International Journal of Computer Science and Engineering Communications
Vol.3, Issue 3, 2015, Page.1159-1164
ISSN: 2347–8586
www.scientistlink.org

# Fault Tolerance in Cloud Using Reactive and Proactive Techniques

**[1]Kalanirnika G R, [2]V.M.Sivagami**
[1] PG Scholar, [2] Associate Professor
Department of Information Technology
Sri Venkateswara College of Engineering, Chennai, India,
*[1]kalanirnika.ravi@gmail.com,[2]vmsiva@svce.ac.in*

**Abstract:** Fault tolerance plays a vital role in ensuring high serviceability and reliability in cloud. A lot of research is currently under way to analyze how cloud can provide fault tolerance for an application. The work proposes a reactive fault tolerant technique that uses check pointing to tolerate the fault. The work proposes a VM- µ Checkpoint framework to protect both VMs and applications in the VMs against transient errors. The VM-µCheckpoint mechanism is implemented using CoW-PC (Copy on Write – Presave in cache) algorithm. The CoW-PC algorithm presaves all the tasks running on the VM's in a cache memory. When there is any transient failure happening in VMs, it is noted and it is recovered using last presaved checkpoint from the cache memory. Once the tasks are executed successfully, the presaved checkpoints are deleted automatically from the cache memory. Thus the algorithm uses in place and in memory recovery of checkpoints that reduces the checkpoint overhead and improves the performance.

***Keywords:*** *Checkpoint, Fault Tolerance, Transient Errors, Copy on Write, Reactive, Proactive techniques.*

## 1. Introduction

Fault tolerance is an approach where a system continues to success even if there is a fault. Fault tolerance is a major concern to guarantee availability and reliability of critical services as well as application execution. To minimize the failure impact on the system and application execution, failures should be handled effectively. Although there are number of fault tolerant models or techniques are available but still fault tolerance in cloud computing is a challenging task. Because of the very large infrastructure of cloud and the increasing demand of services an effective fault tolerant technique for cloud computing is required. In the proposed model fault tolerance is integrated with the cloud virtualization. The basic mechanism to achieve fault tolerance is replication or redundancy. The paper proposes a VM-µCheckpointing technique that tries to minimize the checkpoint overhead and speed up recovery by means of copy-on-write, dirty page prediction, in-place and in memory recovery, as well as saving incremental checkpoints in volatile memory. Virtual machine checkpoints provide a clean encapsulation of the full state of an executing system . There are various faults which can occur in cloud computing .Based on fault tolerance policies various fault tolerance techniques can be used that can either be task level or workflow level.

### A. Reactive Fault Tolerance

Reactive fault tolerance policies reduce the effect of failures on application execution when the failure occurs effectively. There are various techniques which are based on these policies like Checkpointing, and restart, Replay and Retry and so on.

### B. Proactive Fault Tolerance

The principle of proactive fault tolerance policies is to avoid the recovery from faults, errors and failures by predicting and proactively replacing them with the suspected components of other working components. Some of the techniques that are based on proactive fault tolerance policies are Proactive Fault Tolerance using Preemptive migration, Software Rejuvenation, Load Balancing, etc.

### C. Adaptive fault tolerance

The fault-tolerance needs of an application change depending on its current position in its state space and the range of control inputs that can be applied. Therefore, in adaptive fault tolerance all procedures are done automatically according to the situation. Adaptive Fault Tolerance (AFT) can assure adequate reliability of critical modules, under temporal and resources constraints, which is done by allocating just as much redundancy to less critical modules as can be afforded, thus elegantly reducing their resource requirement.

**2. Literature Survey**

The system proposed a technique in [5] tries to improve the virtual machine availability called virtual machine time travel. This is done using check pointing and continuous data protection. This technique tries in reverting a virtual machine's state from transient and persistent, to the past points in time. This capability is used to improve virtual machines availability, and to recover from operator errors. The paper presents an approach to virtual machine time travel which combines Continuous Data Protection (CDP) storage support with live-migration-based virtual machine check pointing. Specifically, the paper presents a novel approach for CDP that enables efficient reverts of the storage state to past points in time and makes it possible to undo the revert which is achieved by a simple branched-temporal data structure. The system thus supports fast, reversible revert which speeds the return to VM availability after a VM failure. For some applications such as system administration and forensics storage cloning is needed to complement time travel, and clones also have applications in areas such as VM image management.

