ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 9(22), DOI: 10.17485/ijst/2016/v9i22/91888, June 2016* ISSN (Online) : 0974-5645

Intra State Recovery System Design for Cloud based Applications

Komal Mahajan* and Deepak Dahiya

School of Engineering and Technology, Ansal University, Gurgaon - 122003, Haryana, India;
kom12mahajan@gmail.com, deepakdahiya@ansaluniversity.edu.in

Abstract

Background/Objectives: Considering the growingdemandfor cloud services for development and deploying of critical business applications, it is extremely important that cloud provider guarantees a reliable and robust service by providing fault tolerance mechanisms that enable seamless execution of the business transaction execution even in presence of faulty components. The objective of this paper is to propose a collaborative fault tolerant mechanism between cloud provider and cloud client. **Methods/Statistical Analysis:** The collaborative fault tolerance approach considers collaboration between the cloud provider and the cloud client to develop a comprehensive fault tolerance solution that can be customised to suit to the hosted cloud applications needs.The proposed design is based on usage ofPersistent Map based strategy. **Findings:** The Persistent Map based strategy saves the state information of execution in the form of P-maps. The P-map is a persistent hash map that stores the current state of execution of a given task. In the case of failure, it can be used to restart the process from the last state at which the task failed and resume the application execution from that point as though no failure occurred.The P-map storage is a crucial element to be considered in the design of the system, that requires careful analysis and can have a huge impact on the execution of an application. **Application/Improvements:** The authors have considered an approach which requires a collaboration between cloud providers and cloud client to design a fault tolerance mechanism that takes into consideration the complex cloud infrastructure as well behaviour and functionality of the application in focus.

Keywords: Cloud Computing, Fault Tolerance, Persistent Maps, Recovery System

1. Introduction

Cloud computing is a new paradigm for providing on-demand metered[1] computing services over Internet using different models and layers of abstraction[2] to its users. The metered scalable computing as a service model of cloud has revolutionised the way enterprises use the IT infrastructure. Cloud computing is gaining popularity over classic data centers[3] for storing and processing huge volumes of data. It is an attractive option for enterprises looking to outsource their entire IT infrastructure to external data centers, thereby eliminating the upfront capital infrastructure investment. The increasing dependency on the cloud gives rise to need formore dependable and reliable cloud systems. A Cloud comprises of interconnected data centers called as clusters[2] with each data center resources having multiple virtual[3] computing and storage units called as Virtual machines. These configurable Virtual Machines are be allocated to the cloud users over Internet on demand pay per use basis. Businesses can reap huge benefits from cloud computing in terms of IT cost saving on initial infrastructure setup and maintenance, scalability, mobility, Pay-per-use software licenses and a semblance of infinite resources availability[4]to the individual user. However, in traditional data center the user has more control over the data and applications over the cloud as complex cloud architecture details are abstracted from the user. The user applications and data is at greater risk because the failures[5] (hardware failure, VM congestion, Network congestion) are out of user control. There is a need to address the research gaps in the area of fault tolerance[6] to achieve reliability[7]

* *Author for correspondence*

and availability for the real time computing on cloud infrastructure. Fault tolerance is concerned with all the techniques necessary to enable a system to recover and function in the event of the hardware or software failures. In order to minimize the impact of failure, the system should actively and proactively handle failure by implementing necessary fault tolerance techniques. Ideally, the applications should be designed and developed to handle the failure independently. However, it is practically inefficient to do so in case of applications designed to be deployed on a cloud environment because the high system complexity details of cloud infrastructure are abstracted from the user. Also, implementing the generic fault tolerance techniques[8] provided by cloud provider are inefficient as well as they do not focus on the application behaviour and functionality. Thus there is a need for a collaborative approach between cloud provider and the cloud client to develop a comprehensive fault tolerance solution that can be customised to suit to the hosted cloud applications needs. In this paper, the authors have proposed a collaborative fault tolerant mechanism that preserves the intra state information so that the system can restart from the last state at the time of failure thus minimising the chances of data loss.

2. Motivation and Problem Definition

Increasing number of companies are adopting the trend of outsourcing the computing services to the cloud providers. It has changed the way computing services[9] are developed, deployed, scaled, updated, maintained and paid for. The cloud offers the benefits of on demand scalable computing resources on pay per use model. With the growing dependency on infrastructure-less cloud computing model, there is a need for providing robust and reliable computing services to support the growing business. A cloud is a set of data centers. Each data center is virtually partitioned into number of VMs. The incoming requests to the cloud are actually deployed on the VMs. There is a possibility of a VM to fail[10] in such an environment due to multiple reasons like power failure, hardware corruption, physical damage of the hardware etc. So we require such a system that responds to unexpected failures i.e., the system is fault tolerant.

