A Review on Virtual Machine Fault Tolerance Techniques

Literature Review
SCS 3216 : Research Methods

Pratheek Senevirathne
19001622

January 17, 2024

Declaration

The literature review is my original work and has not been submitted previously for any examination/evaluation at this or any other university/institute. To the best of my knowledge, it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name : P. L. W. Senevirathne
Registration Number : 2019/CS/162
Index Number : 19001622

Signature & Date

This is to certify that this literature review is based on the work of Mr. P. L. W. Senevirathne under my supervision. The literature review has been prepared according to the format stipulated and is of acceptable standard.

Supervisor Name : Dr. (Mrs). Dinuni K. Fernando

Signature & Date

Acknowledgment

I like to thank my supervisor, Dr. Dinuni Fernando for providing me with the topic for this literature review and leading me throughout the entire project and encouraging me at every time to finish the tasks properly. I would like to thank my parents, siblings, and colleagues for all the support given to me throughout the process of writing this literature review.

Abstract

The virtual machine (VM) is a software emulation of a computer, which allows multiple such VMs to run on a single physical computer, and it is one of the main building blocks of cloud computing. Fault Tolerance (FT) is the ability of a system to continue performing its function without regard to any expected or unexpected hardware or software failures. As with any system, VMs are failure prone. It is hard to implement and thoroughly manage VM FT because of the complex nature of VMs and their management environments. VM failures may occur in any moment of its execution or during migration from one host to another, so to mitigate VM failures, there are two main ways present in the literature, namely, proactive FT and reactive FT. Proactive FT schemes try to predict future failures and try to avoid them, where as reactive schemes react after a fault occurrence to hide the failure to the user. One of main ways of VM FT is Live migration, which allows the VM to migrate to a different host while VM continues its execution. During the migration, it's probable that VMs can fail due to impending failures and hardware and software crashes. It is important that any VMs in transit are not lost. There are ongoing and consistent research endeavors aimed at putting in place various FT strategies for VMs, and to recover VMs in the case of a complete system failure.

This survey provides a thorough and organized explanation of various fault categories, their causes, and multiple strategies for achieving fault tolerance in virtual machines, as documented in existing literature. This survey also contains a comprehensive overview and analysis of different fault tolerance frameworks, outlining their fundamental methods, main characteristics, and drawbacks, and finally, possible future research directions presented.

Contents

List of Figures

List of Tables

List of Acronyms

VM	Virtual Machine
PM	Physical Machine
VMM	Virtual Machine Monitor
FT	Fault Tolerance
AWS	Amazon Web Services
CDC	Cloud Data Center
HPC	High Performance Computing
OS	Operating System
ML	Machine Learning
GCP	Google Cloud Platform
SLA	Service Level Agreement
GB	Gigabyte
QoS	Quality of Service
WWS	Writable Working Set
LRU	Least Recently Used
RDMA	Remote Direct Memory Access
MMU	Memory Management Unit
SVM	Support Vector Machine

1 Introduction

1.1 Virtualization

Virtualization is a technology that uses software to create a virtual platform on top of computer hardware, which enables the hardware components of a single computer, such as processors, memory, and storage, to be divided into multiple Virtual Machines (VMs). The main underlying technology of a VM is Virtualization, and it is the foundation of Cloud Computing (IBM 2023).

1.2 Virtual Machine

A VM can be thought of as a software emulation of a physical computer. This technology allows several Operating Systems (OSs) to run on a single computer, each with its own virtualized hardware, and each VM functions separately from the other VM in the same physical computer/server. VMs are used for a variety of purposes, including testing and development, server consolidation, running legacy applications, etc. They are often managed by a hypervisor which is also knows a Virtual Machine Monitor (VMM), a software that creates and manages VMs.

Public cloud service providers like Google Cloud Platform (GCP), Azure, and Amazon Web Services (AWS) have been utilizing VMs to offer virtual applications to a large number of users simultaneously, providing them with cost-efficient and flexible compute resources. These VMs can be easily scaled up or down to meet the changing demands of the applications they host, and customers only pay for the resources they use.

1.3 Types of VMs

There are two main categories of VMs, namely,

1. Process virtual machine:

   A process VM enables a single process to act as an application on a host machine. It does this by creating a programming environment that is not tied to a specific platform and by concealing the intricacies of the underlying hardware and operating system. One common type of process VM is the Java Virtual Machine, which enables Java applications to run on any operating system as if they were customized for that particular system.

2. System virtual machine:

   A system VM is completely virtualized to act as a Physical Machine (PM). A system platform facilitates the sharing of a host system's hardware among multiple VMs, each having its own OS. This type of virtualization is achieved using a hypervisor.

1.4 Types of hypervisors

There are two main types of hypervisors (VMMs),

1. Type 1 hypervisors:

   Type 1 hypervisors, also known as bare metal hypervisors communicate directly with the computer hardware and completely take the place of the traditional Operating System (OS). It replaces the OS, and all the hardware interactions of VMs are handled via the hypervisor. They are mainly used in virtual server environments.

2. Type 2 hypervisors:

   This hypervisor function as a software on top of an OS installed in a PM (Host OS). They are usually utilized on desktop/laptop devices to run alternative OSs, and to create an isolated environment.But they tend to have a performance penalty as they need to use the host OS to access and manage the underlying hardware .

This literature review mainly focuses on reviewing scholarly articles on fault tolerance mechanisms of system virtual machines running on top of bare-metal hypervisors. Since the main application area of the topic is on Cloud Data Centers (CDCs) , most of the articles discuss the application on Cloud Computing.

1.5 VMs' role in Cloud Data Centers

According to Bulao (2023), by the end of 2022, 57% of businesses globally have moved their workload to cloud services. Most of those services/applications run on VMs, resulting millions of virtual machines running in the CDCs. In fact, a lot of services offered by the cloud service providers use at least a single VM. In a single data center, there are hundreds of physical servers, each running up to 50-150 VMs, resulting in tens of thousands of virtual machines per CDC. At this scale, regular hardware and software faults related to VMs in a CDC are inevitable.

According to extensive studies carried out by X. Sun et al. (2019) and Birke et al. (2014), most of the VM and PM failures (around 81%) are related to hard drive failures, considering the number of failures that is happening in a CDC, the software and other hardware related failures very much affect the performance/up time of the VMs. Birke et al. (ibid.) have found out that even though VMs have a lower failure rate compared to PMs, the VMs have higher inter-failure times.

1.6 Faults, Errors and Failures

A fault refers to an abnormal condition or malfunction that occurs in one or many software or hardware components, which might cause the system not being able to complete its tasks. One or more such faults in a system leads to a system error. An error is characterized as a decline in one or more components of a system, causing a discrepancy between the typical and current state of the system. These errors can ultimately cause a system failure, which will interrupt the normal working of system services. If not managed correctly, system failures can cause the system to become inoperable (D. Sun et al. 2012; Jhawar and Piuri 2017).

VM failure can be seen as a failure in the end user application/service running in the VM. An end user application/service may experience a failure when the is a failure or an error in the PM (server) components, or in software components such as the VMM, VM instance, or the OS. Figure 1 depicts this as a fault tree where the root show the failure seen to the end user.

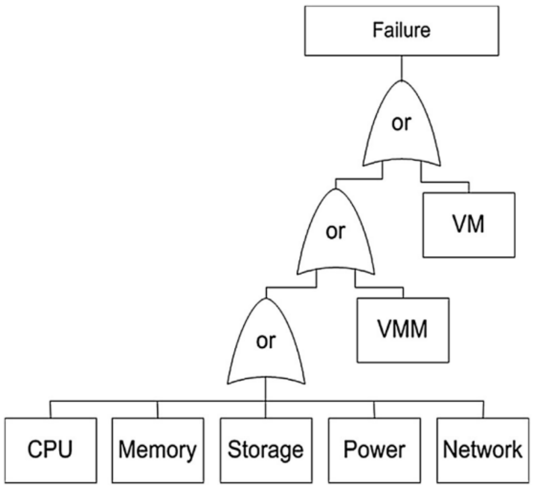

Figure 1: Fault tree characterizing VM failures (Jhawar and Piuri 2012).

