This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

TOWARD A SMART CLOUD: A REVIEW OF FAULT-TOLERANCE METHODS IN CLOUD SYSTEMS

Mukosi A. Mukwevho and Turgay Celik

This paper presents a comprehensive survey of the state-of-the-art work on fault tolerance methods proposed for cloud computing. The survey classifies fault-tolerance methods into three categories: 1) ReActive Methods (RAMs); 2) PRoactive Methods (PRMs); and 3) ReSilient Methods (RSMs). RAMs allow the system to enter into a fault status and then try to recover the system. PRMs tend to prevent the system from entering a fault status by implementing mechanisms that enable them to avoid errors before they affect the system. On the other hand, recently emerging RSMs aim to minimize the amount of time it takes for a system to recover from a fault. Machine Learning and Artificial Intelligence have played an active role in RSM domain in such a way that the recovery time is mapped to a function to be optimized (i.e by converging the recovery time to a fraction of milliseconds). As the system learns to deal with new faults, the recovery time will become shorter. In addition, current issues and challenges in cloud fault tolerance are also discussed to identify promising areas for future research.

I. INTRODUCTION

THE use of cloud computing [1] has witnessed a significant amount of growth over the past decade. More services are becoming available for use over the clouds. Cloud architectures mainly involve sharing of computing resources amongst different users. The shared resources include applications, hardware, and software platforms. Cloud architectures are normally comprised of three key layers, i.e. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Faults can occur at any of these layers, however the algorithms for systems to recover from faults are mainly described and implemented at software level. Depending on the nature of the fault, a manual intervention by system engineers is required in order to recover the system fully.

A typical cloud architecture is depicted in Table I. At the lowest layer is the physical hardware which is virtualized and partitioned into different hosts, this layer is referred to as Infrastructure as a Service (IaaS). The virtualized hosts are made accessible to end-users through the virtual machine management system. The next layer is the Platform as a Service (PaaS). PaaS provides a software development framework in which consumers can build their own applications. PaaS also includes a set of predefined Operating System (OS) and application servers for the consumer to use. Some large corporates can develop specific software solutions and make them available for consumers to use, such a layer is called Software as a Service (SaaS), e.g. the email application developed by Google. SaaS includes software packages that offer solutions to businesses such as Enterprise Resource Planning (ERP) systems, e.g. SAGE, SAP and Oracle. In such situations, SaaS therefore provides a mechanism for corporates to rapidly implement ERP solutions into their business and enjoy low cost of ownership. The low cost is a result of low license fees, no hardware to procure and no need for maintenance teams to be deployed. See Table I for further details.

Deployment / Access Models	Cloud Services	Resources
Private Clouds Public Clouds Community Clouds Hybrid Clouds	Software as a Service (**SaaS**)	**Software Applications:** Web Services APIs, Business Applications Packages
	Platform as a Service (**PaaS**)	**Platform Frameworks:** Software Development Frameworks, Storage Packages
	Infrastructure as a Service (**IaaS**)	**Virtual Infrastructure:** Virtual Machines (VMs), Virtual Storage Blocks
		Physical Infrastructure: Hardware CPU, Disk Memory, Bandwidth

TABLE I: A typical cloud architecture

As mentioned above, cloud computing enables developers and corporates to deploy software, applications and infrastructure that can be shared by various users locally or remotely, using devices with different computing capabilities to access such services over the local network or Internet. Many arguments have been developed in favor of the cloud computing revolution, of which the economic and mobility arguments stand out. However, despite these benefits of the cloud computing revolution, many critical issues still pose serious challenges. These challenges include security, strong and reliable connectivity and fault-tolerance at large. The focus of this paper is on fault tolerance of cloud services. The reliability of could services is directly impacted by failures that occur in cloud infrastructures.

Furthermore, cloud services are hosted on virtual infrastructure which is hosted on some industry state-of-art virtualization platforms such as VMware, XEN, Azure and VirtualBox. These technologies provide a number of features (such as vShephere HA and DR and Live Migration) as part of their fault tolerance offering. The ultimate objective of this research is to define a framework for cloud fault tolerance. It is not aimed at narrowing down to a specific cloud technology implementation. However, the features that are provided by the available virtualization platforms will be taken into con-

Mukosi A. Mukwevho (email: mukosi@gmail.com) and Turgay Celik (email: celikturgay@gmail.com) are with the School of Computer Science and Applied Mathematics, University of Witwatersrand, Johannesburg, South Africa. Turgay Celik is also with the School of Information Science and Technology, Southwest Jiaotong University, Sichuan Chengdu, China.

Manuscript received August 25, 2017 and revised January 19, 2018.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

sideration during the design phase of our framework.

Much focus on fault tolerance of cloud systems has been directed towards innovating ways for guaranteeing the availability of central cloud systems, limited amount of research has been done on fault-tolerance to improve the reliability of cloud systems. In this paper, we seek to conduct a review of various methods that have been used for managing faults in cloud systems. The study is aimed at identifying the most commonly used methods and areas where there are gaps. Where we find gaps, some suggestions on possible way(s) of addressing such gaps will be proposed. To conduct this review, we therefore require a framework to be used as guidance for this research. We find the idea of classifying fault management systems as suggested by *Abid et al.* [2] quite interesting. In this paper, fault management approaches in cloud environments is classified into three categories, namely, 1) ReActive Methods (RAMs); 2) PRoActive Methods (PRMs); and 3) ReSilient Methods (RSMs).

The approaches falling into the first category (RAMs) are based on classical techniques of managing faults in distributed systems, such as *replication*, *checkpointing/restarting*, *detection* and *recovery*. Here, a fault is normally managed through implementation of some sequential logic in a step-by-step process which enables a system to recover from failures. The second category (PRMs) is a class of methods which primarily implement failure avoidance approach. These methods are characterized by the use of techniques such as *monitoring*, *prediction* and *preemption*. Under normal operation, in PRMs, the system state is constantly being monitored, the steps to avoid a fault are invoked as soon as the possibility of a system fault is predicted. In PRMs, failure prediction is done by using statistical modeling mostly offline.

Thirdly, RSMs have recently attracted more attention from researchers. RSMs appear to be the future direction for fault tolerance in cloud systems. By definition, the resiliency of a system is the measure of how quick the system can recover and continue to operate normally after there has been a system outage or fault. The outage could be due to the equipment failure, power outage or disruption. Normally, RSMs include techniques dealing with ability to respond to the client despite the failure, monitoring of system state, ability to learn and adapt from faults and prediction. In RSMs, the learning and adapting of the system is based on Machine Learning (ML) or Artificial Intelligence (AI).

The ability of RSMs to guarantee a response to the client is directly related to the reliability of the system which forms part of the Quality of Service (QoS). Currently, most research efforts in cloud fault management are centered around improving the reliability of cloud services. Most research work on cloud reliability is focused on improvements and optimizations on checkpointing, storage usage and virtual machine (VM) migration[3, 4, 5, 6]. Limited amount of research has been carried out in exploring the ability for cloud systems to learn and adapt from faults (or a Smart Cloud which can learn from its experience).

The operation of PRMs and RSMs might appear to be similar, i.e. they both implementing strategies that reduce or eliminate the effect of failures on cloud systems. The difference between PRMs and RSMs is quite subtle. In RSMs, the model and learning structures are continuously updated to cater for changing dynamics of cloud systems. This is where RSMs take action to mitigate flows of PRMs by adaptively learning their computing environment characteristics. Although PRMs implement prediction of failures, there is no learning and adapting of methods.

Smart Cloud is an emerging research field which explores the use of Machine Learning techniques (such as supervised, unsupervised and reinforcement learning) for handling faults in cloud systems. In order to support this emerging research field, there is a need for reviewing existing fault-tolerance methods. The rest of this paper is structured as follows. A brief overview of the fault tolerance (FT) model is presented in Section II. Some challenges on cloud FT are described in Section III. An overview of recent cloud FT studies is presented in Section IV. A *taxonomy* and reviews of RAMs, PRMs and RSMs are provided in Section V. Future directions are presented in Section VI. Finally, we conclude this review paper at Section VII.

II. Cloud Fault Tolerance Model

A fault can be any event that occurs in a system which impacts the normal operation of the system. Generally, a fault is the fundamental impairment of the normal system operation, and they cause errors. Errors in turn cause system failure. Fault tolerance is a measure of the ability of a system to continue serving its client requests in the presence of a failure.