Since there are shortcomings in design and development of standalone fault tolerant applications because of abstraction[8] provided by cloud. Also, generic fault tolerance technique do not cater to customised behaviour and functionality of the applications. So we need to follow a collaborative approach to design a fault tolerance technique that takes into account both the cloud architecture and application behaviour.

This leads us to the following problem definition i.e.,

"To design a collaborative framework for fault tolerance in cloud based infrastructure to improve reliability and availability of the applications deployed on cloud infrastructure".

3. Related Study

Cloud Computing is an alternative solution for industries in which computing resources are no longer hosted on firm's in-house data centers but out-sourced to a third-party cloud provider on pay per use basis, accessed over internet. Given the scalable nature of cloud, it has become a popular solution for deploying reliable applications. However, due to dynamic nature of cloud, it is a challenge to guarantee reliability and availability for the real time computing. Considering the growing importance of cloud, it is necessary to focus research on correct and continuous operation of the applications deployed on cloud even in the presence of faulty components. A number of fault tolerance techniques have been proposed which allow applications to actively or reactively deal with faults. Based on time at which an action is taken, fault tolerance techniques are classified into Reactive[11] and Proactive[12] techniques. Reactive fault tolerance policies reduce the impact of failures when the failure effectively occurs. The Proactive fault tolerance policy is to avoid recovery from fault, errors and failure by predicting them and proactively replace the suspected component. Check pointing/Restart[11,12] is a reactive fault tolerance policy in which a task is restarted at the last checked pointed state rather than from the beginning, which makes it efficient for tasks with long execution times. In Replication based fault tolerance policies, replicated tasks are run on different resources, for successful execution and getting the desired result. It can be implemented using tools like HAProxy, Hadoop and Amazon EC2[13] etc. The Reactive fault tolerance policy based on Job Migration migrates the task to another machine/VM in case of fault. This technique can be implemented by using HAProxy. Retry

fault tolerance retries the failed task[14] on the same cloud resource. Task resubmission fault tolerance policy is based on resubmission of the failed task either to the same or to a different resource at runtime. User defined exception handling fault tolerance policy[13,15] specifies the required treatment of a task failure for workflows. A Rescue workflow fault tolerance technique enables workflow execution to continue even if it fails till the time it becomes impossible to move forward without handling the failed task. Software Rejuvenation is a proactive fault handling technique that designs the system for periodic reboots. It restarts the system with clean state. Self Healing[16,17,27] is another Proactive Fault tolerance technique in which multiple instances of an application are executing on multiple virtual machines which automatically handles failure of application instances. Proactive Fault Tolerance can also be achieved using Pre-emptive Migration[18] which relies on a feedback-loop control mechanism where application is constantly monitored and analyzed.

4. Current Work Limitations

- Single Point of Failure: The non-replication based fault tolerance policies are prone to single point of failure. Consider the case of Check pointing/Restart fault tolerance policy which is subjected to single point of failure in case of VM/machine failure. In that case, it is necessary to migrate the application to another VM/machine.
- Additional cost of operation: To improve the reliability of system, some approaches are based on incorporating redundancy in order to improve the reliability of a system. Some systems are based on space redundancy which use additional redundant hardware, software or information components and some may be based on time redundancy that may duplicate the computation or a combination of both. These approaches add to the cost of operation and can be avoided in case of non-critical business operations.
- Missing deadlines for critical job functions: In job migration, the failed tasks are resubmitted to same or different resources which will not waste any unnecessary resources but increase the makespan[20] of a workflow. Such techniques are not beneficial in case of critical tasks when the success rate of a job is based on deadlines.
- Domino effect: In case uncoordinated checkpoints are used, the rollback can lead to the Domino effect[21] i.e., Cascaded rollback which causes the system to roll back to too far in the computation.
- Overhead of maintaining checkpoints: There is an additional overhead involved in maintaining multiple checkpoints. Checkpoints consume storage resources. The users may put possible useless checkpoints. With the passage of time, some checkpoints and recovery information may become useless, and are needed to be cleaned up from the memory. Hence, garbage collection is necessary to free up the storage consumed by deletion of such useless recovery information.
- Loss of application instance specific data: In case multiple application instances are running, even if one application instance fails, it may lead to application instance data loss.
- Migration Overhead: The Job migration overhead[22] can be in the form of time and performance impact. There is migration time and cost involved to assign individual jobs to anew VM as per an assignment plan. The job migration can further degrade the performance of the impacted VM as well as it can have adverse performance impact on the destination VM.
- Loss of intra-state information: In case of check pointing, the system is rolled back to the last consistent state. This leads to loss of intra-state information.