When it comes to cloud services, even small system outage (less than an hour) will cost the cloud service provider millions of dollars in loss in revenue . For instance, during the Google Cloud outage which lasted around an hour in December 2020, Google incurred an estimated loss of $2.3 million (Khalili 2020). This is one of the reasons which makes Fault Tolerance (FT) as one of the well researched area in cloud computing. Even with latest improvements with regards to FT in cloud technologies, according to Lawrence (2022), high cloud outage rates have not changed significantly during the past three years (2019-2022).

1.7 Fault tolerance

FT is the ability of a system to continue operating without any issues even in failure of its sub-components. The main idea of FT is to ensure that the system remains available, and that data is not lost or corrupted in the event of failure. A system that lacks the ability to tolerate faults, even if it has been designed well and uses top-quality components, is not considered reliable. Reliability is among the most significant aspects of the cloud, as lots of time-sensitive and real-time apps are running on the platform. Thus, the fault tolerance has a sizable attention in literature (D. Sun et al. 2012; Jhawar and Piuri 2017).

1.8 The scope

The scope of this literature review is to:

- Describe various fault types that may occur in virtual machines and causes for them.
- Describe the basic virtual machine fault tolerance approaches.
- Describe and analyse different virtual machine fault tolerance frameworks/methods proposed in the literature.
- Provide future research directions under virtual machine FT.

2 Fault classification

As stated by Jhawar and Piuri (2017), faults related to distributed computing systems like CDCs can be classified mainly into two types,

1. Crash faults:

   Faults which happen because of an issue in the system components , such as the power supply, memory units, secondary storage units, processors, networking devices, etc. Usually, this type of fault requires manual interaction to repair the affected components. We can use software systems to tolerate crash faults that may happen in a system.

2. Byzantine faults:

   Byzantine faults refer to errors or failures in computer systems that can lead to inconsistent or conflicting results , making it difficult to determine the correct outcome . The term ”Byzantine” is used because these faults resemble the behavior of the Byzantine generals problem in distributed systems, where conflicting information and multiple decisions can cause problems. These faults can arise due to hardware/software failures, or malicious activity. They are particularly challenging to properly check and find the conditions caused the fault.

As shown by the figure 2, some fault types can be in both crash and byzantine fault categories, because they exhibit properties of both crash and byzantine faults. Hasan and Goraya (2018) have classified and summarized fault types they identified in the literature , table 1 gives a brief idea of them.

3 Fault tolerance approaches

Virtual machine FT schemes proposed in the literature as solutions to tolerate the aforementioned faults can be mainly classified in to two types, namely, proactive approaches , and reactive approaches. Under these two approaches several fault tolerance methodologies have been proposed in the literate . Figure 3 provides an outline of all the VM FT methodologies classified as a hierarchy .

Fault type	Category	Brief description
Configuration fault	Crash	Refers to a situation where the arrangement of the system's components is disrupted
Constraint fault	Both	Refers to a scenario where a malfunction occurs and is not addressed by the party responsible for handling it
Hardware fault	Crash	Refers to an issue that arises at the infrastructure level due to the malfunction of a piece of hardware.
Network fault	Crash	Refers to a problem caused by the malfunction of any component in the network, such as switches or routers.
Parametric fault	Byzantine	Refers to a problem caused by unexplainable changes in the parameters.
Participant fault	Both	Refers to a problem resulting from a disagreement between different users, such as consumers, providers, administrators, etc.
Resource contention fault	Both	Refers to an outcome of a conflict when a shared hardware or software resource is being accessed by more than one process.
Retrospective fault	Both	Refers to a problem caused by the absence of information about the system's previous behavior.
Software fault	Byzantine	Refers to a problem associated with the software part of the system, typically caused by software updates, malware, or attacks.
Stochastic fault	Both	Refers to a problem caused by a lack of statistical data to determine the system's state.
System fault	Byzantine	Refers to a problem caused by a lack of complete understanding of the processes which handles provisioning of services.
Time constraint fault	Byzantine	Refers to an instance where a S/W is not able to finish its task within the given timeline.

Table 1: Fault categorization

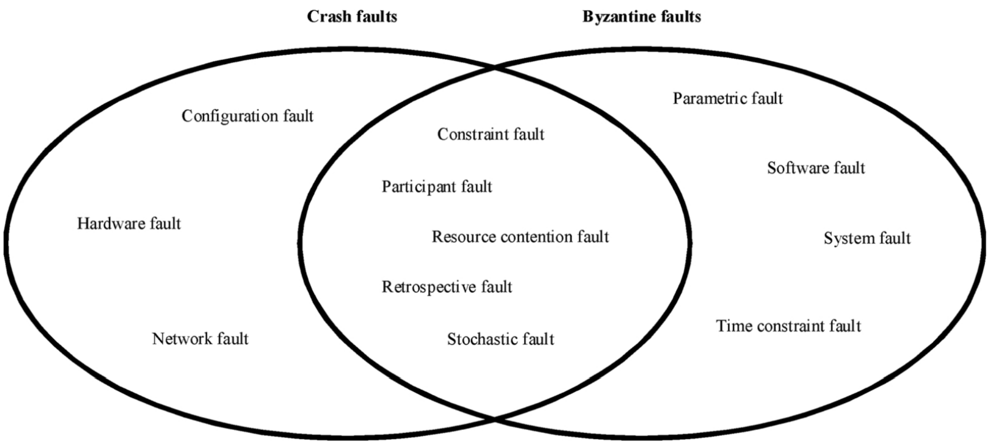

Figure 2: Fault categorization (Hasan and Goraya 2018)

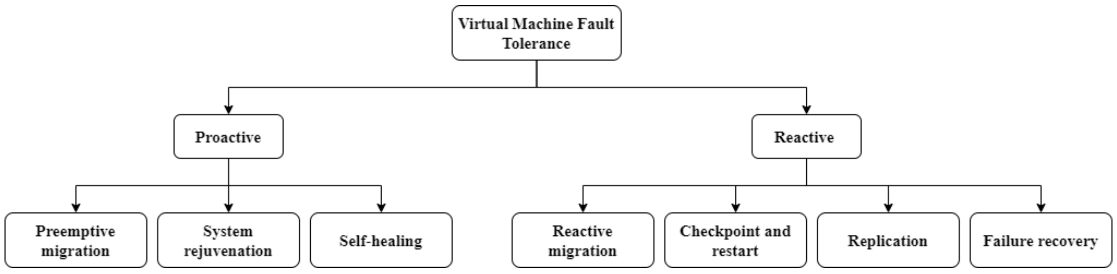

Figure 3: VM FT methodologies

3.1 Proactive approaches

The term *proactive* in the area of FT refers to the capability of the system to be ready for possible future faults and handle them before a system failure occurs. Proactive fault tolerance methodologies continuously monitor the system and predicts fault occurrence using a machine learning or a similar approach.

