FTM- A Middle Layer Architecture for Fault Tolerance in Cloud Computing

L. Arockiam
Department of Computer Science, St. Joseph's College, Thiruchirappalli

Geo Francis E
Christ Academy, Bengaluru

ABSTRACT

Due to cloud computing, many of the traditional issues such as scale have been eliminated to some extent, but the stability, availability and reliability of cloud computing has received relatively limited attention. As cloud computing envisages "computing as a service" it presumes 99.99% reliability as Electricity Grid has achieved. *Reliability* of a cloud computing system depends on the probability of the failure occurring in different layers of the architecture. Virtualization technique is common in cloud computing, i.e., many virtual machines even with different operating systems may be running in a single physical machine. In order to achieve optimum fault tolerance to these virtual machines, in this paper, a middle layer is proposed and it can be placed between application layer and virtualization layer in cloud system architecture. Purpose of this middle layer is to tolerate node failure. This layer can be seen as an assemblage of various components, each with a specific functionality and it makes use of combinations of various fault tolerant strategies to achieve optimum result. Performance of this middle layer is automatic and it is user transparent too, i.e., considering economic factors, dependability factors and user's interest, it makes use of different permutations.

1. INTRODUCTION

Nowadays many of the applications in the field of research and development deals with tera-byte or even peta-byte scale of data. With the increasing scale of data sets, rolls up the problem of how to process them. Keeping infrastructures such as OS, application software, their installation, configuration, updating, huge servers and large network of nodes inside client's campus requires a significant amount of intervention of space, cost, time and personnel. Due to the difficulty to maintain such an internal infrastructure and associated cost, companies have started to outsource their hardware resources. In this context, cloud computing changed the face of computing architecture, where a web browser operate as an interface between clients and the cloud, and cloud provides the entire infrastructure. Due to high computation capabilities cloud computing is getting common in commercial purposes, research applications and day to day regular activities in which resources are shared among the cloud service consumers, partners, providers and vendors in the cloud value chain. Since cloud computing is taken up by leading industrial giants such as Microsoft, IBM, Amazon etc. cloud computing become increasingly popular.

Cloud architecture is very complex and heterogeneous system and the resources are distributed globally and there exist high level of abstraction in different layers of cloud architecture. Therefore, the end user of a cloud application usually has no idea where the data is stored, which processor is executing the computing task and how the entire process is done. They just sign up to an application request for various services. Although infinite computing resources are available on demand from the perspective of an end-user, "pay as you go" service model is adapted in cloud computing. The volume of the data to be processed, the complexity of the computing task, the time taken and all these items are considered to calculate the amount to be paid to the cloud service provider. Additionally, in cloud computing environment the customers outsource their data to the cloud which perform the computing operations and storing. Reliability, therefore, is an important issue in cloud computing.

Traditionally, one of the backbones of software reliability is avoiding the faults. Since cloud architecture is very complex and built on data centres comprising thousands of interconnected servers with capability of hosting a large number of applications and distributed globally, fault prevention techniques in developing stage is very tedious. Fault avoidance techniques or fault removal techniques such as testing to detect and remove fault, therefore, won't be enough in the case of cloud computing. In this context to achieve reliability to a greater extent, the system must be fault tolerant. According to Avi˘zienis [1], "The function of fault tolerance is to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service. This goal is accomplished by the use of error detection algorithms, fault diagnosis, recovery algorithms and spare resources."

Cloud computing architecture consists of various layers, e.g., customer applications, virtual machines, physical resources etc. A fault in cloud could be in any of the physical resources such as RAM, CPU, Hard Disk, or in the OS of the VM, or with the networking parts such as switches, routers etc. or in the application side such as compiler error, a programming error etc. Therefore it is

a Herculean task to achieve fault tolerance in all these layers in run time. The scope of this paper is limited to the failure of a computational node i.e., a Virtual Machine (VM). VM technology is widely adopted as an enabler of cloud computing and it provides a higher degree of efficiency and agility to cloud computing. A VM is a software implementation of a computing environment in which an operating system or program can be installed and run. Virtualization helps to make more efficient use of hardware resources. In Virtualization, resources of a single physical computer are shared among several different virtual computing environments effectively, i.e., one computer pretending to be several computers.