5. Proposed System Design

The proposed system design has been illustrated in the Figure 1. The Fault Detection Manager component provides support for detection or prediction of failure in the nodes. Fault detection manager[23] continuously monitors the system for faults/failures and once detected, it notifies the Recovery Manager of the failure. RecoveryManager component handles the failure and is responsible for resuming the system back to operational state from failure state. The Recovery Manager coordinates the task of replication of the failed VM on a new destination VM. For this, it invokes the functionality of ReplicationManager which is responsible for handling the replication. The ReplicationManager uses the LoadBalancer[19] to find a new home VM for the application on the failed VM. The LoadBalancer component is responsible for finding an appropriate VM for the application in such a way that the load of the system remains balanced. The LoadBalancer can choose the VM based on Round Robin[2], equally balanced, first fit fashion etc., depending on the load balancing policy it uses. Once the LoadBalancer finds a new home VM for application, the next phase is to resume the processing of the application from the point of failure

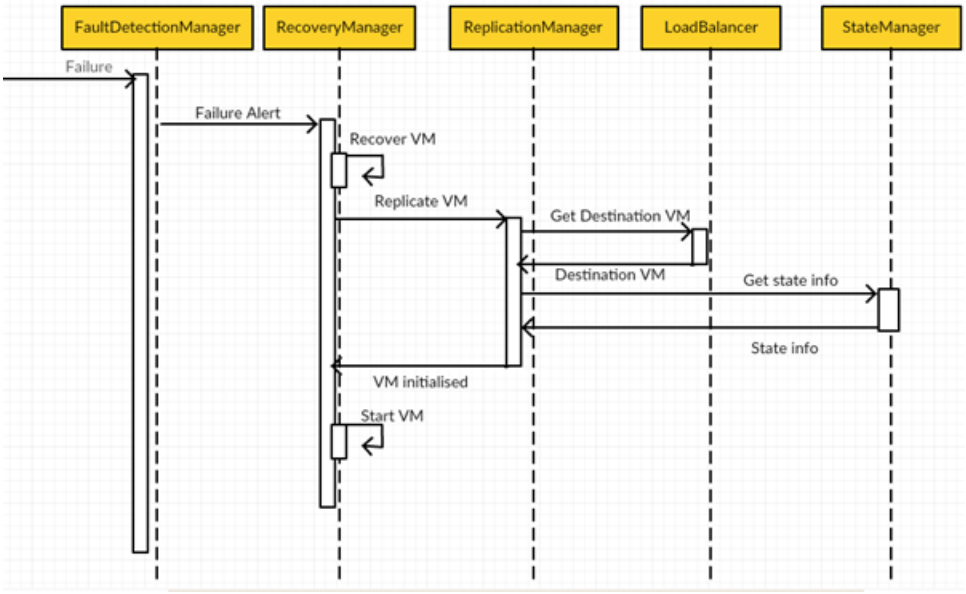

Figure 1. Sequence diagram showing the interaction among different components of a recovery system.

in a seamless manner. The ReplicationManager uses the latest state details controlled by StateManager to resume the application processing from the point of failure giving an illusion of no failure no. 1.