3.1.1 Preemptive migration

In preemptive migration frameworks , the system migrates (moves) the VM from possible suspicious nodes that might fail in the future to a healthy node or PM. In this method, the system is equipped with pre-fault indicators which will predict any event of a fault in close future. If the system predicts such a failure, the VMs in the probable failing node is migrated to another node immediately. Engelmann et al. 2009; Polze, Tröger, and Salfner 2011

Fault tolerance is one of the main applications of VM migration. There are other applications of VM migration such as for system maintenance, power management, load balancing , etc. (Ahmad et al. 2015). In this review, I focus on the application of VM migration on fault tolerance. VM migration can be subdivided into two, namely, Live VM migration and Non-Live VM migration . These are the primary methods engaged to migrate a VM from one PM to another PM. Figure 4 illustrates the taxonomy of VM migration schemes.

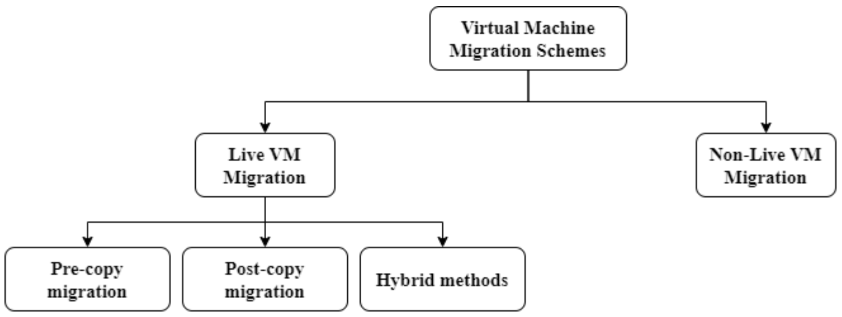

Figure 4: Classification of VM migration Schemes

3.1.1.1 Non-live VM migration

This schemes pauses the running VM at the source node immediately after receiving the signal to migrate the VM, and then it copies the entire VM memory state from source to target node, and starts and resumes the VM at the destination node after all the memory pages and the system state is copied. This scheme guarantees a consistent, and predictable migration time, and the memory pages are copied exactly once. Even though migration time and the network workload is greatly reduced when using this method, the end user applications will face a long period of service interruption, and using this method may lead to Service Level Agreement (SLA) violation.

3.1.1.2 Live VM migration

Under this pattern, the frameworks try to migrate the VMs while the VM is turned on, and the applications are still running in them with minimal interruptions to the running applications (Clark et al. 2005). The main objectives of this approach is to optimize the end user application performance , minimize the down time. There are three sub-categories, Pre-copy migration, Post-copy migration, and hybrid approach. Figure 5 illustrates the timeline of pre-copy vs post-copy methods.

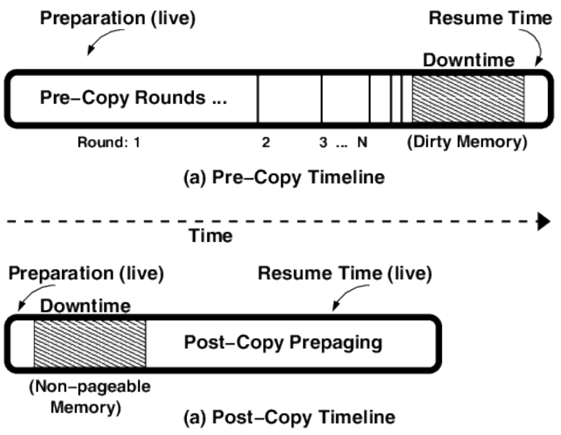

Figure 5: Pre-copy vs Post-copy timeline (M. Hines and Gopalan 2009)

1. **Pre-copy:**

   A pre-copy migration scheme will migrate the VM from probable failing node to a target node by iteratively copying all of its memory content before activating the VM in the target node. The pre-copy approach consists of a few rounds, in each round the dirtied memory pages are copied, till the rate of dirtying of pages goes below a certain value. Then the VM is stopped (paused) in the currently running node and rest of the dirtied memory content is copied to target, and the it is started back again in the target . This migration scheme is resource intensive and the total migration time is also comparatively high (Shribman and Hudzia 2013).

   Pre-copy migration schemes can further be divided to two subcategories on the objective of the migration scheme, namely, improving network performance or improving application performance .

   (a) *Network Performance:* Since pre-copy process copies dirtied memory pages over and over again, a VM migration process may migrate several hundred of Gigabytes (GBs) worth of memory pages. This affects the network performance of the CDC. This type of VM migration schemes try to use network bandwidth optimization strategies to reduce the total network bandwidth used.

   (b) *Application Performance:* This type of pre-copy frameworks try to improve performance of end user applications running in the target VM by using techniques to properly manage the daemon, which is the background process that manages the VM migration, to cut down the probability of SLA violation.

2. **Post-copy:**

   This scheme will migrate the VM from the host node which may fail in the future to a target destination node by immediately suspending the VM and capturing the minimum possible state to migrate the VM. This minimum state includes, non-pageable memory, register state, and CPU state. This minimum state is then moved to the target, and the VM is started in the target with this minimum state. If the guest OS requires a page to read/write which is not already copied, the migration system will copy it to the VMs memory on the fly. The end user applications may face reduced system performance during this stage. Meantime the migration system will copy the rest of the memory pages to the target server in the background. When all the pages are copied from the source server, the connection to the server will be terminated and the source VM can then be terminated.

3. **Hybrid approach:**

   These frameworks integrates the properties of both the previous methods to reduce the migration time, and improve system performance. It includes a finite pre-copy stage before moving on to migrating the VM and running the post-copy stage. This greatly reduces the page faults that may occur in the future as a large amount of the memory is already copied, so this method reduces the workload on the network and improves the application performance.

3.1.2 System rejuvenation

System rejuvenation is the operation of creating regular backups of the system. Then the system is repaired from errors and then it is booted back up again to achieve a refreshed system state. System rejuvenation is a process which is used to counteract the phenomenon known as *software aging* , which is the software performance reduction, and the increased failure rate that happens in many complex long-running software systems. With regards to VMs, the VMM or the hypervisor is mainly rejuvenated to have a healthy system state, and to maximize the system availability. In VM rejuvenation, other VM fault tolerance techniques such as VM live migration is used to achieve high service availability during the rejuvenation process. VM rejuvenation can be categorized mainly into three categories (Machida, Kim, and Kishor S. Trivedi 2010),

1. Cold-VM rejuvenation: All VMs in the server are turned off before stating the VMM rejuvenation process.

2. Warm-VM rejuvenation: All VMs in the server are paused before the VMM rejuvenation without clearing the VM memory images.

3. Migrate-VM rejuvenation: All VMs in the host PM is migrated to another PM before starting the VMM rejuvenation.

VM rejuvenation can again be categorized into two, according to the time interval of the rejuvenation process (Hasan and Goraya 2018),

1. Fixed time rejuvenation: The system rejuvenation events happen evenly at fixed time intervals.

2. Variable time rejuvenation: Two consecutive rejuvenation events may happen at different time intervals, which is decided by the system according to the working conditions of the system.

3.1.3 Self-healing

Self healing is the ability of software systems to automatically recover from faults by using fault recovery methods which contain system inspection tasks periodically. This type of systems can predict the erroneous conditions, and adjust to restore normal system state without human interaction. Self-healing mechanisms are inspired by the biological fact of how the living organisms survive in challenging environment conditions (Tu 2010). VM self-healing approaches use VM rejuvenation approaches, snapshot and restore , check-pointing approaches with machine learning to achieve self-healing capability from malware attacks, or faults.

3.2 Reactive approaches

Reactive FT strategies deal with faults after they have happened. The impact of these failures is reduced through maintenance procedures. The operations of reactive strategies are reaction-based rather than based on anticipation. They *react* after a fault has occurred. These strategies are generally conservative in nature and don't require the analysis of the system's behavior, thus avoiding any unnecessary overhead on the system. Reactive VM fault tolerance can be achieved in following ways:

3.2.1 Reactive migration

Reactive migration methods use VM migration techniques that I explained earlier under proactive migration schemes to migrate the VM to a suitable destination node after a fault has been detected. Generally, this technique is used in tolerating crash faults. But, unlike the preemptive migration frameworks where the VM is migrated as a result of a positive fault prediction result, reactive migration happens only after the detection of an actual fault.