Fault Tolerance Manager (FTM) is a middle layer proposed in this paper to tolerate the failure of a VM. Failure of a VM is detected with the help of Fault Detector (FD), and preventive measures are taken by recovery overseer (RO) with the help of Replication Manager (RM) and Checkpoint Manager (CM). If permanent faults occur or some nodes are not performing according to the reliability criteria, those nodes are removed.

The rest of the paper is organized as follows: Section II presents the related work on fault tolerance in cloud computing. Section III proposes FTM, a dedicated middle layer for optimum fault tolerance with a detailed description of its various components. In section IV future works are proposed and paper is concluded.

2. RELATED WORKS

No fault tolerance technology is available with Euclyptus and CLEVER. OpenNebula implements VM Fault Tolerance. In order to overcome VM failure, a Virtual Machine hook can be set to "resubmit" the failed VM [2]. VM Crash is recovered by "onevm restart" functionality. Windows Azure [3] offers Fault Tolerance management with the replicas of each VM and this solution is limited to the applications developed in the Windows Azure platform. VM failure of Amazon EC2 is take care by Simple Que Service (SQS) and Amazon Machine Image (AMI) [4]. Service requests are queued up till they are executed properly, or deleted by the user with the help of SQS. In EC2 we can publish many Amazon Machine Images (AMI), on the failure of an AMI, we can easily replace it with the help of an API invocation.

Wenbing Zhao et. al. [5] proposes a FT middleware which implement a synchronized server replication strategy, where a failed server is repaired with a consistent state. Alain Tchana et al. [6] suggest a fault tolerance method collaborating cloud provider and cloud customer. Their integrated approach makes fault tolerance available in all levels of the cloud. However, they are not making use of VM checkpoint solutions to achieve optimum fault tolerance. Slawinska, Magdalena et al. [7] suggest transparent check pointing at the user's level provided by Distributed Multithreaded Check Pointing. By considering economic and dependability factors check points with various parameters are fixed. If these parameters are not satisfied, the thread is restarted.

3. FTM: THE PROPOSED MIDDLE LAYER ARCHITECTURE

FTM is a dedicated service layer placed between application layer and virtualization layer of Cloud Architecture. It comprises of Fault Detector, Replica Manager, Check Point Manager, Recovery Overseer and Communication Manager. A small description on each element of FTM and their functionality is given further in the section.

A. Fault Detector

There are two types of fault detection strategies, they are push model and pull model. In push model, Fault Detector (FD) sends signals to various nodes to check the health status. Whereas, in pull model method, each component in the system send signals to FD telling their health status. If no signal is obtained, FD considers that particular node is unhealthy and it reports to the master, so that, no more tasks are given to that particular node. FD of the proposed middle layer makes use of pull model method.

There are many methods to detect failure based on heartbeats and timeouts with a virtual machine such as Red Pill (a technique to detect the presence of a virtual machine developed by Joanna Rutkowska), principle canonical correlation analysis [8], decision trees [9], system performance method [10], machine learning approach [11] etc. The FD proposed in this system adapts a similar methodology mentioned in the above cases. It detects the failed VM by comparing machines performing the same task at the same time. VMs which deviate from the normal behaviour are marked as dubious. FD also identifies wrong and unexpected output from a particular node and communicates with recovery manager for appropriate action.

Apart from periodic health check up, a minimum reliability level is fixed in general. To set up a minimum reliability level, method suggested by Malik Sheheryar et al [12] is used by FD. According to this method each node is assigned a reliability weights. Nodes are added and removed from the list on the basis of their reliability. If a particular node is not reaching the minimum reliability weight, that node won't be used anymore. Once FD detects such a node in the system it communicates with the schedular with the help of CS (Communication System) of FTM.

