get,

$q_0 \rightarrow aa'q_2$

(ii) Global checkpoint updating

$q_0 \rightarrow aq_1$

$q_1 \rightarrow bq_3$

$q_1 \rightarrow b'q_4$

By replacing non-terminals on RHS of production we can get,

$q_0 \rightarrow abb'q_4$

(iii) Local checkpoint updating

$q_0 \rightarrow aq_1$

$q_1 \rightarrow bq_3$

$q_4 \rightarrow cq_3$

$q_3 \rightarrow dq_5$

$q_4 \rightarrow d'q_5$

By replacing non-terminals on RHS of production we can get,

$q_0 \rightarrow abcdd'q_5$

(iv) GC updating after recovery

$q_0 \rightarrow aq_1$

$q_1 \rightarrow bq_3$

$q_4 \rightarrow cq_3$

$q_3 \rightarrow dq_5$

$q_4 \rightarrow d'q_5$

By replacing non-terminals on RHS of production we can get,

$q_0 \rightarrow abcdd'q_5$

(v) GC updating after recovery

$q_0 \rightarrow aq_1$

$q_1 \rightarrow bq_3$

$q_1 \rightarrow b'q_4$

$q_4 \rightarrow cq_3$

By replacing non-terminals on RHS of production we can get,

$q_0 \rightarrow abb'cq_3$

2.3.2. Test case for Global Checkpoint Updating

(i) Global checkpoint updating

$q_0 \rightarrow aq_1$

$q_1 \rightarrow bq_2$

$q_1 \rightarrow b'q_3$

$q_3 \rightarrow cq_4$

$q_2 \rightarrow dq_4$

By replacing non-terminals on RHS of production we can get,

$q_0 \rightarrow abb'cdq_4$

(ii) GC updating after recovery

$q_0 \rightarrow aq_1$

$q_1 \rightarrow bq_2$

$q_1 \rightarrow b'q_3$

$q_3 \rightarrow cq_4$

By replacing non-terminals on RHS of production we can get,

$q_0 \rightarrow abb'cq_4$

5. Conclusion

Due to the dynamic nature of cloud environment system behavior will become unexpected and results in faults or failures. Fault detection in clouds computing environment is one of the biggest challenge now days.

The propose work emphasized towards the study of various fault tolerance techniques and then detection of faults and it prevention by issuing checkpoints in round robin manner, test cases are also designed for validation of proposed model. The faults will be handled using round robin technique with the updating of local and global checkpoint.

Acknowledgement

In all humility and with much fervor, the authors are very thankful to Computer Science and Engineering department, Shri Ramswaroop Memorial University, Lucknow to carry out the above research work.

References

[1] Malik, Sheheryar and Fabrice Huet. “Adaptive fault tolerance in real time cloud computing services, 2011 IEEE world congress on. IEEE 2011.

[2] L.L. Pullum. “Software fault tolerance and implementation Artech House, Boston, London,UK 2001.

[3] S.Lakshmi Sudha. “Fault tolerance in cloud computing” IJESR International Journal of Engineering Sciences Research, Vol 04, 2013.

[4] Jhawar, Ravi, Vincenzo Piuri & Marco Santambrogio, “Fault tolerance management in cloud computing”. A system level perspective” Systems Journal. IEEE 7.2(2013): 288-297.

[5] 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, Aug 9-12, 2009.

[6] 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.

[7] Jhawar, Ravi, Vincenzo Piuri and Marco Santambrogio “Fault Tolerance Management In IAAS cloud”. IEEE 2012.

[8] L.Malhotra, D. Agarwal & A.Jaiswal, Virtualization in cloud computing, JITSE, Volume 4, Issue 2,ISSN: 2165-7866.

[9] R.Jain & S. Paul (2013) Network virtualization and software defined networking for cloud computing: A survey IEEE 24-31.

[10] F.Lombardi, R.Di Pietro, Secure virtualization for cloud computing, Journal of network and computer applications, Elsevier 2010.

[11] OMG, 2001, Unified Modeling Language Specification. Available online via http://www.omg.org.

[12] OMG, 2002, OMG XML Metadata Interchange (XMI) Specification. Available online via http://www.omg.org.

[13] Object Management Group.UML Profile for Schedulability, Performance & Time Specification. OMG Adopted Specification pt/02-03-02, Object Management Group, March 2002.

[14] P.Kahkipuro. UML Based Performance Modeling Framework for Computer-Based Distributed Systems. In: R. Dumko (Ed.), Proceedings of 2nd International Conference on Unified Modeling Language (UML′99), pp.167-184. Springer (LNCS vol.2047), 1999.

[15] S. Pllana, T. Fahringer, 2002, On Customizing the UML for Modeling Performance Oriented Applications. In <<UML>>, Model Engineering Concepts and Tools, springer Verlag., Dresden, Germany.

[16] S. Pllana, T. Fahringer, 2002, UML Based Modeling of Performance Oriented Parallel and Distributed Applications, Winter Simulation Conference.