Reactive migration schemes are generally used for soft deadline applications because the downtime of the system is comparatively high compared to proactive migration, which may lead to SLA violation. Still, the overhead associated with running a machine learning algorithm to predict fault occurrence is completely avoided in this mechanism.

3.2.2 Checkpoint and restart

In this type of schemes, the VM state is taken as a snapshot periodically and is saved to tolerate any future failure event. When a failure is perceived, the VM is restarted from the last saved checkpoint snapshot. This mechanism is one of the most used FT mechanisms in the cloud due to its dual applicability with other fault tolerance schemes. *Dual applicability* means that the checkpoint restart mechanism can be used as a standalone system and as an auxiliary system with other fault tolerance mechanisms, for example, checkpoint restart can be implemented with proactive migration, to provide a robust FT framework. The time period between taking snapshots can also be fixed or dynamic according to the reported rates of subcomponent failures.

3.2.3 Replication

In this scheme, the VM is replicated and the same task is executed on the replicas. If one/main instance goes down, the task execution will continue on in the other replica(s). This scheme is used in tolerating crash and byzantine faults. Even though running multiple replicas are expensive, this is a methodology that guarantees a near 100% service uptime, with little to no interruptions for the end user applications. There are two types of replication strategies that can be seen in the existing literature, .

Active replication is where both the primary node and the secondary (backup) node(s) perform the same operation in parallel , whereas in passive replication, only the primary node is responsible to perform operations and the secondary (backup) nodes(s) will get regular state updates from the primary node and takes over only when a fault is detected in the primary node.

3.2.4 Failure recovery

Failure recovery of VMs can be thought of as the last resort option for VM fault tolerance, and recovery can be used when all the fault tolerance mechanisms failed to successfully tolerate faults, and the system is in a failed, unresponsive, or completely stalled state. Some failure recovery schemes try to mitigate/recover from failures that might occur during the execution of other fault tolerance mechanisms.

Almost all the VM fault tolerance schemes that we have discussed up to now are based on having some part of the system functioning to perform the fault tolerance mechanisms, and these frameworks assume that the VMM/hypervisor is in fully working condition. However, the hypervisor is also another piece of software that is failure-prone, and failure of the hypervisor will cause an entire system failure. Frameworks such as ReHype, proposed by Le and Tamir (2011) provide mechanisms to recover hypervisor failures.

The table 2 presents a comparison between different Virtual Machine fault tolerance techniques.

Technique	Category	Overhead	Cost
Migration	Proactive & Reactive	Varied	Varied
System rejuvenation	Proactive	High	High
Self healing	Proactive	Average	High
Checkpoint and restart	Reactive	Average	High
Replication	Reactive	Low	V. High
Failure recovery	Reactive	Average	Varied

Table 2: Comparison between VM fault tolerance techniques

4 VM fault tolerance schemes

4.1 Proactive schemes

Proactive fault tolerance schemes continuously monitor the system and predicts possible future fault occurrence using a machine learning or a similar statistical approach. This section presents proactive schemes and also schemes which can be used in both proactive and reactive ways.

4.1.1 Proactive FT Using Preemptive Migration

Sub-category: Preemptive migration.

Engelmann et al. (2009) have presented a proactive FT scheme composed of four types of fault tolerance mechanisms depending on the monitoring ability, and the ability of processing the data of compute nodes. Each consecutive type has increasing complexity, but with higher accuracy and reliability of fault prediction and recovery. Their work is build on top of work carried out by Nagarajan et al. (2007) and Vallee, Charoenpornwattana, et al. (2008), on proactive fault tolerance. The proposed framework for FT in High Performance Computing (HPC) systems uses control mechanism based on a *feedback loop*. Each running VM is continuously monitored, and actions are taken to prevent VM failure by moving them from unhealthy to healthy nodes. The feedback loop is created by ongoing observation of system health, VM migration when faults are detected, and adjustment of VM allocation based on VM health status.

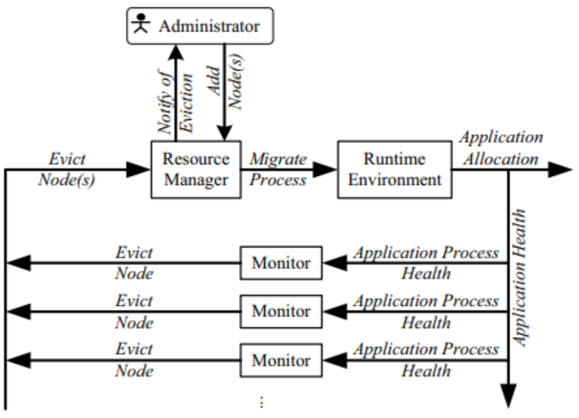

Figure 6: Type 1 (Engelmann et al. 2009)

Type 1 is the most basic form of fault tolerance. It involves having a monitoring system on every node that perpetually checks different health params of the system and applications. When a problem is detected, such as a faulty fan or dangerous temperature, the monitoring system alerts the Resource Manager to migrate that node. This type of fault tolerance mechanism tends to give false positives and false negatives. Figure 6 depicts an overview of the type 1 fault tolerance architecture.

Type 2 utilizes a filtering mechanism to post-process the raw sensor data with short term historical context to provide a better result of prediction of future faults, and this can reduce the false positive and false negative results. Figure 7 depicts the overview of the type 2 FT architecture.

In Type 3, the aggregated data from Type 2 system node health monitors is passed on to a Reliability Analysis component, which will perform node-wise and whole system detection of correlations of faults. As they suggests, the Reliability Analysis component can be a trend analyzer, or it could be a ML model. Figure

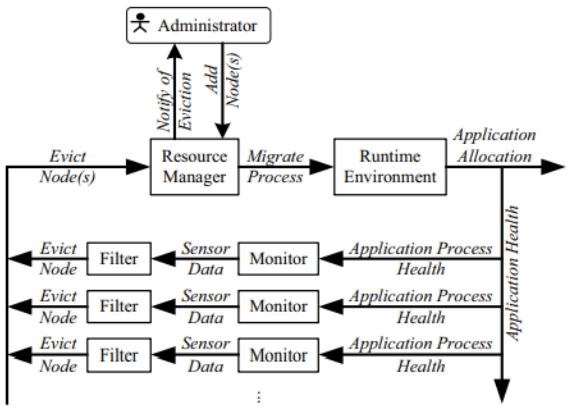

Figure 7: Type 2 (Engelmann et al. 2009)

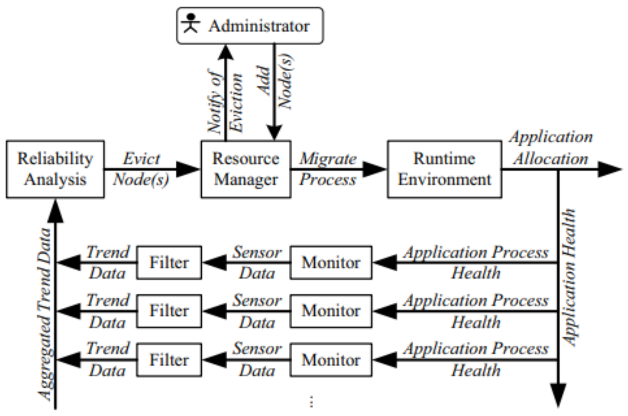

Figure 8: Type 3 (Engelmann et al. 2009)

8 depicts the overview of the type 3 FT architecture.