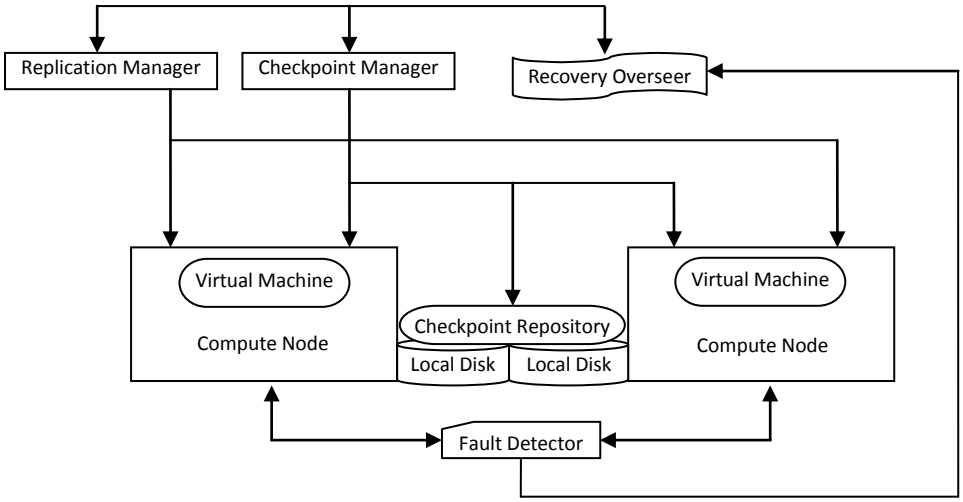

Figure 1: FTM: Fault Tolerance

B. Replication Manager

To ensure the job is finished in time, replication scheme is a good choice for providing high reliability. In the system, RM decide how many replicas to be made and of what type. In order to tolerate *n* failures, have to schedule *n + 1* replicas for each task in the workflow. There are two types of replication methods: active and passive. Active replicas will be run along with the main instance concurrently in other virtual machines. Once a node fails, or comes up with a wrong output, result from one of the active replicas could be used. And therefore, active replicas provide high degree of reliability. But the system won't go for active replicas always, because in active replication all copies of a task need to run concurrently and input data needs to be transferred to each of these copies, which incurs high communication costs, computation cost and power consumption. In the case of passive replication, a back up is only executed when its corresponding task fails. It does not always lead to a higher reliability with more replicas. Besides, the more replicas imply more resource consumption and higher economic burden. RM goes for dynamic number and types of replicas for each task. Various criterions are taken care: user's reliability requirement, probability of failure of particular kinds of tasks and Service Level Agreements (SLA). RM makes use of *MaxRe* algorithm [13], to decide on the number of replicas. *MaxRe*'s design is based on user's availability requirement and probability of the node failure; and *MaxRe* wont exceed the system capacity also.

RM is transparent to client too. Cloud computing adapt a service model, "pay as you go". According to this model, the amount to be paid depends on the volume of the data to be processed, to be stored, time of the computation etc. For some of the cloud providers such as Amazon, Google etc. the price varies depending on the time and date specifications. According to Amazon EC2 Instance Purchasing Options [14], in the case of Spot Instances, there is greater savings if your applications have flexible start and end times. In such cases, number of active replicas won't cost you more. Therefore, user has a role to decide on the number of replicas and types of replicas.

RM ensure that the replicas of a VM fail independently. A VM may fail even due to various security related issues. Unexpected intrusion to the instance by an unsecured third party may lead to unexpected wrong output. Same bugs can be triggered in all the instances. In that case, replication becomes ineffective. Therefore, RM goes for Independent replica. To achieve more independence of replica failures, the address of other replicas must be unknown to a potentially compromised replica. If it is known to the compromised replica, there is a chance for the same failure to be happen with the replica to be replaced. Therefore, a respectful level of isolation is made among different execution domains.

C. Checkpoint Manager

Checkpoint Manager (CM) functions as the traditional checkpointing-restart method. CM saves recovery information periodically during failure-free execution. When a failure occurs, the previously saved recovery information can be used to restart the computation from an intermediate state, thereby reducing the amount of lost computation. But in the case of FTM, whether to use checkpoints or not is to be decided by Recovery Overseer.

Checkpointing consumes relatively less resources in comparison with replication. The optimal number of checkpoints minimizes the expected total execution time. Some of the issues handled by CM are: The number of check points, the points during the execution of a program should use checkpoint, Reduction of checkpointing overhead, ckeckpointing storage type and location, and checkpointing frequency.

There are three checkpointing strategies. They are coordinated checkpointing, uncoordinated

checkpointing, and communication-induced checkpointing. In coordinated checkpointing, processes synchronize checkpoints to ensure their saved states are consistent with each other, so that the overall combined, saved state is also consistent. In contrast, in uncoordinated checkpointing, processes schedule checkpoints independently at different times and do not account for messages. Communication-induced checkpointing attempts to coordinate only selected critical checkpoints [15]. FTM go for coordinated check points.