There are four kinds of system faults: 1) *transient faults*; 2) *intermittent faults*; 3) *permanent faults*; and 4) *Byzantine faults*. We briefly describe each kind as follows:

- *Transient faults* occur as a result of some temporary condition affecting the system. In cloud computing, this include conditions such as network connectivity issues and service unavailability. Transient faults disappear as soon as they are rectified.
- *Intermittent faults* occur randomly at irregular intervals and they normally resemble malfunctioning of a system, hardware device or component. Intermittent faults are extremely difficult to diagnose and resolve permanently. An example could be a hard disk that seize to function as a result of temperature fluctuations but it returns to normal at some stage.
- *Permanent faults* continue to exist until the root cause is resolved. They normally occur as a result of a complete malfunction of a system component, and it is normally straight forward to diagnose them.
- *Byzantine faults* are the hardest to detect for which a system component might behave arbitrarily and provide incorrect results to the client. This can be caused as a result of a corrupted internal state of a system, data corruption or incorrect network routes. They are extremely hard and expensive to handle. In certain circumstances, replicas of system components are used and a voting technique implemented to choose the correct answer.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

III. Fault Tolerance Challenges in Cloud Systems

A number of challenges needs to be considered when providing for fault tolerance in cloud systems. Some of the key challenges are as follows:

- *Heterogeneity and the lack of standards*–Cloud computing enables the commoditization of computing resources. As a result, different hardware and OS vendors have deployed clouds based on their own architectures. Therefore, it is possible to deploy a single large cloud system having components hosted on heterogeneous platforms. This puts pressure on designing fault tolerance solution as they must consider aspects of each OS vendor in the overall fault tolerance solution.
- *Need for automation*–The future is smart and it requires automation. The number of VMs hosting cloud systems is growing exponentially. Management of such systems by humans will become virtually impossible. As a result, there is a need to consider automation to manage the fault tolerance solutions for large systems. The main challenge for automation is the lack of generic frameworks (APIs) that can be applied to any cloud system for FT with very little effort (need for plug and play FT).
- *Downtime in the clouds*–Cloud architectures are made of multiple data centers distributed geographically and managed by different vendors. A complete downtime on one data center can affect many organizations. Each organization has different Service Level Agreements (SLAs) for clouds and the fault tolerance provider must ensure the SLAs for all organizations are met.
- *Consideration for RPO and RTO*–Recovery Point Objective (RPO) is the amount of data that can be lost if a server had a failure. Recovery Time Objective (RTO) is the time that it takes the systems to be up and running after a failure. Considering these definitions, in any system, the objective will be to reduce both RPO and RTO to the absolute minimum achievable. Resilient methods (discussed later in this paper) can be modeled to address continuous minimization of both RPO and RTO. Through the nature of resilient methods operations, their learning function can be defined in such a way that it minimize both RPO and RTO.
- *Workloads in the cloud*–There are two types of workloads in the clouds, i.e. *cloud-enabled* and *cloud-native* workloads. Cloud-native, also referred to as "born in the cloud", refers to applications built using the cloud model fully. Cloud-native applications are composed of multiple services and each service is elastic, resilient, and can be used to compose other applications. Cloud-native workload refers to computing workload that is generated by pure cloud-native applications. In some instances, it is not possible to migrate all components of an application to the cloud, this results some components of an application being hosted in the cloud and some being hosted internally within an enterprise, this is normally referred to as cloud-enabled. Cloud-enabled workload refers to workload generated by applications that comply to the cloud-enabled model. The cloud-native and cloud-enabled workloads have disjoint characteristics which require some consideration when implementing fault-tolerance (FT). Both workloads can be mapped to one fault tolerance framework which manages all components as a unit. In other words, the FT mechanism must include monitoring of all cloud-based and locally-based components in one view. In such cases, both proactive and resilient methods should be applicable to deal with fault tolerance requirements of both cloud-native and cloud-active models.

IV. Fault Tolerance and Reliability in the Clouds

Cloud systems are normally hosted in data centers which are built using a large number of commodity servers. Virtualization technology is used to provide computing resources which are then amongst many cloud users. This virtualization of resources leads to complex infrastructure designs which expose the hardware into conditions that they were not originally designed for and leading to failures. Failures can occur at hardware, system (Host or VM), software or operator level. The faults in a cloud system can lead to failures that result in a catastrophic outage of the system and therefore affecting the reliability of the cloud system.

The reliability of a cloud system is the measurement of how well the cloud system delivers the service to the user under the predefined conditions. Such conditions are normally defined as QoS which forms part of the contract between the cloud service provider and the client (or user). The reliability of cloud systems ultimately depends on how well the VMs hosting the services can tolerate faults. A number of studies have been done on VM fault tolerance [3, 4, 5, 6, 7, 8]. In general, the techniques used for fault tolerance in cloud systems involve optimizations on checkpointing [4], redundancy [3, 7] and network bandwidth [9, 10]. Checkpointing can occur at process level or VM level. A process or VM state is constantly been saved during execution. In the event of a process failure, execution is restored from the checkpoint onwards and not from the beginning. In the event of a VM failure, the VM image is restored to another machine and processing continues from the checkpoint of the failed VM.

Most studies on reliability of cloud systems are focused on optimizing the checkpointing algorithm as well as the VM redundancy. *Liu et al.* [3] and *Zhou et al.* [4] describe a redundant VM approach which takes the network topology into consideration when selecting a set of VM-hosting servers with the objective of minimizing network resource consumption. *Zhou et al.* [5] propose a solution for reducing the storage used during the checkpointing of VMs. Finally, *Zhou et al.* [5] provide a survey of studies on enhancements of cloud reliability.

In most cases, the adoption of cloud services involves the migration of existing systems hosted at an organization data center to the cloud environment. Organizations are attracted by the value proposition of cloud hosting as a result of features such as low maintenance costs, high scalability, and pay-per use models. Besides saving the hosting costs, such migrations should also improve on the overall reliability of the system. It is not an easy decision to migrate some well established enterprise systems to the cloud.

Despite the perceived benefits of moving systems into the cloud, organizations still have to make a pragmatic decision

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

based on a scientific method required to inform such decisions. Also, important considerations such as system security (as governed by the organization's security policy) needs to be looked at. Some system components are just better off running in the local private environment. Some guidelines are therefore required to assist with implementing the optimal cloud migration structure.

Qiu et al. [11] introduce a reliability-based framework (ROCloud) which can be used as part of decision making when considering cloud migration on systems. ROCloud is made up of two algorithms (ROCloud1 and ROCloud2) which are used to rank applications based on their structure and historical reliability information. ROCloud1 and ROCloud2 are used to rank ordinary and hybrid applications, respectively. The ranking results are used to automatically select the optimal fault tolerance strategy to use. The framework uses four common fault tolerance strategies, i.e. Recovery Block, N-Version Programming, Parallel and VM Restart. Each strategy ranks each system component based on three parameters: *response time*, *resource cost*, and *failure rate*. Experiments were conducted and the results showed a significant improvement just by filtering few error-prone components and moving them to the cloud.

V. Taxonomy of Fault Tolerance Methods

Our study seeks to provide a review of fault-tolerance methods which have been proposed for cloud computing. We also identify the contributions of these different methods to the literature. We reviewed a number of key papers including conference proceedings, journal articles, research thesis, and book chapters on fault-tolerance methodologies. Although we tried our best to classify papers into categories, some of the papers where not possible to fit into one category.

Our taxonomy groups fault tolerance methods into three categories, i.e. *reactive methods, proactive methods and resilient methods*. A number of fault tolerance review studies have classified fault tolerance methods into reactive and proactive categories only [3, 12, 13, 14, 15, 16]. To our knowledge, this paper is the first to group fault-tolerance methods into these three categories. There has been a growing interest on intelligent systems that can learn through interaction with the environment and adapt their fault tolerance accordingly. The future direction on fault tolerance in the clouds is moving toward intelligent and resilient methods. A number of fault tolerance studies have already applied Machine Learning in various ways to introduce intelligence and resiliency [17, 18, 19, 20, 21, 22, 23, 24, 25]. Therefore, we include resilient methods as a third category on its own as follows:

- Reactive Methods (RAMs): The emphasis is mainly on system **recovery**. The state of the system is constantly saved and used during the recovery process. The key techniques used are replication, checkpointing and restarting.
- PRoactive Methods (PRMs): Mainly concerned with **preventing** complete outage of the system. These methods work by constantly monitoring the system and conducting fault predictions so that the effects of faults are prevented well before they occur. The key techniques used are monitoring, prediction and reallocation of cloud resources.
- Resilient Methods (RSMs): These methods share a number of common features with proactive methods. RSMs operate by predicting faults and implement methods to avoid or minimize the impact of such faults on the system. In addition to monitoring and prediction, resilient methods also adapt (fine tune) the system fault tolerance capability by incorporating intelligent learning through interacting with the hosting environment. This is where RSMs significantly differ from PRMs.

It should be noted further that the categorization of FT methods above does not only apply to cloud computing. This is a categorization that has been used by most survey-type of studies on fault tolerance in general. We see cloud computing as an evolution of grid computing which in turn is also a special case of distributed computing. The techniques used in each category are listed in the taxonomy as presented in Table II. The pros and cons of each category are described in Table III. The following sections provide a brief overview of selected papers under each category.

A. Reactive methods (RAMs)

Reactive methods (RAMs) are used to mitigate the effect of failures after they have occurred. Based on the current literature, key techniques used for reactive fault-tolerance include *Checkpointing/Restarting*, *Replication*, *SGuard*, *Retry*, *Custom exception handling*, *Task resubmission* and *Rescue workflow*. This section provides a review of the selected papers on RAMs.

1) Checkpointing/Restarting

The checkpointing/restarting techniques work by continuously saving system state, in the event of failure, the job is started from the most recent state. These techniques are suitable for long running jobs. The following paragraphs will summarize a number of papers on checkpoint based algorithms, i.e. [26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37]. These papers where chosen based on the variation on how they incorporate checkpointing in their solution.