Type 4 provides an advance form of proactive fault tolerance architecture, where the Reliability Analysis component introduced in the Type 3 is further enhanced by utilizing a History database for storing prior system reliability patterns to match with the currently experiencing pattern to provide a better fault prediction. This database can also be used when conducting investigations on the system. Figure 9 depicts the overview of the type 4 FT architecture.

Key Features: The authors have provided 4 schemes for proactive FT in VMs using preemptive migration pattern. Any type can be implemented on a system according to the necessity.

Limitations: The presented frameworks are not properly evaluated. The authors have mentioned that only the first two frameworks are based on previous work, and the latter two frameworks are presented only with a theoretical evaluation, which might not be accurate in real world conditions.

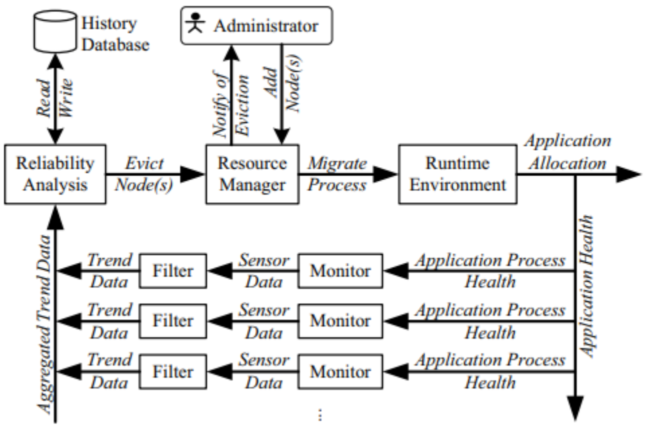

Figure 9: Type 4 (Engelmann et al. 2009)

4.1.2 Improved pre-copy approach

Sub-category: Preemptive/reactive migration.

Ma, F. Liu, and Z. Liu (2010) have proposed an improved version of pre-copy based migration scheme, initially presented by Clark et al. (2005). This scheme can be used both in preemptive and reactive migration schemes. Initially, this scheme moves all the memory pages, then proceeds to replicate the pages that were altered during the most recent cycles in a step-by-step manner. When the size of an application's Writable Working Set (WWS) diminishes, or when the maximum number of iterations is attained, the virtual machine halts, and both the CPU state and any changed pages from the last cycle are transmitted to the target node. (ibid.).

Ma, F. Liu, and Z. Liu (2010) have demonstrated that while the pre-copy strategy can effectively balance downtime and overall migration time, a maximum number of iterations must be set since the WWS may not converge when dealing with predominantly read-intensive workloads. Moreover, the efficiency of the migration may be diminished in the case of many moderately write-intensive workloads.

To address these challenges, the authors have proposed a solution, the use of a bitmap page that indicates which pages are frequently updated. This approach involves transmitting only these pages during the last rounds of the iteration process, ensuring that they are sent only once and avoiding duplicate transmissions.

They have implemented this concept using the Xen hypervisor , by modifying its existing migration mechanism, which migrates the memory pages by dividing it into three categories using three bitmaps,

- TO_SEND: Checks the dirty pages during the previous iteration, that is the pages which may sent in the current iteration.
- TO_SKIP: Checks skippable pages in the current iteration.

- TO_FIX: Checks which is to be transferred at last.

Ma, F. Liu, and Z. Liu (2010) proposed to add another bitmap, TO_SEND_LAST, to check frequently modified pages, which will get transferred during the last rounds of the iteration process. Compared to TO_FIX bitmap, TO_SEND_LAST bitmap will record constantly rewritten pages. Figure 10 gives an overview of the working principle of this scheme.

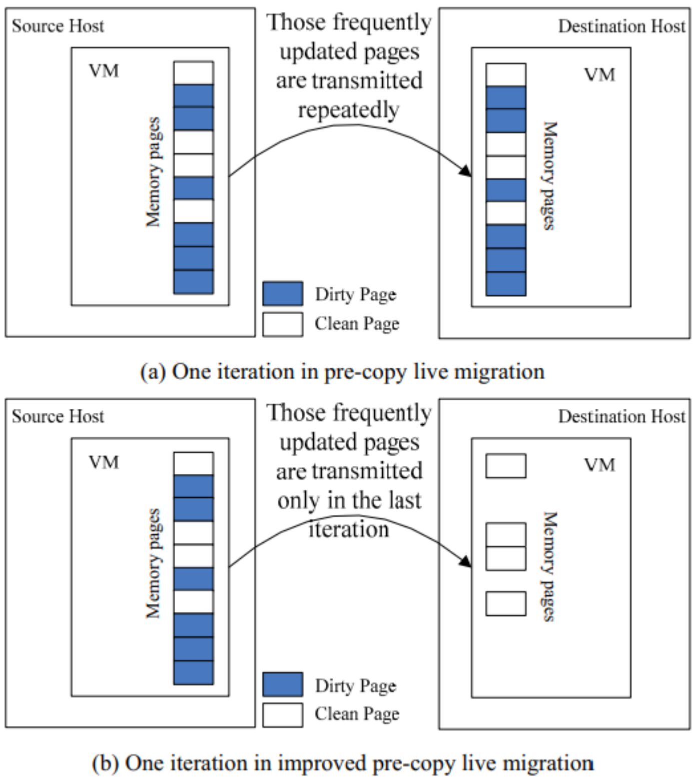
Source Host
VM
Memory pages
Those frequently updated pages are transmitted repeatedly
Destination Host
VM
Memory pages
Dirty Page
Clean Page
(a) One iteration in pre-copy live migration
Source Host
VM
Memory pages
Those frequently updated pages are transmitted only in the last iteration
Destination Host
VM
Memory pages
Dirty Page
Clean Page
(b) One iteration in improved pre-copy live migration

Figure 10: Xen pre-copy vs Improved pre-copy (Ma, F. Liu, and Z. Liu 2010)

They have compared their implementation with Xen pre-copy, and they have found that their improved pre-copy approach can reduce data transmission by 34.0%, and the migration time by 32.5% , and more importantly, their approach can complete the migration in no more than 5 iterations where as Xen takes around 30 iterations.

Key Features: Greatly improved existing pre-copy migration method by a small addition to the Xen live migration system. Added a massive performance boost to the Xen hypervisor without using any additional resources.

Limitations: The performance of this approach when testing with VMs that uses high memory write operations is comparatively low.

4.1.3 Post-Copy Migration

Sub-category: Preemptive/reactive migration.

While most live migration mechanisms use a pre-copy approach, M. R. Hines, Deshpande, and Gopalan (2009) have proposed a post-copy approach which is comparable to and in some cases surpasses the performance of pre-copy approaches. They have improved the traditional post-copy method, which I have explained earlier in this paper, by using several pre-paging methods to reduce page faults that will happen during the post-copy migration phase of the existing scheme. They have also proposed a mechanism named *Dynamic Self-Ballooning (DSB)* to remove the transmission of free memory pages, that can be used in both pre-copy and post-copy approaches.

The authors have used a combination of following pre-paging techniques in their adaptive pre-paging mechanism,

- Demand paging: Demand paging is considered to be one of the most straightforward techniques. When the virtual machine is launched at the target node, most of its memory accesses result in a page fault, which is then resolved by requesting the relevant page over the network.
- Active push: Active pushing means the source node proactively *pushes* memory, that is, the source node will send the VM pages even without a page fault at the destination VM.
- Pre-paging: Pre-paging will predict future occurrence of page faults and pushes them to the target before a page fault. This will greatly reduce the network page faults.