The system proposed a technique in [3] uses memory exclusion to optimize performance of virtual machine check pointing. Virtual Machine (VM) level checkpoints bring several advantages. They are compatibility, transparency, flexibility, and simplicity. But, the size of VM-level checkpoint will be very large and even in the order of gigabytes. This major disadvantage takes very long time for VM check pointing and restarting. Therefore to reduce the size of VM checkpoint, the paper proposes a ballooning technique. The basic function of ballooning technique is to adjust the memory usage of domain which is done by passing memory pages back and forth between the hypervisor and the virtual machine page allocator. A pseudo-device-driver is loaded into the guest OS to implement the balloon module. The driver is responsible in making the memory adjustment using the existing OS functions. This avoids modifying the native memory management algorithms. To reclaim the memory, the hypervisor instructs the driver to inflate the balloon which allocates pinned physical pages within the VM. The same way, the hypervisor can deflate the balloon by instructing it to return previously allocated pages. The guest OS is reserved to physical memory and creates memory pressure in the VM when the balloon inflates. Saving unnecessary free pages in the VM is eliminated using this method.

The system in [1] proposed a checkpoint agent for user-level process in the check pointing VM, which takes a periodical checkpoint interval of the protected VM. The checkpoints are then stored in the check pointing VM. At each checkpoint interval the Copy-on-write- Basic (CoW-B) mechanism is invoked to identify and store the needed state information. Thus, the checkpoint agent stores a small fraction of the protected VM rather than the entire system image. And the checkpoints of multiple protected VMs on the same physical machine can be stored in the check pointing VM. At the beginning of a checkpoint interval, all memory pages of the protected VM are set as read only. From that point on, any write to a read-only page triggers a page fault. Thus a page fault is triggered at the beginning of each checkpoint interval.

The system in [6] proposed a model for adaptive fault tolerance in real time cloud computing. In the proposed model, the system tolerates the faults and makes the decision on the basis of reliability of the processing nodes that is the virtual machines. The reliability of the virtual machines is adaptive, which changes after each and every computing cycle. If the virtual machines manage to produce the correct result within the time limit, its reliability will increase. And if it does not produce the result within time, its reliability will decrease. The proposed technique is a good option to be used as a fault tolerance mechanism for real time computing on cloud infrastructure. It has an advantage of forward recovery mechanism, a dynamic behavior of reliability configuration and it is highly fault tolerant.

The system proposed a technique in [4] called ReVivel I/O to handle I/O efficiently in highly available rollback recovery servers. The increasing demand for reliable computers has led to proposals for hardware-assisted rollback of memory state. Such approaches promises major reductions in Mean Time to Repair (MTTR). The benefits are especially compelling the database servers, where the existing recovery software typically leads to downtimes to tens of minutes. therefore, adoption of such proposals is hindered by the lack of efficient mechanisms for I/O recovery. Thus the paper presents and evaluates ReViveI/O, a technique for I/O undo and redo that is compatible with mechanisms for hardware assisted rollback of memory states. The paper has been implemented using a Linux-based prototype that shows that low-overhead; low-MTTR recovery of I/O is feasible. For 20 to 120 ms between the checkpoints, a throughput oriented workloads such as TPC-C has a negligible overhead. Moreover, for 50ms or less between the checkpoints, the response time of a latency-bounded workload such as Web Stone remains tolerable. In all cases, the recovery time of the ReViveI/O technique is practically negligible. The results are cost-effective with highly available server. The system in [2] proposed a method called Remus to produce high availability via asynchronous virtual machine replication. Allowing applications to survive hardware failure is an expensive process, which generally involves re-engineering a software to include complicated recovery logic as well as deploying special – purpose hardware.

This represents a severe barrier to improve the dependability of large or legacy applications. The paper elaborates the construction of a general and transparent high availability service that allows the existing and unmodified software to be protected from the failure of the physical machine on which it runs. Remus provides a very high degree of fault tolerance, to a point that the running systems can transparently continue the execution on an alternate physical host when a failure has occrs with only seconds of downtime, while completely preserving host state with active network connections. This approach describes protected software in a virtual machine, asynchronously propagates the changed state to a backup host at frequencies as high as forty times a second, and uses speculative execution mechanism to concurrently run the active VMs slightly ahead of the replicated system state.

**3. Proposed Scheme**

In the reactive approach the fault tolerance policy, it allows the occurrence of fault, error and failure and then tackles the fault to provide service continuously. Checkpointing is a reactive fault tolerance in which, when a task fails, it is allowed to be restarted from the recently check pointed state state rather than from the beginning. For long running applications it is an efficient task level fault tolerance technique.