Okamura et al. [27] propose a dynamic checkpointing scheme based on reinforcement learning. This technique can become useful in situations where the distribution of system failures is unknown. Firstly, the checkpointing problem is modeled as a semi-Markov decision process. Secondly, the representative reinforcement learning algorithm (known as Q-learning algorithm) is applied. Q-learning allows for the construction of an adaptive checkpointing scheme. The algorithm is formed by agents which adapt to their environment through learning and interaction experiences.

Moody et al. [29] describe a novel multi-level checkpointing system. This approach aims to reduce the checkpointing cost of growing high performance computing (HPC) systems. The system defines a set of L checkpointing mechanisms. Each level represents a checkpointing mechanism with different cost and resilience level which ultimately maps to the type of storage used, e.g. local memory, USB, remote memory, using a software RAID, local SSD or remote file system. The first level 1 is the level with the lowest cost and the last level L is the level with the highest cost. The lower levels employ

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

Category	Algorithm/Technique	
	Name	**Description**
Reactive methods (**RAMs**)	Checkpointing/Restarting	The system state is continuously being saved, in the event of failure, the job is started from the most recent state. Suitable for long running jobs [26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37].
	Replication	Some system components are replicated and deployed simultaneously across different resources. The aim is to make the system robust and to guarantee the execution of jobs [9, 26, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47].
	Retry	The system retries failed requests on the same resource multiple times [48, 49] .
	Task resubmission	The system resubmits the failed task to the same or different resource for execution [44].
	Custom exception handling	The system handles specific faults by executing pieces of code implemented by software developers [50].
	Rescue workflow	The system allows the workflow to continue even if the task fails until it becomes impossible to continue without attending to the failed task [51, 52].
	Load balancing	The system incorporates load balancing to distribute the workload equally across the nodes that are forming a cloud system. The fair distribution or workload ensures the nodes are not overwhelmed and therefore prevent a faulty state of the system. This technique is commonly applied in multi-clustered applications [53, 54, 55, 56].
	N-Version and recovery block	The system is developed by using the multi-version programming model. Multiple programs that are functionally equivalent are developed by separate teams working from the same set of specifications. The teams work independently and do not communicate. The idea of N-version is based on the fact that independently developed programs greatly reduce the probability of similar faults in two or more versions. Recovery Block (RB) utilizes different representations of input data to provide the tolerance of design faults [57, 58, 59, 60, 61].
Proactive methods (**PRMs**)	Software rejuvenation	The system is designed for periodic restarts. The system is restarted with a clean state each time [62, 63, 64].
	Self-healing	This technique enables the system to automatically detect, diagnose and repair software and hardware faults. Such systems are made of multiple components that are deployed on multiple VMs [65, 66, 67, 68].
	Preemptive migration	This technique prevents the system components that are about to fail from impacting the performance of the system. This is achieved through monitoring and by moving components away from nodes that are about to fail to run on more stable nodes [3, 13, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78].
	Prediction	The system is empowered to predict faults before they occur. This gives the cloud system a chance to take corrective steps to avoid or reduce the impact of the fault [13, 17, 79, 80, 81, 82, 83, 84].
	Monitoring (Feedback loop)	Using monitoring, the system is able to monitor a set of state variables during runtime. The state variables are filtered and supplied to some policy manager through the feedback loop mechanism [14, 85].
	SGuard	This technique is meant for video streaming systems. It enables a real-time video streaming system to perform rollback and recovery and with very little turbulence to the video streaming operation [18].
Resilient methods (**RSMs**)	Machine learning (ML)	The system attributes are modeled using ML techniques, specifically reinforcement learning (RL). Cloud systems are empowered to learn by interacting with their environment and adapt their fault handling strategies accordingly. Reinforcement Learning appears to be the most common technique used in the FT domain [19, 20, 21, 22, 23, 24, 25, 86].
	Fault induction	The resiliency of the system is tested on catastrophic errors. Specific errors, to simulate a catastrophic situation, are introduced to a system and observations are made on how the system responds. If the system crashes, mechanisms to handle such errors are implemented and the resiliency of the system is significantly improved as a result. Fault induction is a fairly recent technique being used by corporates developing cloud systems such as Amazon and Google [87, 88].

TABLE II: A taxonomy of fault tolerance methods for cloud computing

lightweight checkpoints that have lower overhead costs and are therefore ideal to handle most common failure modes. Similarly, the higher levels have expensive checkpointing costs and are used for less frequent failure modes. A Scalable Checkpoint/Restart (SCR) library is used to implement the system and it can save checkpoints to RAM, Flash or disk on the computing nodes. The system is further modeled using a probabilistic Markov model which is useful in predicting performance on current and future systems. The overall results showed a two reduction of load on parallel file system of current and future systems. *Di et al.* [30] further optimized the multi-level checkpointing by developing a method that further optimizes the selection of the levels. Furthermore, *Di et al.* [33] improved multi-level checkpointing by optimizing the checkpointing intervals for systems with uncertain number of cores.

Oliner et al. [32] introduce cooperative checkpointing technique which is a robust checkpointing algorithm comprised by a set of rules and policies that enable the checkpointing decision to be taken jointly by the application, compiler and the operating system (gatekeeper). In this approach, the developer inserts requests for checkpointing at optimal places in the code, the compiler further optimizes these checkpoint requests, and the gatekeeper makes the final call to either grant or deny the checkpoint. The gatekeeper considers a number of system runtime factors to grant/deny the checkpoint request, such as CPU load, Disk I/O, Network I/O, Job Scheduling Queue, Failure Event Prediction, and QoS guarantees. *Oliner et al.* [32] also show that cooperative checkpointing is simple and practical and can be applied on top of existing applications checkpointing mechanisms. A number of experiments were conducted and showed that cooperative checkpointing outperforms periodic checkpointing.

Jangjaimon and Tzeng [35] present an enhanced adaptive incremental checkpointing (EAIC) fault tolerance mechanism. EAIC aims to provide FT for multithreaded cloud applications that are hosted on multicore cloud infrastructures based on the futuristic RaaS (Resource-as-a-Service) model for cloud computing. An Adjusted Markov model (AMM) was constructed to cater for spot instances (SI), reserved instances (RI) and hardware failures. In RI, customers purchase reserved instances where the resources (such as CPU, IO or memory) are provisioned beforehand. For SI, a customer can bid for unused resources and often at a much lower price than for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

Category	Advantages	Disadvantages
Reactive methods (**RAMs**)	1) They can reduce the effect of failures on the system in the situation where failures occur less frequently. 2) They are suitable for long running tasks. The checkpoiting/restarting techniques are more suitable for long running tasks.	Not ideal for short running or real-time tasks. The recovery from a fault will add substantial delays to the overall response times.
Proactive methods (**PRMs**)	1) They can maximize the availability of the system provided the prediction of fault works accurately. The possibility of system interruptions is reduced due to the replacement of components that are predicted to fail with working ones. 2) Suitable for real-time applications since the effect of failures becomes transparent to the system by handling faults before they occur. 3) They limit exposure of the system to failures by avoiding recovery from faults.	In the event of incorrect prediction of failures (false predictions), system might execute the fault-tolerance procedures unnecessarily. PRMs are based on statistical models which are executed offline, this might not necessarily capture the changing system dynamics.
Resilient methods (**RSMs**)	Also known as Adaptive Fault Tolerance (AFT) - the system continuously learns and adapts to fault tolerance situations. As a result, the system only make use of resources sufficient to deal with the fault situation and therefore reducing the resource requirement for handling faults.	Models and learning structures must be updated continuously to cater for changing dynamics of a cloud system. This might become resource intensive especially when the dynamics are changing rapidly. Agent need time to learn the changing dynamics of a system, the learning process might be impacted if the environment is changing quickly.

TABLE III: Advantages and disadvantages of FT methods in the clouds

RI. The results showed a significant reduction in application runtime and cost both attributed to the use of multilevel checkpointing. This observation was made when using RI and SI.

Zhao et al. [36] propose a novel approach which determines how peer-to-peer checkpointing can be used to provide elastic and joint reliability optimization in cloud computing. In general, this work [36] leverages on checkpointing technique with cloud utilities to jointly maximize reliability under resource constraints in the data center. The main focus is on peer-to-peer checkpointing which is used to improve reliability under network resource constraints. In a normal scenario, VM images are sent to a central storage server which can result in network congestion due to high bandwidth usage. To alleviate this problem, peer-to-peer checkpointing is a distributed approach where cloud operators choose where to route checkpoints among peers with sufficient bandwidth. Simulation results showed that this approach significantly improved reliability when compared to both random peer-to-peer and centralized checkpointing.

Amoon [37] describes an adaptive fault-tolerance framework for cloud computing based on checkpointing and replication. The framework is adaptive in a sense that it is capable to select the optimal fault-tolerance approach to use for the customer's task. Furthermore, the framework proposes a replication algorithm that adaptively determines the number of replicas required for an application. In this way, replication is only applied to VMs that have a large performance impact on the cloud if they fail. Checkpointing is also adaptive in a way that the length of checkpointing interval is adaptively determined based on the failure probability of VMs.