The main challenge in the pre-paging mechanism is to predict possible future network faulting pages, and to push them before network page fault happens at the destination node. They have described two approaches for this, namely, Bubbling with Single Pivot, and Bubbling with Multiple Pivots. Both approaches uses a pivot page, which refers to a page that caused a page fault in the target. The active push part of the algorithm begins at a central pivot page and then sends pages located symmetrically around that pivot in each iteration. The authors use the term "bubbling" to describe this process, as it resembles the way a bubble grows around a center point. Figure 11 graphically illustrates the bubbling pre-paging strategies.

Key Features: The authors have introduced a pre-paging mechanism using a single/multi pivot bubbling algorithm, which greatly reduced the network bound page faults during VM migration, also they have introduced the Dynamic self ballooning mechanism to get rid of the transmission of free memory pages.

Limitations: The test results of this approach on a VM with a read-intensive workload resulted in a poor performance.

4.1.4 Pre-Copy/Post-Copy Migration

Sub-category: Preemptive/reactive migration.

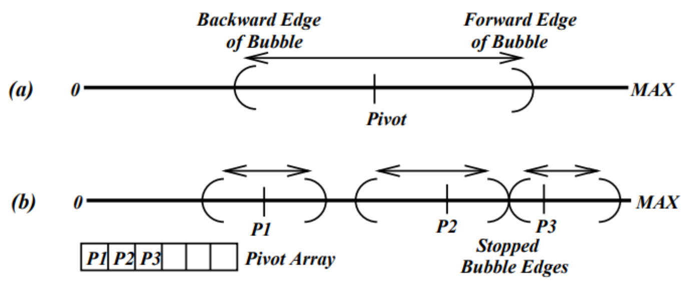

Figure 11: Bubbling pre-paging strategies (M. R. Hines, Deshpande, and Gopalan 2009)

Shribman and Hudzia (2013) have proposed a hybrid live migration framework which uses pre-copy and post-copy approaches together to effectively migrate memory intensive VMs. I have explained a high level overview of this hybrid live VM migration framework earlier under the section 3.1.1.

During the pre-copy rounds, if the rate dirtying of memory is faster than the network bandwidth, the pre-copy method would not properly work. To reduce the risk of getting into this issue, the authors have introduced a Least Recently Used (LRU) paging mechanism to migrate the less frequently used memory pages first. In addition to that they have proposed to use a faster data encoder known as Xor Binary Zero Run Length Encoding (XBZRLE), that will reduce the cost of sending pages over the network.

The authors have introduced three additional techniques to reduce the application Quality of Service (QoS) degradation:

1. Remote Direct Memory Access (RDMA): which was initially proposed by Comer and Griffioen (1990), to allow the network card to transmit data directly between the network and the application memory, without copying the data to data buffers in the OS.

2. Pre-paging: A mechanism similar to the scheme proposed by Ma, F. Liu, and Z. Liu (2010)

3. Linux MMU optimization: Integrating the network page fault system with Linux Memory Management Unit (MMU) to make that the only thread which is paused is the one waiting for the page fault.

They have implemented and tested their scheme usig QEMU and KVM using the Linux operating system. Their test results shows that the hybrid approach is on average 10% quicker than the traditional pre-copy system, and they have found out that pre-copy phase in the hybrid scheme greatly reduces the network page faults that happens in post-copy phase.

Key Features: The authors have introduced a hybrid live migration mechanism, which reduces the migration time and the VM downtime, by using both pre-copy and post-copy approaches with additional optimizations.

Limitations: The scheme is highly coupled with the Linux operating system, which could be generalized such that this approach can be used in other OSs as well.

4.1.5 Quick Eviction of VMs

Sub-category: Preemptive/reactive migration.

Fernando, Bagdi, et al. 2016 have proposed a framework based on pre-copy to significantly cut down the eviction time of VMs. The eviction time of a VM means the time it takes to migrate VM's state from the source VM to the target VM. The authors propose the idea of transferring snapshots of VM memory pages to the target node on a regular basis. In the occurrence of an initiation event , such as the detection of future failure of the system, any leftover pages in the VM that was modified since the last memory snapshot is migrated to the target, and then any other residual state are transferred to the target node to complete the eviction process. Since the final amount of residual memory is typically only a small portion of the total memory , it takes less time to completely migrate the VM. However taking regular snapshots will use network bandwidth, which will degrade the end user application performance. To reduce the network overhead, the authors have developed a mechanism to dynamically vary the snapshot interval according to the VM's workload during normal operation. Additionally, they have added a maximum page transmission limit, which will reduce the network load on the snapshotting process.

The authors have implemented this approach using QEMU and KVM, and tested it with the KVM pre-copy system on the same test environment. They have showed that the total eviction time of this framework is in between 6-246 ms, which is 8-18 times faster than the KVM pre-copy approach.

Key Features: The proposed proactive and incremental VM memory snapshot migration mechanism have reduced the VM migration time significantly, and have proved to migrate the VM in the blink of an eye.

Limitations: This framework requires a VM running in parallel to the source VM to capture the memory snapshots even before triggering the migration process, which is not ideal since the resources are limited in the cloud. In the case of a memory write intensive application, if a significant amount of memory gets dirtied after the final snapshot migration, the network overhead of the previous snapshot migrations will be purposeless, because most of the memory will get transmitted back again during the eviction period.

4.1.6 A Fast Rejuvenation Technique for VMs

Sub-category: System rejuvenation

Kourai and Chiba (2007) have proposed the Warm VM rejuvenation process which is used to proactively counteract the software aging . One such scenario related to VMMs is the VMM may have small memory leaks, which over time may get large enough to degrade the performance of the running VMs. Typically, to counteract the software aging, the system administrators manually reboots the VMM. This process, known as Cold-VMM rejuvenation shuts down all VMs, and then restarting them in a clean state after the VMM rejuvenation. The Cold-VMM rejuvenation approach results in the loss of any running transactions on the VMs, but it has the advantage of cleaning up all aging states.

The main concept of Warm VMM rejuvenation is to save the memory images of all VMs on memory before the VMM reboot and then reuse these images after the reboot. It is highly efficient because the time required for suspend is not related to the size of memory allocated to the VM. The resume mechanism simply unfreezes the frozen memory images after the VMM reboot. The VMM quick reload mechanism, will soft reboots the VMM, which will prevent the server from rebooting, preserving the saved frozen memory of VMs.

The authors have implemented this scheme using the Xen hypervisor, and compared it with cold VMM rejuvenation, and found out that their warm-VMM rejuvenation method reduced the VM downtime by 83%.

Key Features: Warm VMM rejuvenation will prevent the loss of application transactions running on Vms. The memory state of the VMs is saved in the physical memory of the same server, thus avoiding any use of migration mechanism or copy mechanism.

Limitations: This process preserves the aging states of hosted VMs by just suspending them, which are not removed by the VM rejuvenation process. All VMs will pause during the VMM rejuvenation process, thus this will cause reduction in QoS and SLA violation. This limitation is overcome by another scheme, *Migrate VM rejuvenation* by Machida, Kim, and Kishor S Trivedi (2013).

4.1.7 Autonomic Self-Heal Approach of VMs

Sub-category: Self-healing

Joseph and Mukesh (2019) have proposed a VM self healing scheme to automatically recover from malware attacks on VMs by taking memory snapshots of the VMs and detect malicious activity by extracting and analyzing the API calls made by the applications. The malicious activity is predicted by using different Machine Learning (ML) models, to categorize the memory images as affected or un-affected. If a VM snapshot is predicted as attacked, the VM is shutdown, and booted back again with the previous secure snapshot. This method can detect malicious activity not only in a single VM, but also in multiple VMs that may have networked using a Virtual network bridge. In the snapshotting process, the entire VM snapshot is taken including the virtual hard-disk snapshot to rollback the VM in case of malicious activity detection. However the authors only used the memory snapshot of the VM to detect malware.