This paper describes an implementation of a fast and lightweight checkpoint mechanism for virtual machines. The proposed VM-Checkpoint technique uses a checkpointing framework to protect both VMs and applications in the VMs against runtime errors. The VM-Checkpoint technique takes in-memory incremental checkpoints and employs in-place restoration. The proposed Checkpointing technique uses CoW-PC (Copy-On-Write Presave in cache) algorithm to optimize the checkpoint overhead. A VM checkpoint is a consistent image of the complete state of a running VM at a given point in time. Earlier VM – Checkpointing uses a pre copying mechanism to drastically reduce the downtime incurred while a VM is migrated between hosts. Pre copying is a technique that consists of a number of memory copying rounds while a VM is running, before the actual checkpoint/migration point. Pre-copying does introduce substantial resource overheads as copying is performed over a number of rounds; moreover, these overheads and the VM's downtime are sensitive to the write intensity of a VM's execution behavior and can grow to unacceptable levels with particular VM workloads. Rather than performing any copying of state while a VM is suspended, a consistent view of a VM is taken by write protecting all the VMs using CoW (Copy-on-Write) based checkpointing technique. Checkpoint and rollback techniques have been extensively studied in the literature. Checkpoints can be taken in different levels. Here the paper focuses on checkpoint techniques in the virtual machine level. Most existing VM checkpoint/replication techniques are based on live migration of VMs, which continually transmit dirty pages of a VM from a source node to a destination node. These techniques exploit the live migration mechanism for the purposes of VM checkpointing, VM rejuvenation, load-balancing, and fast VM forking. The proposed VM-Checkpoint technique differs from other VM checkpointing techniques in that we aim to i) provide high-frequency checkpointing and rapid recovery of VMs, and ii) It minimizes the checkpoint overhead by placing the checkpoints in-memory and performing in-place recovery.

**4. The CoW-PC Algorithm**

A large number of page faults are incurred in COWB [1] because all memory pages are set as read-only at the beginning of each checkpoint interval. We design an optimized version of this basic algorithm, called Copy-on-Write Presave in cache (COW-PC), to reduce the number of page faults and corresponding performance overhead (checkpoint-caused page faults are reduced by 75% when the checkpoint interval is 50ms in our experiments were studied). Specifically, COW-PC predicts the pages to be updated in the upcoming checkpoint interval and presaves the predicted pages in the cache memory when this interval begins. These pre-saved pages are marked as writable and do not raise page faults. The checkpoint intervals selected in our scheme ranges from tens of milliseconds to several seconds.

Algorithm: CoW-PC

1. *Initiate the Cloud Server.*
2. *Create the required no. of datacenters*
3. *Initiate the broker service.*
4. *Broker service assigns VMs to datacenter.*
5. *Then the tasks are assigned to VMs and the process is presaved in cache memory.*
6. *If tasks are found to have any transient errors, recover the tasks using CoW-PC mechanism by using the last presaved checkpoint from cache memory.*
7. *If tasks are executed successfully, delete the process from cache memory.*