2) Replication

The replication technique works by duplicating some system components which are then deployed simultaneously across different resources. This technique aims to make the system robust, to increase availability and to guarantee the execution of jobs [9, 26, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47]

Bodík et al. [9] present a novel algorithm that can be used for cloud migration. A challenge for cloud migration is to find an optimal deployment model which caters for fault tolerance as well as bandwidth costs. The algorithm also determines the optimal way of replicating system components into the cloud and achieve optimal bandwidth and fault tolerance. One of the primary challenges of deploying systems into the cloud is server availability. This is directly linked to the bandwidth available. Data centers are built with local redundancy, which provides for local servers being taken off line while they are being maintained or repaired. A challenge comes when the entire data center goes off line, due to a disaster or maintenance of the network equipment. This often results in a huge number of servers being taken off line. The available bandwidth is linked to the deployment architecture of a cloud system. A cloud system deployed into one data center is prone to experience various network failures, such as server unavailability and network congestion. The data center becomes a single-point-of-failure in the event of total loss of network connectivity. Increasing the number of data centers where a system is deployed has a direct increase on the bandwidth usage but it significantly improves on fault tolerance. A challenge therefore exists in finding a good balance between providing for high fault tolerance and reducing bandwidth usage. This can be addressed by the algorithms described in [9].

Balasubramanian and Garg [38] describe a fault management solution in distributed systems based on fused data structures designed to handle both data crashes and Byzantine faults. Fused data structures are designed in such a manner that recovery on primary data structures can be done with very limited number of replications. The main advantage of fused data structures is the savings in storage space required for storing replicas of data structures. This technique is mainly suited for distributed storage for solutions based on data structures such as queues, stacks, vectors, binary search trees, hash maps and hash tables. Fused data structures also offer a huge saving on computing resources required to recover from faults.

Cully et al. [39] present Remus which is primarily based on checkpointing and replication. The objective of Remus is to implement a transparent fault tolerance technique that does not require any changes to existing cloud applications. In Remus, the virtual machines hosting an application are paired into a primary and secondary VMs. Replication in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

Remus is done asynchronously. Various techniques are then employed to ensure asynchronous state replication between primary and secondary. The output of the primary VM is buffered and asynchronously replicated to the secondary VM. The primary VM resumes execution as soon as its state has been checkpointed and does not wait for acknowledgement from the the secondary VM. Remus incorporates a simple disk buffering technique to keep the primary and secondary VMs' disks synchronized. The disk writes issued on the primary VM are committed its local disk immediately, they are simultaneously transmitted to a buffer on the secondary VM. The secondary VM commits to its local disk after checkpoint.

Castro and Liskov [42] describe Byzantine fault tolerance (BFT) protocols. BFT protocols mainly aim to solve reliability and ensuring high availability of systems. Most BFT systems are too expensive for practical implementation, as such, to our knowledge so far, there has been no reported commercial data centers implementing BFT techniques. A BFT solution to provide asynchronous, distributed, client-server system requires at least $(3f + 1)$ number of replicas where one is a primary and the rest are backups, where f is the minimum number of faults that can be tolerated at any given point in time. BFT solutions have high resource consumption which can be attributed to the way in which they handle faults. BFT solutions rely on the server state machine replication (SMR), where each replica executes the same request in the same order. The replicas make use of the Byzantine agreement protocol in order to agree on the ordering of a given set of requests. After the order has been agreed, execution begins, the majority voting scheme is then used to choose the correct answer to send back to the client. In this manner, a faulty server can also be detected during this voting phase.

Zheng et al. [46] present a configurable fault tolerance approach based on component ranking (known as FTCloud). FTCloud is made of two algorithms; the first algorithm uses the component execution structures and monitors the execution frequencies in order to construct significant component ranking. The second algorithm uses these significant component rankings together with fault tolerance requirements input from the system designers to identify significant components of a cloud application. Once the component ranking is done, the algorithm automatically determines an optimal fault tolerance strategy for the significant cloud components. The most important observation is that by tolerating faults of a small part of the most significant components, the reliability of cloud application is improved significantly. The optimal fault tolerance strategies used for the significant components are based on redundancy and replication.

Jhawar et al. [47] introduce an innovative, system-level and modular solution for creating fault-tolerance in the cloud. The solution shades away the FT implementation details from the application developers. As such, it creates a service layer of which developers can request FT as a service. The user can specify and apply the desired level of FT without any knowledge of the underlying techniques used for implementing fault tolerance. The solution assumes client applications are deployed on virtual machines and therefore limits the granularity of FT to the VM instance. To be specific, the solution makes use of redundancy and replication to implement FT, multiple copies of VMs are created and readily available to take over when faults occurs.

3) Retry

The retry technique works by simply retrying a failed request on the same resource multiple times [48, 49].

Ramalingam and Vaswani [49] proposes a solution which relies on the retry technique to address failures due to process or communication failures in cloud systems. In such environments, the failed requests are simply retried. The retry mechanism relies on the *indepotence* of the system. Indepotence is described as a criterion such that the results obtained from an application in the presence of duplicate requests and failures are exactly the same as the result obtained from such an application in the absence of duplicate requests and failures. Indepotence is formalized by a language called λ_{FAIL} which is influenced by cloud platforms. In essence, the language λ_{FAIL} formalizes process failures, duplicate requests, data and transactions. The ultimate goal of λ_{FAIL} is to guarantee indepotent cloud services. Such a system is decentralized and there is no need for distributed coordination. As a result, this yields a complete decentralized implementation of fault tolerance which handles multiple retries of same requests and process failures.

Wang et al. [48] analyzes the performance impact of the retry fault tolerance technique to cloud services. This is achieved by constructing a mathematical model of the processing time of a cloud service. The model is followed by a probability distribution of the processing time to analyze the effect of retrying jobs in the presence of failures. A reflection on the Quality of Service (QoS) is also done by calculating the percentage of requests that can be served successfully under the defined threshold values. The performance of cloud systems using retry technique was compared to the performance of the cloud systems using the checkpointing techniques for FT. The results showed that depending on the recovery rate of checkpointing approach, the performance of retry techniques can be better or worse than the checkpointing techniques.

4) Task resubmission

In *task resubmission*, when a failed task is detected, it is resubmitted to the same or different resource for execution [44].

Plankensteiner et al. [44] described the resubmission impact (RI) heuristic and used it to improve fault tolerance approaches that are based on resubmission and replication techniques. Such approaches are normally used to implement fault tolerance in workflow systems that are distributed in the cloud. The improvement is based on the introduction of the new algorithm called *resubmission impact*. Resubmission and replication are fundamental techniques that are widely used for fault tolerance in distributed systems. Resubmission operates by re-executing a single task to the failed resource or to a new one which can significantly increase the task completion time. Replication executes the several copies of the same task to multiple resources concurrently and suffers from the drawback of high resource usage. To find a balance between resubmission and replication the algorithm computes the RI heuristic which is used to describe the impact of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

resubmitting a task to the overall execution of the workflow. RI is defined for each workflow and it is used to deduce the number of replications generated. Experimental results show that RI significantly reduce resource consumption by over 42% compared to a conservative approach. In addition to this, RI does not negatively impact on the task completion rate and overall workflow performance.

5) Custom exception handling

Custom exception handling includes approaches where software developers insert code into the application so that it can handle specific faults during runtime [50].

Liu et al. [50] proposed a framework used for solving fault-tolerance for transactional web services (FACTS) that are orchestrated through a Business Process Execution Language (BPEL) engine implementation. Web services are mainly used to develop modern business applications. FACTS is an aggregation of six components, i.e. EXTRA, WS-BPEL designer, specification module, verification module, implementation module, and planning module. The EXTRA component provides a set of high-level exception handling strategies that operate on the standard WS-BPEL built-in exception and transaction handling facilities. Fault tolerance requirements for composite web services normally emanate from the business requirements. The service designers make use of ECA (Event-Condition-Action) rules to define the logic for handling faults. ECA rules are in turn based on the exception strategies defined in the EXTRA module. The verification module is used to validate the fault-handling logic. It does this by evaluating the principles that the ECA rules must comply to.

6) Rescue workflow

Rescue workflow is a technique aimed as solving fault tolerance for workflow-based systems. The workflow is allowed to continue even if the task fails until it becomes impossible to continue without attending to the failed task [51, 52]. Readers who are further interested in fault tolerance mechanisms for workflow based systems can consult [51].

Hernandez and Cole [52] presents the rescue directed acyclic graph (Rescue DAG), a reliable workflow DAG scheduling mechanism based in rewinding and migration. This mechanism depends on two key components, i.e. DAG Manager (DAGMan) and Rescue DAG. The DAGMan is the meta-scheduler and it is responsible for managing the entire workflow, this includes scheduling workflow onto computing resources. DAGMan also contains a fault tolerance mechanism. The mechanism performs resubmission of failed portions of a DAG workflow. When a task of a DAG fails, the remaining tasks in the DAG are continued until it is not possible to continue due to task dependencies in the DAG workflow. In this event, the DAGMan outputs a special file called the Rescue DAG which contains sufficient details about successful task and unsuccessful tasks. The Rescue DAG is then used to restore the workflow. The failed tasks are resubmitted. The successful tasks are not re-executed and hence a saving on computing resources and time.