As a case study, we consider a client offering hosted on cloud, a web based electronic wallet (e-wallet) service. An e-wallet is a personal virtual wallet, recharged using debit or credit cards, which allows an individual to make electronic commerce transactions. The e wallet is a multitier application with data layer used to store and retrieve the customer's data, application layer used to process the e-wallet transactions, including recharging the wallet, making e-commerce payments etc. We consider e-wallet for the case study as there is huge dependency of the growing customer base on the e-wallet service and such an application would really benefit by moving to cloud. By hosting e-wallet service on the cloud, it can leverage metered services of the scalable, elastic and reliable infrastructure of cloud. Considering the growing customer base on the e-wallet service, it is necessary that application hosted on cloud should offer reliable and robust services. Given the scale of transaction on the e-wallet service, a failure in a VM can have a huge impact on the reliability and availability of the service. Thus it really important that the fault tolerance is imparted in the components, so that the service remains up and available throughout its lifecycle. In order to minimise the impact of failure on the application, it is necessary that the system should have failure handling mechanisms to actively and proactively handle failures. One way to do so is to impart fault tolerance in the application e-wallet itself. But given that the cloud environment is very complex and the underlying details of the application deployment and system implementation and abstracted from the user, the approach would be inefficient. Another way to so, would be to use the generic fault tolerance techniques provided by cloud provider. But in that case the generic fault tolerance techniques would not be very efficient to use either as they cannot be customised to focus on the application behaviour and functionality. Considering the given scenario, the best approach would be to use a collaborative approach between cloud provider and the cloud client to design a fault tolerance technique that takes into consideration the complex cloud infrastructure as well behaviour and functionality of the application in focus. The authors have proposed a collaborative fault tolerant mechanism that preserves the intra state information so that the system can restart from the last state at the time of failure thus minimising the chances of data loss.

Failure is a serious issue and cloud customer requires that cloud provider ensures the application is continuity available in a seamless manner even in case of failures as well as guarantee no or minimum loss of computation just prior to the fault. The fault tolerance techniques[23] should be designed and developed in a manner that they should

Table 1. Snapshot of parameters stored at different stages of a transaction

Stage	Description	Input Parameters	Flags	Parameters Stored in P-Map
1	Login	Username,Password	LoggedIn	Username,LoggedInFlag,UserId(-from the DB)
2	Recharge Information	Amount, Card details (CC No., Expiry Date, CCV No., Card-Holder Name), UserId	CreditStatus=0	Amount,UserId,CreditStatus,Card details(Encrypted)
3	Payment Gateway Transaction		CreditStatus=1	AmountDebited,CreditStatus
4	Wallet Update	CreditStatus,Amount	WalletUpdateStatus=1	UserId,WalletAmount,WalletUpdateStatus,CreditStatus

guarantee the same, especially in the case of large scale long running critical applications where the downtime, delays and computation loss can mean huge loss to the business.

Considering the e-wallet application. One of major functionality provided by the application providers is the ability to recharge the wallet with the desired amount using debit/credit cards or NetBanking. The wallet recharge would consist of number of sub stages each of which would involve some sub-functionality. For the given case study, we consider the following major phases in the recharge process:

- Login.
- Enter the amount one wants to recharge and bank details.
- Connect to the bank gateway and make the payment.
- Increase the wallet credit with the required amount.

Each of the phase is not atomic and consists of sub-actions. In the Login Phase, the user enter its credentials and the application layer checks the authenticity of the data entered by communication with the data layer. In the next phase, the user enters the bank card/netbanking details from which the money is to be transferred into the personal e-wallet. The next phase is connecting to the external bank gateway application where it passes the relevant bank details so as to initiate the transaction. Once the transaction is successfully completed, the next phase is to increase the wallet with required amount debited from user's bank account. Table 1 provides parameters to be stored during different phases of a transaction. Let us take the case that the VM on which the e-wallet app is hosted fails at the fourth stage when the wallet credit is supposed to be increased with the desired amount. In case when the whole transaction is considered atomic and no checkpoints are maintained, the entire transaction has to be rolled back to maintain a consistent state. Also when we maintain checkpoints, if the node and consequently the task fail before saving checkpoint, the progress will continue from the last checkpoint. In such a case, if each phase has a checkpoint, and the processing of all phases is complete and the node fails before saving the fourth checkpoint. During recovery, the recovery system would notice that the last checkpoint[24,25,28] at the third state. Thus, it would continue the processing from the fourth phase which could in the worst case mean that fourth phase would be repeated twice leading to wallet being recharged twice. This could be later noticed during reconciliation. However there could be a chance that it could be missed which could lead to losses. In order to deal with the above problem, we need a collaborative fault tolerance technique.