Check Pointer (CP) will record the state of the job periodically at run time. Check pointing threads will be stored in the head node/Master node. Since all these checkpoints are kept in a particular log file, even if the master node fails, we need not restart the slaves. If the job in a particular node fails, with the help of Communication System, Recovery Overseer will take appropriate action. If it decides to use the checkpointing, then the recorded state of the job is moved to another computational node and resumes the execution from the last checkpoint.

In order to provide persistent storage, a dedicated repository is separately deployed as a distributed storage service. In cloud computing environment, many hard disks attached to computer nodes are not effectively used, i.e., although several hundreds of GB sizes are available with these hard disks, the VMs make use of only a fraction of it. These unfilled and unused hard disk spaces are used to store the recovery information for checkpointing. Since checkpoint repository is different from main cloud repository, the complexity is reduced. Even if, the checkpoint repository increases in size, it won't affect the central could repository, which has to take care of more complex functionalities.

One of the other features of the proposed system is the replicated checkpoints and dynamic nature of checkpoint interval. The number of VM snapshot checkpoint is going to be decided by the complexity of the application and the interest of the user. For certain level of complex applications, a minimum time interval for checkpointing is fixed. Apart from that, user can provide the interval between two consecutive checkpoints. As mentioned earlier, cost of various cloud resources may vary depending upon time and date. Since replicated checkpoints consume some resources, it may cause some economic burden to the client. But, according to the proposed system, the client has also right to decide upon the degree of checkpoints.

The proposed system goes for checkpoint optimization. Proper optimization of checkpoints reduces the resource consumption. When we take a snapshot for checkpointing from a correctly running instance, only a small part varies from the snapshot taken previously. If we are not optimizing the checkpoint, enormous redundant data transfer and use of high bandwidth would be necessary to store each snapshot. *shadowing and cloning* [16] system proposed by Bogdan is implemented with CM to achieve optimization. In this method, an object is duplicated in such a way that it looks like a snapshot, but it changes only the differences and manipulated metadata of previous image taken. These differences are not stored in a separate file; therefore, in the case of an error, it is easy to migrate to another VM.

D. Recovery Overseer

Once a node fails due to any reason, it is the task of Recovery Overseer (RO) to recover from the failure without much damage, taking less response time and waiting time. If we could reduce the additional computation time requirement after the occurrence of the fault, we can bring down the degree of damage. In the case where we resort to re-execution of the task, the system needs the same amount of execution time upon a fault. In this context RO looks for an active replica first. If an active replica is available, the replica is run on another VM and the output is obtained. If no active replica is available, it looks for a passive replica and the replica is run on another available VM. If no replica is

Algorithm 1: Function of Recovery Overseer

Initialization Phase
1. The number of active replicas *NAR*
2. The number of passive replicas *NPR*
3. The number of checkpoints *NCP*
4. The number of checkpoint replicas *NCR*

5. The address of active replicas $AAR=\{AR_1, AR_2,\ldots AR_p\}$
6. The address of passive replicas $APR = \{PR_1, PR_2,\ldots PR_q\}$
7. The address of checkpoints $ACP=\{CP_1, CP_2,\ldots. CP_r\}$
8. The address of checkpoint replicas $ACR=\{CR_1, CR_2,\ldots CR_s\}$

```
Test Phase
Begin
   If NAR ≠ 0 then
         use the output of AAR[0]
   else
         if NPR ≠ 0 then
              make APR[0] active and run on another VM
         end
   else
          if NCP ≠ 0 then
              get the latest and usable checkpoint values
              from ACP[i], where i=1,2,3…r;  and execute
              the process from the checkpoint
         end
   else
         if NCR ≠ 0 then
              do the above step for ACR[i],
              where i=1,2,3…s
         end
   else
         Migrate the job to another VM and execute from
         the beginning
   end
end
```

available for a particular process it looks at CM. If checkpoint is available, then the values are taken from the last checked point and the process is continued from the point. If a checkpoint is failed, then it looks for replica of a checkpoint stored in checkpoint repository. If checkpoint replica is also not available, then the job is migrated to another computational node and re-executes the job from the beginning.