7) Load balancing

Load balancing is key to the fault tolerance and operation of cloud systems. Many servers can crash as a result of exhausted computing resources (CPU or RAM) after being overwhelmed by client requests. To reduce such failures, cloud systems must implement load balancing to serve as the first line of load protection [53, 54, 55, 56]. The traditional centralized load balancing mechanisms are not suitable for cloud systems. Cloud systems are highly scalable and they are built on distributed servers which could be hosted in multiple data centers (depending on the architecture of the cloud system). Due to the scale and complexity of such systems, the centralized assignment of computing requests to the servers is rendered infeasible. Honeybee Foraging Behavior [55], Biased Random Sampling [56] and Active Clustering [53] are amongst mostly used load balancing algorithms for cloud systems. Distributed load balancing algorithms normally have built-in mechanisms which enable them to coordinate in handling the load fairly. A comparative study of these algorithms has been presented in [54].

8) N-Version and recovery block

N-version is a multi-version programming model. Multiple programs that are functionally equivalent are developed by separate teams working from the same set of requirement specifications. The teams work independently and do not communicate at all. The idea of N-version is based on the fact that independently developed programs greatly reduce the probability of similar faults in two or more versions. Recovery Block (RB) utilizes different representations of input data to provide the tolerance of design faults [57, 58, 59, 60, 61].

N-Version has recently found its way into the cloud for security [58] and malicious attacks (anti-virus [57]) applications. Some general background on the relationship between N-Version and fault tolerance is presented in [60]. In the following paragraphs we discuss some research work where the N-Version technique has been applied as part of fault tolerance in a cloud scenario.

Peng et al. [59] reviewed the Enhanced N-Version Programming (ENVP) and Extended Recovery Block (ERB) techniques. ENVP and ERB can be jointly used to improve the reliability of Web Services (WSs) and therefore they have a direct contribution to improving fault tolerance in cloud systems. Both ENVP and ERB are extensions of their original counterparts N-Version (NVP) and Recovery Block (RB) fault tolerance mechanisms respectively. These techniques have been adapted to provide support for WS-based fault tolerant systems. NVP provides fault tolerance by utilizing multiple functionally equivalent software components (or versions). On the other hand, RB uses different representations of input data to tolerate software design faults.

The logic for ENVP and ERB is implemented in the middleware which is responsible for handling interactions between the service users (clients) and the service providers (services). Both NVP and RB have been extended by the introduction of the acceptance test (AT) component into the middleware. The AT component is used to evaluate the correctness of the output from WS's before they are passed to the decision manager (DM). Experimental results showed that adding more AT components improves the overall reliability of the system.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

B. Proactive methods (PRMs)

Proactive methods (PRMs) constantly monitor the system and conduct fault predictions so that the effects of faults are prevented well before they occur. In *monitoring*, the system constantly executes failure *prediction* algorithms to evaluate the status of the system so that the necessary actions can be taken to prevent failures. For cloud systems that are running in virtualized environment, such fault management techniques rely more on the migration, pause/unpause functionality provided by the virtual platform. Based on the current literature, key techniques used for proactive fault-tolerance include *Software Rejuvenation*, *Self-healing*, *Preemptive Migration*, *Monitoring*, and *SGuard*. This section provides a review of the selected papers on PRMs.

1) Software rejuvenation

Software Rejuvenation (SR) is designed for periodic restarts [62, 63, 64], it basically involves gracefully terminating the system and starting it again. SR is complemented by two other techniques: 1) error counting; and 2) N-Version programming. Error counting is also known as "Count the Black Sheep". It is a technique of counting errors as they occur, a record of these error counts is kept so that it can be used for the escalation and speed-up the recovery process.

2) Self-healing

The self-healing technique is a feature of a system that allows it to automatically detect, diagnose and repair software and hardware faults. Such systems are made of multiple components that are deployed on multiple VMs [65, 66, 67, 68].

3) Preemptive migration

Preemptive migration is a prominent approach for virtualized environments. It provides mechanisms for migrating execution of programs from one machine to another in real time. This technique prevents system components that are about to fail from impacting the performance of the system. This is achieved through monitoring and by moving components away from nodes that are about to fail to run on more stable nodes [3, 13, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78]. Applying preemptive migration for proactive FT [69] presents an architecture of a proactive fault tolerance approach which relies on preemptive migration.

Engelmann et al. [69] defines an architecture which servers as a foundation for proactive FT. The architecture is based on preemitive migration of virtual machines (VMs) and therefore it is applicable to cloud computing. Furthermore, a classification of implementation options is provided. The classifications are defined by the monitoring strategy employed by the implementation. The core of this architecture is a feedback-loop control mechanism which forms part of system health monitoring. The system and its applications are monitored and preventative action is taken to migrate application components away from nodes that are predicted to fail to more healthy nodes. Some challenges for monitoring the health of systems are identified. One of the key challenge is that each solution is currently using its own set of metrics for measuring and evaluating the health and interfaces between system components. Standard metrics and interfaces are required.

Nagarajan et al. [70] contributes an automatic and transparent mechanism for proactive FT for message passing interface (MPI) applications. The mechanism combines virtualization techniques with health monitoring and preemptive migration to implement a proactive FT solution. The XEN virtualization environment was used. The solution exploits the live migration feature of XEN which allows guest OS to be relocated together with its running tasks to another node. The migration is triggered when the health of a node is detected as deteriorating. The preemptive migration mechanism works so well in such a way that migration overheads are eliminated. The MPI tasks continue to execute while it is in the process of being migrated.

Liu et al. [3] proposed a proactive coordinated FT (PCFT) solution aimed at providing fault tolerance to parallel cloud systems. A VM coordination mechanism is used to predict a deteriorating physical machine (PM). The VMS on a deteriorating PM are migrated to optimal target PMs. Prediction of a deteriorating machine is based on monitoring of the CPU temperature. The complexity of the algorithm is in finding an optimal target PM which must ensure efficiency, effectiveness, and scalability requirements.

4) Prediction

Prediction forms the core of proactive fault tolerance algorithms. Faults are predicted in advance so as to give the cloud system a chance to take corrective steps to avoid or reduce the impact of the fault [13, 17, 79, 80, 81, 82, 83, 84]. *Salfner et al.* [83] provides a through survey of prediction methods. A simulation framework which can be used to evaluate various FT approaches and policies is presented by *Tikotekar et al.* [84]. This framework puts a considerable emphasis on proactive FT using a prediction component.

Vallee et al. [13] describes a generic framework for proactive fault tolerance. This framework is based on fault prediction and the main objective is to avoid failure. When failures are detected, the system uses the fault management features of the underlying virtualization platform such as pause/unpause or migrate a process or entire virtual machine node. The fault predictor uses the local information on a computing node to predict a probability of a fault occurring currently, the fault predictor periodically analyses the local system logs and generates an alarm event if an abnormal behavior is detected. Two experiments were conducted as part of proving the effectiveness of this framework on computational overheads when doing migrations. The first experiment was done using the XEN virtualization platform and the impact of virtual machine footprint on the migration cost was measured. The impact was measured a a function of virtual machine memory to total migration time (in seconds). Secondly, a similar experiment was conducted using a simulator designed for evaluating fault tolerance policies [84]. The simulation results were found to be consistent with the physical experimental result using XEN which led to a conclusion such that this framework reduces computational overheads.

Pinto et al. [17] presents an enhancement to fault tolerance in Hadoop clusters by incorporating prediction. Prediction is implemented using Support Vector Machines (SVMs) models. SVMs are supervised learning models with associated learning algorithms which analyze the data used for classification

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

and regression analysis. The SVM classifier is hosted on a monitoring PC to perform smart predictions for system faults. The standard Hadoop architecture includes one level of fault tolerance where a job is rescheduled from faulty nodes to other nodes in the network. This results in inefficiencies as it might require reprocessing of sub tasks that were already completed. The SVM prediction models are used to predict faults much earlier so that decisions to reschedule a job are done much earlier. Furthermore, a reinforcement learning module is incorporated to eliminate false positives which significantly enhance the fault tolerance of the clusters.

Costa et al. [79] describes a proactive fault tolerance solution based on prediction for avoiding memory errors efficiently. This solution follows from an observation such that checkpointing/restarting might not be efficient for handling memory faults in high performance computing which executes in petascale. This approach is embedded into the OS. It works by exposing correctable error information to the OS, the pages are migrated and faulty memory is taken offline to avoid crashing applications. An analysis of memory error patterns is done and correctable error patterns are used to predict memory that can possibly fail. A prototype was implemented on Linux running on the IBM's Blue Gene/Q (BG/Q) system which is an HPC system.

5) Monitoring (Feedback loop)

Monitoring is mainly used to supplement other proactive algorithms. It is used to monitor a set of state variables on a running application. The state variables are filtered and supplied to a policy manager through the feedback loop mechanism [14, 85].