The authors has proposed a Persistent Map (P-map) based strategy that saves the state information of execution in the form of P-maps. The P-map is persistent hash map that stores the current state of execution of a given task. In the case of failure, it can be used to restart the process from the last state at which the task failed and resume the application execution from that point as though no failure occurred. The P- maps are used to store the intra-state information for introducing fault tolerance in a VM. The P map structure for the discussed e-wallet application is given below:

```
PMap
{
TransactionId,
Stage,
{ ParameterName1: ParameterValue1, ParameterName2: ParameterValue2, …..n} State
}
// Set of parameters and flags at certain stage of transaction
```

In case of P-map based recovery system, the P-map and

its respective storage is a crucial element to be considered in the design of the system, that requires careful analysis and can have a huge impact on the execution of an application. There are a number of P-map architectures possible based on which the P-maps can be used. The architectures give below provide a light characterization of storage schemes using P-maps:

- Central P-Map Repository per Node.
- Separate Central P-Map Repository with Syncing.
- Separate P-Map servers for node groups/cluster.
- The P-map architectures are discussed in the following sections:

5.1 Central P-Map Repository per Node

In case of the central P-map repository per node, each node has a separate central repository on the same node. All the VMs on the node store their P-maps on the central repository. The data for each transactions on the VM on the node is stored in the node specific central P-Map repository. In case of VM fault, we transfer the jobs to the new VM and P-map data for VM from central P-map repository to the new VM. In case of central P-map Repository per node, when a VM fails, the Fault DetectionManager component notifies the RecoveryManager about the VM failure. The RecoveryManager coordinates the task of replication of the failed VM on a new destination VM. For this, it invokes the functionality of ReplicationManager which is responsible for replication of the failed VM to a new Destination VM. The ReplicationManager uses the LoadBalancer to find a new home VM for the application on the failed VM. Once the load balancer finds the new home VM for the application, the next phase is to resume the processing of the application from the point of failure in a seamless manner. The ReplicationManager uses the latest state details provided by the StateManager to resume the application processing from the point of failure giving an illusion of no failure. The StateManager get the state details from the Central P-Map Repository per Node for the given VM. The replication manager replicates the job processing on the new VM home and also transfers the State details on the new Central P-Map Repository per Node. The RecoveryManager uses the state information to continue the job process from the point of failure as though no failure occurred. The Central P-Map Repository per Node is based on single node model which is suitable for time critical applications since P-map contents are be transferred to the dedicated storage on the same node, thereby eliminating network overheads in terms of delays and cost. Hence, there are no network delays involved in storage or retrieval of P-maps which leads to faster communication. However one major drawback of the proposed design is that in case the entire node fails, the fault cannot be resolved as the P-map resides on the same node.

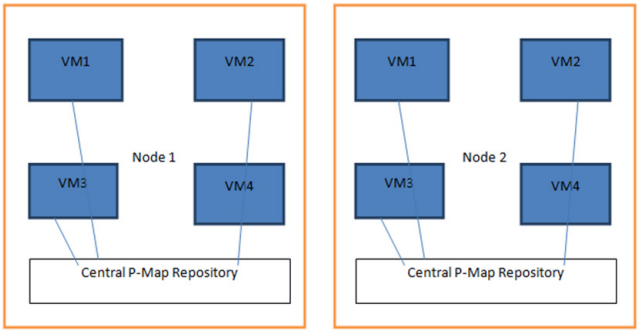

Figure 2. Representation of central p-map repository per node.

5.2 Separate Central P-Map Repository with Syncing

In case of the Separate central P-map repository per node architecture, each VM has a dedicated central P-map server which resides on a separate node. In order to save the P-map on to dedicated servers, network communication is involved. In case a VM fault occurs, we transfer the job to the new node. If the new VM is selected from the same node, then we use the P-map data on the central P-map repository to restart the job on the VM from the failed state. If the new VM is selected on a different node, then first the P-Map repositories are synced using the communication channel between the node P-Map repositories. After the P-map data is updated on destination P-map server, the P-map data is used by the new VM hosting the application to resume from the state of failure. In case of failure in a Separate central P-map repository per node architecture, the Fault DetectionManager component notifies the RecoveryManager about the VM failure. The RecoveryManager coordinates the task of replication of the failed VM on a new destination VM. For this, it invokes the functionality of ReplicationManager which is responsible for replication of the failed VM to a new Destination VM. The replication uses the LoadBalancer to find a new home VM for the application on the failed VM. Once the LoadBalancer finds a new home VM for application, the next phase is to resume the processing of the application from the point of failure in a seamless manner. The ReplicationManager uses the latest state details provided by StateManager to resume the application processing from the point of failure giving an