In the case of long time running jobs, the fault occurs after a significant amount of time, the overall time will be very high. Since we have employed mechanisms such as replication and checkpointing we need to execute the task only after the last checkpoint or continue the task with a replica which considerably reduces the additional computation requirement upon fault occurrence. Also this does not cause any deterioration in the service quality as is the case with exception handlers.

E. Communication System

Communication System (CS) is an intra messaging system of FTM. CS extends to all other components of the architecture and takes care of inter component communication among various elements of FTM. CS has connection with the database, in which user preferences, with respect to replication management and checkpoint management, are stored.

There are two types of messaging models available: Public-Subscribe and Point to point. CS adapts Point to Point messaging model. Any message send by a particular component in the system can be read only by the particular message reader. E.g., a request message is passed from the RO to RM to cheque the availability of active replicas; only RM can read the message. This model avoids unauthorized intruders to the system. Message sender makes sure that the message is received only by the message receiver intended. This task is coordinated by CS.

4. FUTURE WORKS AND CONCLUSION

In this paper, a middle layer is proposed, to ensure fault tolerance with VM failure, in cloud computing environment. Although FTM is user transparent, it is automatic, flexible and agile. The system is designed bearing in mind various services offered by present cloud providers.

Functionality of each component in FTM can be refined taking various algorithms proposed by scholars into consideration. After the refinement, the system has to be implemented on cloud testbed. Research has to be done to amalgamate or to integrate the proposed middle layer with one of the existing layers of the cloud architecture.

5. REFERENCES

[1] Avi˘zienis. “The N-Version Approach to Fault-Tolerant Software.” IEEE Transactions on Software Engineering, SE-11(12) (December 1985) :1491–1501.

[2] http://opennebula.org/documentation:archives:rel2.2:ftguide

[3] http://www.davidchappell.com/writing/white_papers/introducing _windows_azure_v1-chappell.pdf

[4] http://aws.amazon.com/ec2/

[5] Webbing Zhao et. al. “Fault Tolerance Middleware for cloud computing.” Third International Conference on Cloud Computing (2010): 67-74.

[6] Tchana Alain et. al. “Fault Tolerant Approaches in Cloud Computing Infrastructures.” The Eight International Conference on Autonomic and Autonomous Systems (2012): 42-48.

[7] Slawinska, Magdalena, Jaroslaw Slawinski, and Vaidy Sunderam. “Unibus: Aspects of heterogeneity and fault tolerance in cloud computing.” 2010 IEEE International Symposium on Parallel Distributed Processing Workshops and Phd Forum IPDPSW 2 (2010): 1-10.

[8] H. Chen, G. Jiang, and K. Yoshihira. “Failure detection in large-scale internet services by principal subspace mapping.” IEEE Trans. on Knowledge and Data Engineering, (2007).

[9] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. Autonomic Computing, International Conference on Autonomic Computing (ICAC), (2004).

[10] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen. Fingerprinting the datacenter: Automated classifcation of performance crises. Proc. of the 5th European Conference on Computer Systems, (2010):111-124.

[11] I. Cohen, M. Goldszmidt, T. Kelly, and J. Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. in 6th Symposium on Operating Systems Design and Implementation (OSDI), San Francisco, CA, (2004).

[12] Malik, Sheheryar, and Fabrice Huet. “Adaptive Fault Tolerance in Real Time Cloud Computing.” 2011 IEEE World Congress on Services (2011): 280-287.

[13] Zhao, Laiping et al. “Fault-Tolerant Scheduling with Dynamic Number of Replicas in Heterogeneous Systems.” 2010 IEEE 12th International Conference on High Performance Computing and Communications HPCC (2010): 434-441.

[14] http://aws.amazon.com/ec2/purchasing-options/

[15] Roman. A Survey of Checkpoint/Restart Implementations. Technical Report LBNL-54942, Lawrence Berkeley National Laboratory, (2002).

[16] Bogdan Nicolae, Franck Cappello, "BlobCR: efficient checkpoint-restart for HPC applications on IaaS clouds using virtual disk image snapshots," Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (2011): 1-12.

[17] Zheng, Zibin et al. “FTCloud: A Component Ranking Framework for Fault-Tolerant Cloud Applications.” 2010 IEEE 21st International Symposium on Software Reliability Engineering (2010): 398-407.