Egwutuoha et al. [14] describes a proactive technique which relies on monitoring to provide FT to HPC in the cloud. The framework for this FT technique was initially proposed by *Egwutuoha et al.* [85]. According to this study, more than 50% of failures on HPC systems are caused by processors, hard drive, integrated circuit sockets and memory. The benefit of proactive fault tolerance in HPC computing compared to reactive approaches is highlighted. Unlike reactive fault tolerance approaches, which are mainly based on checkpoints and restarts, proactive fault tolerance avoids restarts from checkpoints and therefore reduce operational costs as well as energy consumption. *Egwutuoha et al.* [14] focuses mainly on MPI applications. MPI applications are actually applications running in parallel on different CPUs or VMs and they communicate by exchanging data through messaging, e.g. GROMACS system [89]. *Egwutuoha et al.* [14] employs avoidance approach to tolerate faults. This is achieved by using a combination of system log and health monitoring facilities. The system log provides information regarding reliability, availability and serviceability whereas health monitoring prides the state of the hardware and software. The architecture of the proactive FT solution presented in [14] consists of four types of modules: 1) *Node monitoring module with an lm-sensor*; 2) *A failure predictor*; 3) *A proactive fault tolerance policy module*; and 4) *The controller module*. [14] also presents a skeleton of the fault-tolerance algorithm and some quantitative analysis of the results. The results show a significant cost saving by implementing this FT model. The saving is mainly as a result of saving on operational costs as a result of relinquishing the failed node immediately.

Park et al. [90] describes a monitoring technique that can be applied as part of fault-tolerance in mobile cloud computing. Although *Park et al.* [90] does not describe a full fault tolerance technique, it provides a monitoring technique which can be quite useful in implementing FT in mobile cloud computing. Mobile cloud computing is described as a combination of mobile computing and cloud computing. In the context of mobile cloud computing, cloud computing is provided through mobile devices. Legacy problems on mobile devices seem to be overcome, such problems include low-life batteries and low CPU performance. The paper identified two categories of using mobile devices in mobile cloud computing, i.e. mobile device as interface and mobile device as resource. Most work was previously conducted on using mobile device as interface and recent trend is moving toward using mobile device as resource for hosting cloud services. Using mobile device as a resource is faced with most problems associated with mobility. These include, volatility due to unstable wireless connections, limitations on power supply, low network bandwidth and handover problems due to frequent location changes. As a result, resource monitoring is a key in order to implement a reliable resource scheduling and fault tolerance technique in mobile cloud computing. A resource monitoring technique required to collect and analyze dynamic information on the sate of each participating resource and to ensure stability. *Park et al.* [90] proposes a monitoring technique which is based on Markov chains aimed at monitoring and analyzing resource states. The main aim is to solve the impact of the volatility of mobile devices to the fault tolerance problem.

6) SGuard

SGuard is a rollback and recovery based technique used for real-time video streaming. It is much less turbulent to the video streaming [18].

Kwon et al. [18] presents SGuard an example of a resilient approach to fault tolerance formed by an amalgamation of reactive fault techniques, i.e. checkpointing, rollback, recovery and replication. SGuard is a relatively new technique applied to handle faults in distributed stream processing engines (SPEs) that are deployed in multiple clusters. This deployment model is similar to that of cloud based services. This approach uses the rollback/recovery techniques for fault tolerance. As the system is running, SGuard performs checkpoints asynchronously while the streaming service is running. The failed servers are rolled back and restored based on the most recent checkpoint. The checkpointing, rollback and recovery of failed servers occurs asynchronously without any service disruption. The checkpointed states are saved on distributed file systems (DFS) such as GFS, HDFS or Amazon EC2. SGuard is capable of handling both software failures and hardware crashes. To mask failure, SGuard further employs the replication technique. The state of SPEs is replicated on multiple servers which are classified as primary or secondary. As a result, SGuard provides a less disruptive fault-tolerance solution for real-time video streaming.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

C. Resilient methods (RSMs)

Resilient methods (RSMs) enable a system to continue serving client requests in the presence of faults and to recover quickly within an acceptable time period. The fault could be a result of the equipment failure, power outage or disruption. Generally, resilient systems are made up of components which are responsible for implementing functionalities of the ability to continue responding to the clients in the presence of failures, monitoring of system state, the ability to learn and adapt to the system. *Colman-Meixner et al.* [91] provide a comprehensive survey of resilient techniques applied at different layers of the cloud architecture. The system monitoring component is crucial for keeping an eye on the system state and to detect and predict faults that are about to happen in the near future. The objective of a resilient system is to minimize the overall downtime of the system by maximizing the availability of the system. The learning component is responsible for optimizing the recovery from faults, it does this by using the environment parameters and use them to improve the handling of the fault next time it occurs. That way, the system is able to learn from the environment and adapt its fault tolerance mechanisms.

The adapting of fault tolerance mechanisms normally involves deploying optimal resources to let the system avoid complete outage. System can dynamically add or remove capacity on time before or after the fault is experienced. In essence, resilient methods are formed by combining the techniques used for RAMs and PRMs with an addition of the ability learn through interaction with the environment and to adapt the fault tolerance [19, 20, 21, 22, 23, 24, 25]. This section provides a review of the selected papers on RSMs.

1) Machine learning approaches

Machine learning (ML) brings the smart way of doing fault tolerance. Cloud systems are empowered to learn by interacting with their environment and adapt their fault handling strategies accordingly. Reinforcement Learning appears to be the most common technique used in the FT domain [19, 20, 21, 22, 23, 24, 25, 39, 86]. Most papers reviewed in this section might not directly link to cloud computing. However, we seek to identify sources of information where Machine Learning, especially reinforcement learning has been used to implement or improve the fault-tolerance capability of a system. Such ideas can easily be extended for application into cloud environments.

Tung et al. [86] presents a highly resilient fault tolerance solution for clouds based on priority queues and dynamic routing. The solution is further supported by cloud agility, i.e. the ability to provision or deprovision additional could resources dynamically on the fly. Using the triage approach, cloud service requests are split into high and low priority requests and they are mapped onto high and low priority queues respectively. Distributed dynamic queuing is used to offload requests on local and remote cloud nodes. The queuing technology (such as Rabbit MQ) adds the capability of handing over requests to be processed on remotely hosted could servers. The dynamic routing monitors the load on the priority queues, when the load on the priority queues buildup, it signals the provisioning of additional services on remote sites. Service requests on the lower priority queues (those that can tolerate high latencies) are handed over to be processed on the secondary servers. This solution includes a data replication layer on the backend which synchronizes the data between the primary and secondary services so that they can run independently.

Xu et al. [20] proposed the unified reinforcement learning (URL) solution for FT. URL is an ML technique used to auto-configure VMs and appliances for cloud computing. The VMs are the virtual machines hosting various components of a cloud system. The appliances can be any VM-based software packages such as Apache or Tomcat web appliances. Both VMs and appliances have a large set of configuration values. The process of configuring these values for optimal performance and availability is error prone. In the paper [20], two reinforcement learning agents are described, i.e. VM-Agent and App-Agent. The VM-Agent is used to reconfigure the VMs and the App-Agent reconfigures the appliances. The behavior of each agent is modeled using the Markov Decision Process (MDP). The state of each agent is described as a vector of configuration parameters of the VM or appliance. As the workload demands of the system varies over a time period, the objective of URL is to find good configurations yielding optimal performance for the given workload.

Wu et al. [21] suggested ordinal sharing learning (OSL) for enhancing resiliency and FT. OSL is a robust multi-agent reinforcement learning (MARL) method for job scheduling. Although initially designed for grid computing, the ideas in OSL can be applied to cloud computing as well. OSL is highly scalable. OSL is made up of lightweight and linear utility table that is iteratively exchanged between the agents. The utility table is used to track the efficiency of system resources that are processing the scheduled jobs. The OSL requires very little network communication bandwidth, the agents only exchange the utility table which serves as input to the learning algorithm at each agent. A Scheduler Agent is made up of two main components, i.e. Actor and Learner. The Actor receives the jobs to be scheduled, it buffers the jobs (Job Buffer) in a queue and keeps track of jobs that are currently running (Submitted Job List). The Leaner receives the shared utility table, it uses the information in the submitted job list to learn new scores for the resources that are handling the current jobs. The scores are updated and the utility table is passed to the adjacent node in the grid. This way ensures that the best performing and most reliable resources handle jobs and therefore enhancing the resiliency and fault-tolerance of the system.

Farivar and Ahmadabadi [19] propose two strategies that can be used to design a robust and adaptive fault tolerant control (FTC) system. Although FTC is not directly related to cloud systems, the ideas and the manner in which machine learning (especially reinforcement learning) and neural networks (NN) have been applied can be borrowed and implemented for cloud systems data centers. A similar approach of using NN for fault tolerance control in actuator based systems has been considered in [22]. The faults are generated by some actuators and sensors which can be installed in a typical cloud data center to monitor various aspects, such as intrusion and temperatures. The sensors can be extended to include software

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

defined sensors that can to monitor various software defined metrics such as traffic volumes or system load. Two strategies for FTC were defined. The first strategy involves an intelligent observer designed to monitor an unknown number of nonlinear systems. The second strategy is based on reinforcement learning and it combines some unknown non-linear faulty system and non-linear control theory to guarantee the stability and robustness of the system. Such a non-linear system could be modeled as the server infrastructure responsible for hosting cloud systems. Simulations were done and results confirm that the above FTC strategies perform well. Furthermore, for actuator faults, reinforcement learning was found to perform better than neural networks. Similarly, neural networks performed better than reinforcement learning on sensor based faults.