illusion of no failure. The ReplicationManager notifies the StateManager of the destination VM. If the destination VM is on the same node, there is no need to sync the P-map server contents. The current P-map contents can be used by the new VM hosting the failed application for Recovery. If the destination VM is on a separate node, the StateManager initiates a sync of the contents of the P-map repository of the failed VM with the new node's central separate repository. The synced P-map data can be used by the new VM hosting the application to resume application processing from the point of failure. The separate central P-map repository would not only recover VM failure but also node failure as the central P-map repository resides on a separate server so it not impacted by corresponding node failure. However, the additional fault tolerance imparted in the system with respect to the entire node failure adds to the network overhead involved in communication between the P-map repository and the VM processing node. Also, in case of failure of a single VM, if the destination VM is chosen from different node, there is an additional overhead of P-map sync. This can be taken care if Load balancing policies which are responsible for choosing the destination VM for the failed application are based on closest VM policy such that the algorithm first attempt to find an appropriate VM from the same node. If in case, they fail to find an appropriate VM from the same node, they proceed to search in the next closest node based on service proximity.

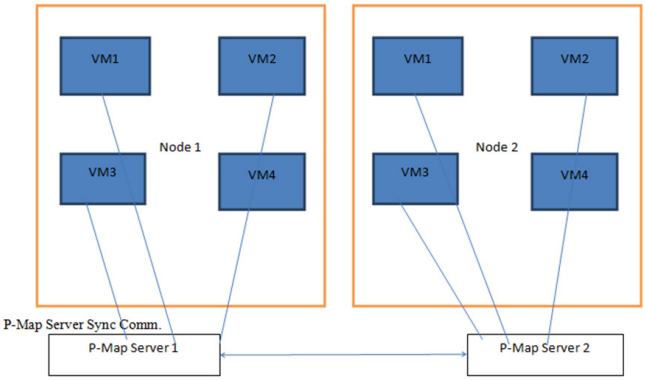

Figure 3. Representation of separate central p-map repository with syncing.

5.3 Separate P-Map Servers for Node Groups/Cluster

In case of the Separate P-Map servers for node groups, a number of nodes which are in the service proximity are classified as node groups. Alternatively, we may consider the existing cluster in the cloud to be a node group. So as per the given architecture, we do not require separate dedicated P-map server per node as it is expensive to maintain the number of servers. So another approach is to maintain separate P-map repository server per node group. Since there is trade off between storage and network communication overhead. So, this could increase the communication overhead between the VM and the corresponding P-maps repositories on the shared server. This architecture would also reduce the communication overhead involved in P-map repository sync as the probability that a sync is required would be drastically reduced as the number of P-Map servers reduces. In case of failure, in Separate P-Map servers for node groups/cluster architecture, the FaultDetectionManager component notifies the RecoveryManager about the VM failure. The RecoveryManager coordinates the task of replication of the failed VM on a new destination VM. For this, it invokes the functionality of ReplicationManager which is responsible for replication of the failed VM to a new Destination VM. The ReplicationManager uses the LoadBalancer to find a new home VM for the application on the failed VM. Once the LoadBalancer finds the new home VM for application, the next phase is to resume the processing of the application from the point of failure in a seamless manner. The ReplicationManager uses the latest state details provided bythe StateManager to resume the application processing from the point of failure giving an illusion of no failure. The ReplicationManager notifies the StateManager of the destination VM. If the destination VM is in the same node group, there is no need to sync the P-map server contents. The P-map contents can be used by the new VM hosting the failed application for Recovery. If the destination VM is on a separate node group, the StateManager initiates a sync of the contents of the P-map repository of the failed VM with the new node group's repository. The synced P-map data can be used by the new VM hosting the application to resume application processing from the point of failure. The Separate P-Map servers for node groups/cluster not only recover VM failure but also node and cluster failure as the central P-map repository resides on a separate server so it not impacted by corresponding node failure. In case of failure of a single VM, if the destination VM is chosen from different node group, there is an additional overhead of P-map sync. This can be taken care by Load balancing policies, responsible for choosing the destination VM for the failed application. If the Load balancing is based on closest VM policy such that the algorithm first attempts to find an appropriate VM from the same node group. If in case, it fails to find an appropriate VM group, which