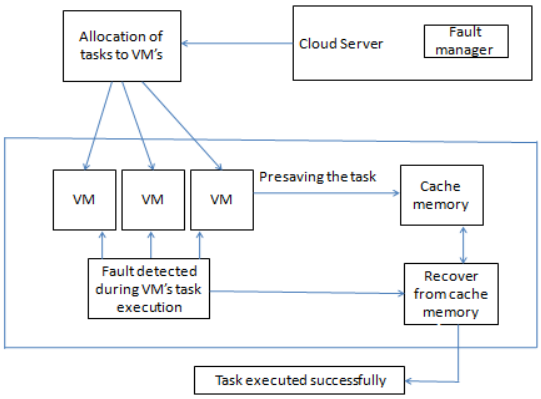


**Fig.1:** Proposed System Architecture of CoW-PC

Fig.1 shows the working mechanism of CoW-PC algorithm. The CoW-PC algorithm presaves all the tasks running on the VM's in a cache memory. The cache memory is placed inside the node itself. In our scheme we assume a node to have 10 VMs. So all the tasks running inside the VM is presaved in the cache memory. When there is any transient failure happening in VMs, it is noted and it is recovered easily using last presaved checkpoint from the cache memory. Once the tasks are executed successfully, the presaved checkpoints are deleted automatically from the cache memory. Therefore the main problem of checkpoint overhead is thus reduced by using the pre saving method. At the beginning of a checkpoint interval, register states in the protected VM are saved in the checkpoint, and all memory pages are set as read-only. From that point on, any write to a read-only page triggers a page fault, the original data of the page are copied into the checkpoint kept in the checkpoint agent memory, and the stored memory page is set as writable. Consequently, the checkpoint consists of pre-write data of pages updated within a given checkpoint interval. The typical checkpoint intervals (selected in our approach) range from tens of milliseconds to several seconds. Due to the space and time locality of cache memory accesses, pages that were updated recently tend to be updated again in the near future. Therefore, the pages dirtied in the previous checkpoint interval are used to predict the pages to be updated in the upcoming interval. The CoW-PC algorithm recovers a guest system and applications in the system from any transient hardware error or transient software error, including both application and system errors. Transient hardware errors include those occurring in the processor (functional units, registers, buses, and control logics) and in cache memory due to events such as radiation or current disturbances.

## 5. Results and Discussion

The proposed work is simulated using Cloud simulator software. The results are based on the implementation of CoW-PC Algorithm. The results shown are based on the performance graphs that are plotted between the virtual machines and their execution time. Two graphs are plotted for comparison, where one graph shows the better performance with less execution time when CoW-PC algorithm was implemented using the cloud simulator. The other graph shows the performance when the CoW-PC algorithm was not implemented and the time for executing the tasks by the virtual machines was higher in this case. The system on which the proposed algorithm is implemented is a heterogeneous system which is capable of executing multiple tasks in parallel with less downtime. The algorithm is implemented in a simulation platform where fault is induced to analyse the performance of the system where transient error occurs in VM ID 3 and it is recovered using the proposed algorithm shown in Fig 3.The output is stimulated using cloud stimulator by creating virtual machines, data center, cloudlets, broker etc. The various parameters which are given as input for setting up this environment are ram size, MIPS, Band width, PES Number. All the results are obtained by using a Intel core i5 PC with 2.5 GHz CPU and 4 GB memory. The operating system used was windows 8, language used is java (jdk 1.5). Net Beans IDE 7.4 was used to implement java and mysql database. The output of the proposed system was implemented using the cloud sim tool. Firstly, the cloud environment for cloud users using cloud data centres, virtual machines, cloudlets were created and the jobs were scheduled using the schedule manager from client to server. Then the deployment of cloud environment with data center network scheme, virtual machine assignments. Then CoW-PC algorithm was implemented in this cloud environment which helps to allocate the resources efficiently. And the performance analysis of CoW-PC on the parameter values resource utilization of virtual machines, execution time and cpu utilization time.

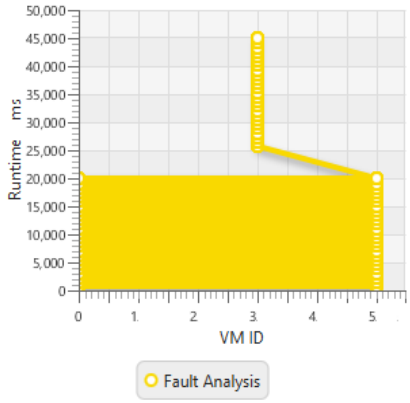


**Fig 2:** Fault analysis

Fig 2 shows the execution time taken by each VM when the fault has occurred in the system. Each VM has its ID named as VM ID 0,1,2,3, and so on. The graph is plotted between VM ID in the x-axis and runtime in the y-axis. In this graph the tasks executed in VM ID 3 takes a longer runtime due to some transient error in the system. Due to this fault the performance of the system will be reduced.

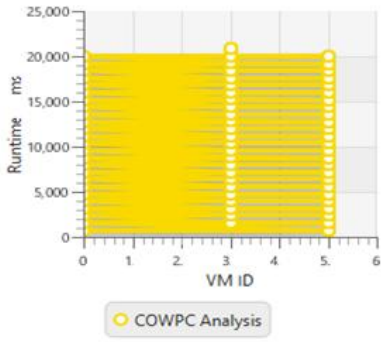


**Fig 3:** CoW-PC analysis

Fig 3 shows the execution time taken by each VM when the Fault had occurred in the system with CoW-PC implementation. It shows that VM ID 3 takes oly a few milliseconds to execute the task when CoW-PC algorithm was implemented.

| Fault Analysis | | CoW-PC Analysis | |
|---|---|---|---|
| **VM ID** | **Runtime (ms)** | **VM ID** | **Runtime (ms)** |
| 0 | 20,000 | 0 | 20,000 |
| 3 | 45,000 | 3 | 21,000 |
| 5 | 20,000 | 5 | 20,000 |

**Table 1:** Performance comparison between Fault and CoW-PC analysis

The Table 1 shows the comparison of fault analysis and CoW-PC analysis. The VM ID 3 takes around 45,000 ms to execute its task when a fault has occurred in the system. But using the CoW-PC algorithm, when fault has occurred in the system, the VM ID 3 takes around 21,000 ms to execute its task. Thus when CoW-PC was implemented in the system the failure in the system is efficiently handled by reducing the runtime by half. Thus the performance of the system will also be increased.