Forster and Murphy [23] presents FROMS (Feedback ROuting to Multiple Sinks), a machine learning-based multicast routing paradigm for wireless sensor networks (WSNs) based on reinforcement learning. This was achieved by modeling multicast routing in WSNs as a reinforcement learning problem. Although WSNs are not viewed as directly related to cloud computing, there is a view that end-user devices in poorly connected networks could form a dynamic WSN and use it to access cloud services. In such scenario, a fault tolerance solution for WSN can contribute to solving FT for cloud systems by making WSNs more robust. FROMS is basically a routing protocol for WSNs. The advantages of FROMS include flexibility in optimizing routing over a different sever conditions such as route length, battery levels, recovery after failures and support for sink mobility. These claims are supported as part of the experimental results obtained in [23].

Wang and Usher [24] describe an application of RL to agent-based computing. RL is generally used to empower an autonomous agent to learn to select proper actions for achieving its goals through interacting with its environment. The selection of actions depends on the problem being solved. Although not directly linked to fault tolerance, *Wang and Usher* [24] applied RL on computing agents in the manufacturing systems. A well-known RL algorithm, Q-Learning, is used to enable a machine agent to learn commonly accepted dispatching rules and this relies on the best dispatching rules being previously defined. This application of RL can be of use for fault tolerance in cloud computing. Since cloud computing includes many different types of computing agents accessing services over the Internet. There is a need to solve the connectivity problem, especially on mobile networks. The Q-Learning algorithm can be applied to enable the agent to select the most reliable connection to the Internet at all times, based on the list of available network access points.

Chen and Marculescu [25] presents an Online Distributed Reinforcement Learning (OD-RL) algorithm for improving the performance of many-core systems under power constraints. The OD-RL algorithm is based on Dynamic Voltage Frequency Scaling (DVFS) techniques for saving power. Most data centers hosting cloud services are deployed in server infrastructure based on many-cores (CPU) systems, therefore OD-RL is applicable to the clouds. Reinforcement learning is used to learn the optimal policy to control the voltage/frequency (VF) at CPU levels. The maximize-the-max method which is used to manage power budget levels at a more coarser grain global level is defined. Our interest is more on the application of RL at the finer grain CPU level. Experimental results showed that OD-RL resulted in significant power savings, improvements on throughput as well as energy efficiency.

2) Fault induction

Limoncelli [87] describes the term *antifragility* and the application of failure induction methods to FT at large corporations such as Google and Amazon. Amazon began using the fault induction methods back in the early 2000s. This was done through a program called GameDay. GameDay is a program designed to increase the resilience by purposely subjecting major failures into systems deliberately and at given time with the aim of discovering flaws and dependencies between systems. A GameDay test is similar to a fire drill in an organization. The GameDay operation does not only focus on computer systems, it also includes testing on software and people (implying business processes) with the aim of preparing them for a response to an actual disaster event. A GameDay exercise simulates a real disaster and therefore the participants can include staff members at various levels of an organization [87]. The GameDay phenomenon has been actively used at large organizations such as Google and Amazon. GameDay tests can be repeated. A GameDay exercise is marked successful only if everything works fine when the test is repeated. Part of the outcome of this technique is to enable organizations to learn from failures. Significant results has been achieved by using the GameDay methods.

D. Strengths and Weaknesses

In this section we review the strengths and weaknesses of fault tolerance methods. The strengths and weaknesses of traditional methods has been well studied in the literature, these include [26], [12], and [31]. As such, in this review, we will present the strengths and weaknesses of Machine Learning specific methods since this is an emerging field.

The growing use of cloud computing is resulting in exponential growth of systems complexity and scale. This will eventually lead to a state when traditional methods for fault tolerance are no longer efficient and feasible. Alternative methods to deal with the fault tolerance issues such as resource management, security and energy efficiency in the cloud are therefore required. The papers reviewed in this study have evaluated various machine learning algorithms which contribute differently into cloud fault tolerance.

Machine Learning (ML) provides the tools and algorithms that have the capability of handling huge sets of data and continuously learn and adapt the system. ML is comprised of three main categories, i.e. supervised, unsupervised and reinforced learning (RL). RL is more suitable for implementing control optimization types of solutions. As a result, RL is preferred for implementing fault tolerance [19, 23, 92].

In general, RL algorithms are grouped into two main classes, i.e. model-based and model-free approaches [93]. The model-based approach constructs a model of the world using the experience from the agent's interaction with the environment (cloud). The model is used to learn a value function.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

The model-free approach estimates the value function directly from its interactions with the environment. Each class has different strengths and weaknesses. The key strength of model-based approach is such that, in most cases, they find a good value function with less interactions with the environment and therefore are usually seen as yielding better performance, this is known as *data efficiency* [93]. However, this comes at a cost, model-based approach usually demand higher amount of computing resources. The main strength of model-free approaches is that they demand less computational resources and therefore they can support representations much larger than model-based approaches, this is known as *computational efficiency*. Another strength of model-free algorithms is such that they are scalable, they grow linearly with the number of features that are representing the environment [93].

The ultimate objective of RL is to learn an optimal policy which is used to govern the agent's behavior. An RL system is formed by a number of agents that learn through some experience involving trial and error from interactions with the environment. The RL agents continuously adapt to the environment by maximizing the rewards earned from an action-value function. The agents learn the best action to take from every system state. RL is applicable to systems with extremely large number of states and complex structures and therefore suitable to cloud systems. However, a weakness of such systems is that a huge amount of computing resources is required especially storage and memory.

Some well-known Machine Learning tools have been used with RL algorithms including Q-Learning, Artificial Neural Networks (ANNs), Naive Bayes, Random Forests and Deep Learning [15]. Next we look at the strengths and weaknesses of using each approach for fault tolerance solutions.

Q-Learning is a model-free approach which is mainly used to find an optimal policy by learning optimal Q-factors for each sate-action transition. It is mainly applied to any Markov Decision Process with finite states [20]. The key strength of Q-Learning is that it is model-free and quite straight forward to implement. The main weakness of Q-Learning is the amount of storage required to keep track of each Q-factor, especially when the number of state-action pairs grows too large.

Artificial Neural Networks (ANNs) can be used to complement Q-Learning. ANNs are applied when the number of state-value pairs are too large and therefore not efficient to be stored in Q-factor lists. As such, all Q-factors for a given action are stored in one network. However, training ANNs to represent Q-factors can be complex and time consuming.

Support Vector Machines (SVMs) are an example of a data mining tool that have recently been applied in RL especially when classification or regression is required [94]. The strengths of SVM include a high level of prediction accuracy and they work even when the training examples contain errors. On the weaknesses side, SVMs require long training times, the learned function is hard to understand since it is represented in weights.

Naive Bayes is a Bayes Theorem based classifier which is model-based and can be used with RL. It is useful in situations where the model is not fully known and there are uncertainties. The key strengths of Naive Bayes is that it is seen as fast, robust and can work with incomplete models. Naive Bayes can handle very large data sets and it outperforms many other sophisticated classifiers. A key weakness of Naive Bayes is its reliance on the Bayes theorem which assumes that all attributes (features) are independent. Such an assumption can lead to simply ignoring the impact of correlation of attributes in predictions.

Deep Reinforcement Learning is the application of Deep Neural Network with RL. This is particularly useful when the number of state-action pairs grows too large. In stead of storing the Q-factor for each state-action pairs, deep neural networks are used to store the Q-factors for each action, these networks are also referred to as Deep Q-Network (DQN) [95]. The key strength of this approach is the ability to handle large data and therefore enable RL to be extended to problem areas that were previously intractable [96]. Meanwhile, their main weakness relates to their training time.

Random Forests are tree based classification structures that have been used with RL. The strengths of tree-based algorithms include a significantly reduced classification error (highly accurate) and lower computing resource load [97]. They can handle fairly large feature variables and they are efficient on large databases. In most reported that Random Forests offer better convergence to the learning function approximation [97]. On the weaknesses side, Random Forests turn to suffer the problem of overfitting which is caused by the over growing trees. Cross-validation can be used to combat the problem of overfitting [98].

VI. Emerging Directions

Based on the papers that were reviewed as part of this study (see Table II), it is evident that there is currently a significant amount of fault-tolerance solutions that are still based primarily on reactive and proactive methods. In the context of cloud systems, the performance, flexibility and scalability of these methods still remain to be proven. For some of these methods, it is uncertain how they will work on systems deployed on distributed and heterogeneous cloud platforms. Current methods are not scalable and require some form of manual intervention for their smooth operation and configurations, as such, we propose that the future of fault tolerance in the cloud will be based on automation.