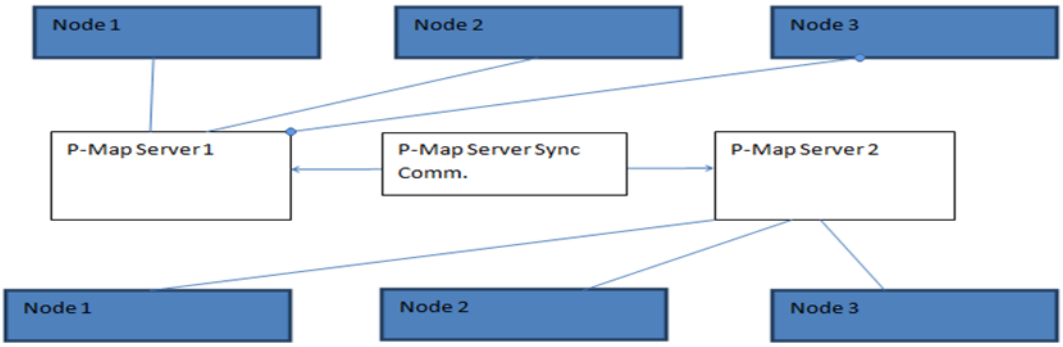

Figure 4. Representation of separate p-map servers for node groups/cluster.

has less probability, it proceeds its search to the next node group based on service proximity.

6. Conclusion and Future Work

Cloud Computing has favoured a technological revolution that has accelerated the progress characterized by innovations in the field of utility computing. Scalable computing resources are being offered as a service to the enterprises and end users on pay per use basis making them dependent on the offered services. Considering the dependency of critical business application on cloud services, it is necessary to focus research to find means and way to increase the reliability and robustness of the cloud system.

One major aspect that needs immediate attention from industry as well academic research community is efficient recovery system that introduce fault tolerance in the cloud infrastructure. This is because failures are a critical issue for business process continuity and it is important that the application continues to process the transactions with no or minimum loss of computation in an event of failure. The major contribution of the proposed work is that the authors have designed a fault tolerant mechanism that preserves the intra state information in the form of Persistent maps so that the system can restart from the last state at the time of failure thus minimising the chances of data loss. Ideally, the application hosted on cloud should actively and proactively handle failures but this may not be the scenario with cloud hosted application as complex system architecture of cloud is abstracted from the application designers. Also, the generic fault tolerance mechanisms provided by the cloud may not be efficient as they would cater to the business functionality and behaviour of each of the hosted applications So the authors have considered an approach which requires a collaboration between cloud providers and cloud client to design a fault tolerance mechanism that takes into consideration the complex cloud infrastructure as well behaviour and functionality of the application in focus.

The proposed design is based on usage of Persistent Maps (P-map), a persistent hash map, to save the state of execution of transaction on cloud applications. In the case of failure, P-maps can be used to restart the process from the last state at which the task failed and resume the application execution from that point as though no failure occurred. The P-maps are used to store the intra-state information for introducing fault tolerance in a VM. The authors have also discussed the respective storage of P-map in case of P-map based recovery system which is a crucial element to be considered in the design of the system. The authors have discussed Central P-map repository per node, in which each node has a separate central repository on the same node. All the VMs on the node store their P-maps on the central repository. The data for each the transactions on the VM on the node are stored in the node specific central P-Map repository. The authors have also discussed the proposed architecture, Separate central P-map repository per node, in which each VM has a dedicated central P-map server which resides on separate node and network communication is involved in order to save the P-map on to dedicated servers. Lastly, the authors have discussed another proposed architecture, Separate P-Map servers for node groups, in which a number of nodes which are in the service proximity are classified as node groups and there is a separate P-map repository sever per node group, which eliminates the need to have a separate dedicated P-map server per node as it is expensive to maintain the number of servers.