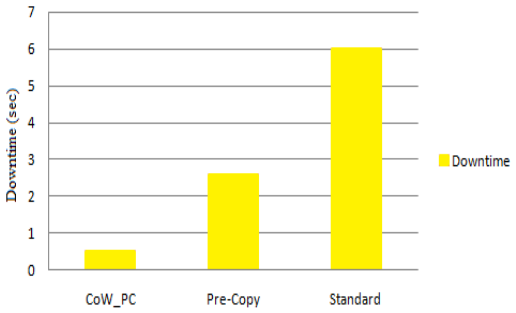


**Fig 4:** Comparison of Proposed algorithm with standard Benchmark

Fig 4 shows the downtime comparison of CoW-PC algorithm with pre-Copy and standard checkpointing algorithm. The Pre-copy mechanism uses the migration techniques to tolerate the fault whereas CoW-PC algorithm does not use any such migration techniques, thus avoiding the overhead of migrating and reduces the downtime also. This clearly shows that the reduced downtime increases the performance of the system by 75% compared to the next closest system.

## 5. Conclusion

This paper proposes VM-µCheckpoint that uses the CoW-PC algorithm which is a light weight VM checkpointing technique. It minimizes the checkpoint overhead by placing the checkpoints in-memory and performing in-place recovery. It provides high frequency checkpointing and rapid recovery of VMs. The VM-µCheckpoint was implemented using the cloud simulator tool. The experimental results show that the proposed technique achieves better performance in the occurrence of faults and the runtime performance incurs low overhead due to checkpoints. It was also studied that it showed better performance compared to the existing technique based on VM live migration.

## 6. Future Work

The proposed algorithm can be implemented in real time platforms such as Xen server since the simulation results were found to be effective by reducing the checkpoint overhead and improving the performance of the system with less downtime. The checkpointing based technique is called the reactive approach of Fault Tolerance. This approach reacts after the fault has occurred. Though the algorithm tolerates the fault, there is always performance degradation in the system due to the occurrences of the fault. So, the scope of the future work may be implementing a proactive fault tolerance approach. The dynamic Load balancing technique which is a proactive approach that can be implemented to proactively detect the occurrence of the fault before it could occur. By predicting the occurrence of the fault in advance and tolerating it by load balancing techniques will yield better performances.

## References

[1] L. Wang, et al., Checkpointing Virtual Machines Against Transient Errors: Design, Modeling, and Assessment.
[2] B. Cully et al. Remus: High Availability via Asynchronous Virtual Machine Replication, Networked Systems Design and Implementation, 2011.
[3] Haikun Liu, Hai Jin, Xiaofei Liao, Optimize Performance of Virtual Machine Checkpointing via Memory Exclusion, 2012.
[4] Jun Nakano, Pablo Montesinos, Kourosh Gharachorloo†, and Josep Torrellas, ReViveI/O: Efficient Handling of I/O in Highly-Available Rollback-Recovery Servers,2012
[5] Paula Ta-Shma et al., Virtual Machine Time Travel Using Continuous Data Protection and Checkpointing,2012
[6] Sheheryar Malik et al., Adaptive Fault Tolerance in Real Time Cloud Computing,2013
[7] C. Clark et al. Live Migration of Virtual Machines. In Networked Systems Design and Implementation, 2005. B. Nagarajan et al. Proactive Fault Tolerance for HPC with Xen Virtualization, Proceedings of International Conference on Supercomputing, 2011.
[8] Gu et al. Fault Inject Based Study of Fault Resilience of Hypervisor, University of Illinois Urbana Champaign report 2010.
[9] Robert Bradford et al. Live Wide-area Migration of Virtual Machines Including Local Persistent State, The 3rd International Conference on Virtual Execution Environments, 2010
[10] Tobias Distler et al. Efficient State Transfer for Hypervisor-based Proactive Recovery, 2nd Workshop on Recent Advances on Intrusion-Tolerant Systems, 2010.