The authors have considered several DoS/DDos attacks, probing, and rootkits, as security attacks for this analysis and ML algorithms training process. For possible malicious activity detection, the authors have developed automatic agent manager tool, which interacts with the nitro monitoring tool which monitors the system calls to detect possible malicious activity. If the nitro monitoring tool detects any malicious behavioural pattern, it alerts the agent manager tool, which will take a memory snapshot and pass it to the ML model to predict if it is actually a malware attack. For the ML algorithms, the authors have used several ML models to predict malicious activity by training them on Malicious and non-malicious API calls. The prediction is done by extracting system API call features from the memory snapshot.

Their test results showed that the Naive Bayes model produced 82.25% accuracy, Support Vector Machine (SVM) model produced 90.16% accuracy, with no false negatives, and the random forest model had 96.75% accuracy but with 6% false negatives.

Key Features: A self healing mechanism for VMs, by automatically detecting and fixing malicious attacks by using a VM snapshot inspection and roll back method.

Limitations: This scheme only scans the memory snapshot to detect/predict malicious activity and roll backs the VM to a previous snapshot which is secure, but this rolled back VM snapshot may have undetected malicious programs in the hard disk snapshot, which may again activate/attack. The authors have not considered using an anti-virus software and integrating with it to detect and possibly removing the malware before going into a system roll back.

4.2 Reactive schemes

Reactive FT schemes try to deal with faults after they have happened. The impact of these faults is reduced through FT techniques. These techniques are generally conservative in nature and don't require the analysis of the system's behavior, thus avoiding any unnecessary overhead on the system.

4.2.1 Checkpoint/Restart of Virtual Machines

Sub-category: Checkpoint and restart

Vallee, Naughton, et al. (2006) have proposed a checkpoint/restart mechanism for Xen hypervisor , which is built on top of primitive checkpointing mechanism of Xen, which only generated checkpoints using only the memory image, and not the file system. Their main intent was to implement a reactive VM checkpoint/restart mechanism to achieve fault tolerance, if any major failure has occurred to the system. They have proposed two sub systems, namely, the Checkpoint Manager , and the Resource Manager . The checkpoint manager is responsible for creating, compressing and restoring checkpoints, where as the resource manager is responsible for storing and retrieving the checkpoints either on the locally attached storage or a remote network storage. They also have proposed mechanisms for identifying and properly storing the checkpoints for future access.

Key Features: Introduced a VM checkpointing mechanism to restore VM state even after a major failure in the system, where the fault tolerance mechanism can even be configured to store checkpoints and restart the VMs across data centers.

Limitations: Authors have only presented a theoretical framework without any implementation that can be properly evaluated.

4.2.2 VMware FT for VMs

Sub-category: Replication
Scales, Nelson, and Venkitachalam (2010) from the VMware team have designed and implemented a fault tolerance system in VMware vSphere (VMware 2023), that replicates the operation of a primary VM with a backing VM on a different host. The authors have proposed the VMware deterministic replay , which is used to replicate the execution stages the VM on the backing VM. If the primary host/VM fails, the backing VM immediately takes over without any interruption to the end user applications.

To achieve this, the backup server should maintain a state that is almost the same as the primary server all the time. One approach is to transmit updates of all primary server states to the backup server almost constantly. However, transmitting the state, especially changes in memory, can require a large amount of bandwidth. To reduce the bandwidth, the authors have used the idea of modelling the VM as a deterministic state machine , initially proposed by Schneider (1990). To ensure accurate replay of VMs, hypervisors need to capture extra information related to non-deterministic operations such as interrupts. The authors have proposed a *logging channel* to transfer both the deterministic and non-deterministic operations captured by the hypervisor to the backup VM to accurately replay these operations. These operations are captured to a log file and the entries are streamed to the backing VM using the logging channel .

In a failure , the control is moved to the backup VM, and it is expected to execute differently from the primary VM due to non-deterministic events during execution, but it is important to ensure that no state or data that is visible to the end user is lost during fail over . The authors have implemented several mechanisms to ensure that the client will not see any inconsistencies. These mechanisms includes several disk locking mechanisms, and a proper mechanism to sync the logging and replaying . They have also implemented a method to restore fault tolerance after the backup VM takes over by creating a new backing VM on another host.

The authors have extensively tested their implementation, and they have shown that the network overhead of the log streaming in the logging channel is about 20 Mbit/s. And overall overhead of the fault tolerance mechanism is less than 10%. Since the log streaming mechanism has low bandwidth requirements, which makes it possible to use this scheme for safeguarding VMs against entire data-center failures, by creating the backup VM in a different data-center.

Key Features: Introduced a robust reactive VM fault tolerance framework using the replication method. The backup VM can continue operation immediately after the primary failed, with near zero downtime. Fault tolerance can even be achieved across data-centers.

Limitations: The implementation only supports fault tolerance for single processor VMs, ie, VMs with only one vCPU.

4.2.3 Live Migration Ate My VM

Sub-category: Failure recovery

Fernando, Terner, et al. (2019) have proposed a framework named *PostCopyFT* to recover VMs during post-copy live migration from destination or network failures. In post-copy approach, if the destination node or the network connection within the source VM and the target VM fails, it will make the VM unrecoverable (if there are no other FT strategies employed), because the updated state of the VM is divided between the target VM and the original VM. PostCopyFT works by sending incremental checkpoints of the VM from the target VM back to the source VM after starting the VM in the target node. Thus, if the target VM fails, the source VM can recover the state from the last successful checkpoint and continue operating normally.

Checkpoints can be sent periodically or on event based. After resuming the migrating VM at the target, PostCopyFT snapshots the VM's initial memory and execution state and transfers it to a checkpoint store , a key-value storage placed in memory at the original VM or another staging VM. Afterwards, PostCopyFT continues to capture any additional changes in the target VM.

The reverse checkpoints are significantly smaller in size than full VM checkpoints because they contain only the updated pages, CPU state, and I/O state since the last checkpoint. After the VM has been successfully migrated to the target using post-copy migration, the checkpointing mechanism can then be safely stopped. To check the liveliness of the destination VM, the authors have proposed to use heartbeat messages, and when the destination stops responding to the heartbeat messages, the migration is considered to be failed, and checkpoint restore mechanism is then started at the source node to continue operation.

To cut down the impact of the checkpoint mechanism, the authors have implemented it using a separate thread which runs in parallel with the VM. To reduce the VM downtime, authors have implemented two optimizations, namely, performing the VM checkpointing in two stages and storing checkpoints locally in memory until both the stages of VM checkpoints are complete.

The authors have evaluated their framework comprehensively under 6 different areas, and found out that their implementation of this framework in the QEMU and KVM resulted in comparable migration time to KVM post-copy migration, with minimal effect on end user applications.

Key Features: A new concept to recover VMs after a post-copy VM migration failure due to network or target node failure, which will unless otherwise become a complete VM failure.

Limitations: This scheme is not suitable for instances where the post-copy migration started because of a possible future source node failure, or it is already partially failed.

5 Conclusion and future work

Fault tolerance of virtual machines has been one of the major issues when managing multiple servers running mission critical, delay sensitive real-time applications, and especially in cloud computing. In this paper, I have discussed different fault types, their causes, and various virtual machine fault tolerance techniques in a systematic manner. I have also given an overview of 10 eminent virtual machine fault tolerance schemes/frameworks, along with their key features and limitations. Based on my survey on existing literature on the area, I recommend following probable areas of future research,

- Optimizing the overhead of proactive VM fault tolerance frameworks
- Implementing hybrid frameworks that will use two or more VM fault tolerance techniques together to provide robust VM fault tolerance
- Optimizing the cost of temporary VM checkpoint storage during migration
- Reducing the impact of VM migration on end user applications

References

Ahmad, Raja Wasim et al. (2015). "Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues". In: *The Journal of Supercomputing* 71.7, pp. 2473–2515.