7. References

1. Mell P, Grance T. The NIST definition of cloud computing.
2. Mahajan K, Makroo A, Dahiya D. Round robin with server affinity: A VM load balancing algorithm for cloud based infrastructure. Journal of information processing systems. 2013; 9(3):379–94.
3. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M. A view of cloud computing. Communications of the ACM. 2010 Apr 1; 53(4):50–8.
4. Shen Z, Subbiah S, Gu X, Wilkes J. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. Proceedings of the 2nd ACM Symposium on Cloud Computing; 2011 Oct 26. p. 5.
5. Zhang Q, Cheng L, Boutaba R. Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications. 2010 May 1; 1(1):7–18.
6. Zhao W, Melliar-Smith PM, Moser LE. Fault tolerance middleware for cloud computing. 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD); 2010 Jul 5. p. 67–74.
7. Birman KP, Van Rennesse R. Reliable distributed computing using the ISIS toolkit. Los Alamitos, CA. CA: IEEE Computer Society; 1994.
8. Jhawar R, Piuri V, Santambrogio M. Fault tolerance management in cloud computing: A system-level perspective. IEEE Systems Journal. 2013 Jun; 7(2):288–97.
9. Dikaiakos MD, Katsaros D, Mehra P, Pallis G, Vakali A. Cloud computing: Distributed internet computing for IT and scientific research. IEEE Internet Computing. 2009 Sep; 13(5):10–3.
10. Inoue T, Umeno H, Tanaka S, Yamamoto T, Ohtsuki T. System for recovery from a virtual machine monitor failure with a continuous guest dispatched to a nonguest mode. United States patent US 5,437,033; 1995 Jul 25.
11. Oldfield RA. Investigating lightweight storage and overlay networks for fault tolerance. Proceedings of the High Availability and Performance Computing Workshop; Santa Fe, NM. 2006 Oct.
12. Vallee G, Engelmann C, Tikotekar A, Naughton T, Charoenpornwattana K, Leangsuksun C, Scott SL. A framework for proactive fault tolerance. IEEE Third International Conference on Availability, Reliability and Security, 2008, ARES 08; 2008 Mar 4. p. 659–64.
13. Bala A, Chana I. Fault tolerance-challenges, techniques and implementation in cloud computing. IJCSI. 2012 Jan; 9(1):288–93.
14. Hwang S, Kesselman C. A flexible framework for fault tolerance in the grid. Journal of Grid Computing. 2003 Sep 1; 1(3):251–72.
15. Juhnke E, Dornemann T, Freisleben B. Fault-tolerant BPEL workflow execution via cloud-aware recovery policies. IEEE 35th Euromicro Conference on Software Engineering and Advanced Applications, 2009, SEAA'09; 2009 Aug 27. p. 31–8.
16. Dai Y, Xiang Y, Zhang G. Self-healing and hybrid diagnosis in cloud computing. Cloud Computing. Springer Berlin Heidelberg; 2009 Dec 1. p. 45–56.
17. Krutz RL, Vines RD. Cloud security: A comprehensive guide to secure cloud computing. Wiley Publishing; 2010 Aug 9.
18. Engelmann C, Vallee GR, Naughton T, Scott SL. Proactive fault tolerance using preemptive migration. 2009 IEEE 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing; 2009 Feb 18. p. 252–7.
19. Mahajan K, Dahiya D. A cloud based deployment framework for load balancing policies. 2014 IEEE 7th International Conference on Contemporary Computing (IC3); 2014 Aug 7. p. 565–70.
20. Wu Z, Ni Z, Gu L, Liu X. A revised discrete particle swarm optimization for cloud workflow scheduling. 2010 International Conference on Computational Intelligence and Security (CIS); 2010 Dec 11. p. 184–8.
21. Guermouche A, Ropars T, Brunet E, Snir M, Cappello F. Uncoordinated checkpointing without domino effect for send-deterministic mpi applications. 2011 IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2011 May 16. p. 989–1000.
22. Beloglazov A, Buyya R. Energy efficient resource management in virtualized cloud data centers. Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing; 2010 May 17. p. 826–31. IEEE Computer Society.
23. Jhawar R, Piuri V, Santambrogio M. A comprehensive conceptual system-level approach to fault tolerance in cloud computing. 2012 IEEE International Systems Conference (SysCon); 2012 Mar 19. p. 1–5.
24. Nicolae B, Cappello F. 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 Nov 12. p. 34.
25. Pal AS, Pattnaik BPK. Classification of virtualization environment for cloud computing. Indian Journal of Science and Technology. 2013 Jan; 6(1). DOI: 10.17485/ijst/2013/v6i1/30572.
26. Rajathi A, Saravanan N. A survey on secure storage in cloud computing. Indian Journal of Science and Technology. 2013 Apr; 6(4). DOI: 10.17485/ijst/2013/v6i4/31871.
27. Nagaraju S, Parthiban L. SecAuthn: Provably secure multi-factor authentication for the cloud computing systems. Indian Journal of Science and Technology. 2016 Mar; 9(9). DOI: 10.17485/ijst/2016/v9i9/81070.
28. Kumari PS, Kamal ARNB. Optimal integrity policy for encrypted data in secure storage using cloud computing. Indian Journal of Science and Technology. 2016 Mar; 9(11). DOI: 10.17485/ijst/2016/v9i11/88453.