Birke, Robert et al. (2014). "Failure Analysis of Virtual and Physical Machines: Patterns, Causes and Characteristics". In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 1–12. DOI: 10.1109/DSN.2014.18.

Bulao, Jacquelyn (2023). *How Many Companies Use Cloud Computing in 2023? All You Need To Know.* URL: https://techjury.net/blog/how-many-companies-use-cloud-computing/#gref (visited on 01/22/2023).

Clark, Christopher et al. (2005). "Live migration of virtual machines". In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 273–286.

Comer, Douglas E and James Griffioen (1990). "A new design for distributed systems: The remote memory model". In.

Engelmann, Christian et al. (2009). "Proactive Fault Tolerance Using Preemptive Migration". In: *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pp. 252–257. DOI: 10.1109/PDP.2009.31.

Fernando, Dinuni, Hardik Bagdi, et al. (2016). "Quick eviction of virtual machines through proactive snapshots". In: *2016 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, pp. 156–157.

Fernando, Dinuni, Jonathan Terner, et al. (2019). "Live migration ate my vm: Recovering a virtual machine after failure of post-copy live migration". In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications.* IEEE, pp. 343–351.

Hasan, Moin and Major Singh Goraya (2018). "Fault tolerance in cloud computing environment: A systematic survey". In: *Computers in Industry* 99, pp. 156–172.

Hines, Michael and Kartik Gopalan (Mar. 2009). "Post-copy based live virtual machine migration using pre-paging and dynamic self-ballooning". In: pp. 51–60. DOI: 10.1145/1508293.1508301.

Hines, Michael R, Umesh Deshpande, and Kartik Gopalan (2009). "Post-copy live migration of virtual machines". In: *ACM SIGOPS operating systems review* 43.3, pp. 14–26.

IBM (2023). *What is virtualization?* URL: https://www.ibm.com/topics/virtualization (visited on 01/20/2023).

Jhawar, Ravi and Vincenzo Piuri (2012). "Fault tolerance management in IaaS clouds". In: *2012 IEEE First AESS European Conference on Satellite Telecommunications (ESTEL).* IEEE, pp. 1–6.

— (2017). "Chapter 9 - Fault Tolerance and Resilience in Cloud Computing Environments". In: *Computer and Information Security Handbook (Third Edition).* Ed. by John R. Vacca. Third Edition. Boston: Morgan Kaufmann, pp. 165–

181. ISBN: 978-0-12-803843-7. DOI: https://doi.org/10.1016/B978-0-12-803843-7.00009-0. URL: https://www.sciencedirect.com/science/article/pii/B9780128038437000090.

Joseph, Linda and Rajeswari Mukesh (2019). “To Detect Malware attacks for an Autonomic Self-Heal Approach of Virtual Machines in Cloud Computing”. In: *2019 Fifth International Conference on Science Technology Engineering and Mathematics (ICONSTEM)*. Vol. 1, pp. 220–231. DOI: 10.1109/ICONSTEM.2019.8918909.

Khalili, Joel (2020). *Here’s how much cash Google lost due to last week’s outage.* URL: https://www.techradar.com/news/google-blackout-saw-millions-in-revenue-vanish-into-thin-air (visited on 01/28/2023).

Kourai, Kenichi and Shigeru Chiba (2007). “A Fast Rejuvenation Technique for Server Consolidation with Virtual Machines”. In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)*, pp. 245–255. DOI: 10.1109/DSN.2007.6.

Lawrence, Andy (2022). *2022 Outage Analysis Finds Downtime Costs and Consequences Worsening as Industry Efforts to Curb Outage Frequency Fall Short.* URL: https://uptimeinstitute.com/about-ui/press-releases/2022-outage-analysis-finds-downtime-costs-and-consequences-worsening (visited on 01/28/2023).

Le, Michael and Yuval Tamir (2011). “ReHype: Enabling VM survival across hypervisor failures”. In: *ACM SIGPLAN Notices* 46.7, pp. 63–74.

Ma, Fei, Feng Liu, and Zhen Liu (2010). “Live virtual machine migration based on improved pre-copy approach”. In: *2010 IEEE International Conference on Software Engineering and Service Sciences.* IEEE, pp. 230–233.

Machida, Fumio, Dong Seong Kim, and Kishor S Trivedi (2013). “Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration”. In: *Performance Evaluation* 70.3, pp. 212–230.

— (2010). “Modeling and analysis of software rejuvenation in a server virtualized system”. In: *2010 IEEE Second International Workshop on Software Aging and Rejuvenation*, pp. 1–6. DOI: 10.1109/WOSAR.2010.5722098.

Nagarajan, Arun Babu et al. (2007). “Proactive fault tolerance for HPC with Xen virtualization”. In: *Proceedings of the 21st annual international conference on Supercomputing*, pp. 23–32.

Polze, Andreas, Peter Tröger, and Felix Salfner (2011). “Timely Virtual Machine Migration for Pro-active Fault Tolerance”. In: *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pp. 234–243. DOI: 10.1109/ISORCW.2011.42.

Scales, Daniel J, Mike Nelson, and Ganesh Venkitachalam (2010). “The design of a practical system for fault-tolerant virtual machines”. In: *ACM SIGOPS Operating Systems Review* 44.4, pp. 30–39.

Schneider, Fred B (1990). “Implementing fault-tolerant services using the state machine approach: A tutorial”. In: *ACM Computing Surveys (CSUR)* 22.4, pp. 299–319.

Shribman, Aidan and Benoit Hudzia (2013). “Pre-copy and post-copy vm live migration for memory intensive applications”. In: *Euro-Par 2012: Parallel Processing Workshops: BDMC, CGWS, HeteroPar, HiBB, OMHI, Paraphrase, PROPER, Resilience, UCHPC, VHPC, Rhodes Islands, Greece, August 27-31, 2012. Revised Selected Papers 18*. Springer, pp. 539–547.

Sun, Dawei et al. (2012). “Modelling and evaluating a high serviceability fault tolerance strategy in cloud computing environments”. In: *International Journal of Security and Networks* 7.4, pp. 196–210. DOI: `10.1504/IJSN.2012.053458`. eprint: `https://www.inderscienceonline.com/doi/pdf/10.1504/IJSN.2012.053458`. URL: `https://www.inderscienceonline.com/doi/abs/10.1504/IJSN.2012.053458`.

Sun, Xiaoyi et al. (2019). “System-Level Hardware Failure Prediction Using Deep Learning”. In: DAC ’19. Las Vegas, NV, USA: Association for Computing Machinery. ISBN: 9781450367257. DOI: `10.1145/3316781.3317918`. URL: `https://doi.org/10.1145/3316781.3317918`.

Tu, Huan-yu (2010). “Comparisons of self-healing fault-tolerant computing schemes”. In: *Proceedings of the World Congress on Engineering and Computer Science*. Vol. 1.

Vallee, Geoffroy, Kulathep Charoenpornwattana, et al. (2008). “A framework for proactive fault tolerance”. In: *2008 Third International Conference on Availability, Reliability and Security*. IEEE, pp. 659–664.

Vallee, Geoffroy, Thomas Naughton, et al. (2006). “Checkpoint/restart of virtual machines based on Xen”. In: *Proceedings of the High Availability and Performace Computing Workshop (HAPCW 2006), Santa Fe, New Mexico, USA*.

VMware (2023). *VMware vSphere Documentation*. URL: `https://docs.vmware.com/en/VMware-vSphere/index.html` (visited on 02/16/2023).

Index