In trying to address some of these challenges, we are starting to see the emergence of agent-based cloud computing as a means for solving autonomous cloud problems [99]. As such, we are arguing that agent-based solutions, including reinforcement learning, are the emerging directions for autonomous fault tolerance in the cloud. An agent is a fully functional computing node which is capable of making decisions independently and interacting with other agents through cooperation, coordination, and negotiation [99]. We are witnessing the emergence of agent-based automation in various core aspects of the cloud such as resource allocation [100, 101], job/task scheduling [102, 103] and fault tolerance [104]. Furthermore, we will continue to see development and application of advanced machine learning techniques, such as deep learning, to support agent-based automation methods, hardware and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2816644, IEEE Transactions on Services Computing

infrastructure monitoring [105, 106, 107, 108, 109] and elastic caching [110] which have direct impacts on autonomous fault tolerance.

The following paragraphs provide a review of some of the recent papers where the agent-based computing and reinforcement learning paradigms have been investigated in relation to cloud computing and fault tolerance.

Sim [99] provides an introduction to the concept of agent-based cloud computing and explains how the agent-based computing paradigm is applied to the management of cloud computing infrastructures and resources. According to *Sim* [99], agent-based cloud computing involves the construction of service discovery, service negotiation, and service composition functions of the cloud. Service discovery is implemented by the Cloudle which is an agent-based search engine for cloud services. It is further shown that agent-based negotiation mechanisms can be used to implement service negotiation and cloud commerce. Furthermore, agent-based cooperative problem solving techniques are shown to solve the problem of automating cloud service composition. Experimental results show that using the agent-based automation approach to cloud computing, the agents achieved a high utilization and success rates in negotiating for cloud resources. Agents can also successfully compose cloud services by autonomously selecting services supported by the Cloudle mechanism.

Arabnejad et al. [101] describe another automation approach where RL can be applied to automate the problem of dynamic resource allocation in the clouds. The goal is to implement a cloud management solution which is self-adapting and auto-scaling depending on the fluctuation of system workloads. Reinforcement learning is used decide when to add or remove resources while still guaranteeing the agreed system SLAs. The fuzzy logic approach is used. Two dynamic learning strategies based on fuzzy logic are compared, i.e. Fuzzy SARSA learning (FSL) and Fuzzy Q-learning (FQL). Both approaches are capable of handling different workload patterns such as sudden and periodic workloads. Furthermore, FSL and FQL are capable of delivering resources on-demand, while reducing operating costs and avoiding SLA violations.

Dalília and Coutinho [104] put forward an autonomous and reinforcement learning based solution to the problem of balancing between replication and checkpointing in opportunistic grid systems. These opportunistic grid systems are defined as low-cost and large computational grids that are dynamically formed by the use of idle processing power of non-dedicated computing resources. Such non-dedicated resources can be geographically distributed across many different administrative domains and such resources join and leave the grid randomly. As such, there is a need to constantly monitor and detect and react to grid formation events in a timely manner. RL is used to automatically adjust the threshold used to switch between checkpointing and replication. Using RL, the switching decision is based on both the number and reliability of the computing nodes in the grid. Experimental results showed that this approach was able to learn an optimal threshold for switching between replication and checkpointing.

VII. Conclusions

This paper provided a review of various methods for implementing fault-tolerance in distributed or cloud systems. We have classified the fault-tolerance methods into three main categories: 1) reactive methods; 2) proactive methods; and 3) resilient methods. The reactive and proactive methods are mainly based on the traditional methods of fault tolerance, such as replication, checkpointing, retry, monitoring and pre-emptive migration.

Some of these methods have been used to implement fault-tolerance in cloud systems to some extent. For example, most data centers where the virtualization technology is used rely on preemptive migration for handling faults that are caused by server outages. These traditional methods have limitations. Firstly, they are based on fixed logic and handle faults in a specific way as defined by their implementation. As a result, they lack the capability of handling new faults that might arise in the future. Secondly, these implementations only consider the inherent system attributes when making decisions on handling faults. Very limited consideration is made to external or environmental attributes that might affect system performance (such as, temperature, power and weather).

Since the future of computing is moving towards the cloud, systems will be exposed to faults that will not be handled by traditional fault tolerance methods. Therefore, there is a need to develop systems that can learn and adapt through their interactions with the environments in which they run. Such systems will require the use of Machine Learning methods as part of their fault tolerance solution. As we have seen in this paper, machine learning had been used in creating fault tolerance solutions. However, machine learning has mainly been used as a sub-component of the overall fault tolerant solution. Some solutions have used machine learning mainly for prediction using a set of defined variables. In other applications, machine learning has been used in management of hardware faults. Such systems are in turn fixed and not dynamic enough to handle future and unknown faults.

There is a need to further extend the application of machine learning to fault tolerance by defining a reusable framework that can be used in cloud environments for handling faults. Such a framework will be known as Smart Cloud. The main component of the Smart Cloud will be any set of agents that are interconnected and learning how to handle faults by interacting with the environments in which they will be executing. As a direct consequence of this, such agents will be empowered to make connectivity decisions that will enable them make optimal use of energy as well.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this paper.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing,”

Communications of the ACM, vol. 53, no. 4, pp. 50–58, Apr 2010.

[2] A. Abid, M. T. Khemakhem, S. Marzouk, M. B. Jemaa, T. Monteil, and K. Drira, "Toward antifragile cloud computing infrastructures," *Procedia Computer Science*, vol. 32, no. 0, pp. 850 – 855, 2014.

[3] J. Liu, S. Wang, A. Zhou, S. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[4] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, and R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 902–913, Nov 2017.

[5] A. Zhou, S. Wang, Z. Zheng, C. H. Hsu, M. R. Lyu, and F. Yang, "On cloud service reliability enhancement with optimal resource usage," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 452–466, Oct 2016.

[6] S. Ananth and A. Saranya, "Reliability enhancement for cloud services - a survey," in *International Conference on Computer Communication and Informatics*, Jan 2016, pp. 1–7.

[7] J. W. Lin, C. H. Chen, and J. M. Chang, "Qos-aware data replication for data-intensive applications in cloud computing systems," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 101–115, Jan 2013.

[8] Z. Qiu and J. F. Prez, "Enhancing reliability and response times via replication in computing clusters," in *IEEE International Conference on Computer Communications*, April 2015, pp. 1355–1363.

[9] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012, pp. 431–442.

[10] J. Liu, S. Wang, A. Zhou, F. Yang, and R. Buy, "Availability-aware virtual cluster allocation in bandwidth-constrained datacenters," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[11] W. Qiu, Z. Zheng, X. Wang, X. Yang, and M. R. Lyu, "Reliability-based design optimization for cloud migration," *IEEE Transactions on Services Computing*, vol. 7, no. 2, pp. 223–236, April 2014.

[12] P. K. Patra, H. Singh, and G. Singh, "Fault tolerance techniques and comparative implementation in cloud computing," *International Journal of Computer Applications*, vol. 64, no. 14, pp. 37–41, Feb 2013.

[13] G. Vallee, K. Charoenpornwattana, C. Engelmann, A. Tikotekar, C. Leangsuksun, T. Naughton, and S. L. Scott, "A framework for proactive fault tolerance," in *Third International Conference on Availability, Reliability and Security*, March 2008, pp. 659–664.

[14] I. Egwutuoha, S. Chen, D. Levy, B. Selic, and R. Calvo, "A proactive fault tolerance approach to high performance computing (hpc) in the cloud," in *International Conference on Cloud and Green Computing (CGC)*, Nov 2012, pp. 268–273.

[15] Z. Amin, H. Singh, and N. Sethi, "Review on fault tolerance techniques in cloud computing," *International Journal of Computer Applications*, vol. 116, no. 18, pp. 11–17, Apr 2015.

[16] G. P. Sarmila, N. Gnanambigai, and P. Dinadayalan, "Survey on fault tolerant - load balancing algorithms in cloud computing," in *International Conference on Electronics and Communication Systems*, Feb 2015, pp. 1715–1720.

[17] J. Pinto, P. Jain, and T. Kumar, "Hadoop distributed computing clusters for fault prediction," in *International Computer Science and Engineering Conference (ICSEC)*, Dec 2016, pp. 1–6.

[18] Y. Kwon, M. Balazinska, and A. Greenberg, "Fault-tolerant stream processing using a distributed, replicated file system," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 574–585, Aug 2008.

[19] F. Farivar and M. N. Ahmadabadi, "Continuous reinforcement learning to robust fault tolerant control for a class of unknown nonlinear systems," *Applied Soft Computing*, vol. 37, pp. 702–714, 2015.

[20] C.-Z. Xu, J. Rao, and X. Bu, "Url: A unified reinforcement learning approach for autonomic cloud management," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 95–105, 2012.

[21] J. Wu, X. Xu, P. Zhang, and C. Liu, "A novel multi-agent reinforcement learning approach for job scheduling in grid computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 430–439, 2011.

[22] L. Liu, Z. Wang, and H. Zhang, "Adaptive NN fault-tolerant control for discrete-time systems in triangular forms with actuator fault," *Neurocomputing*, vol. 152, pp. 209–221, 2015.

[23] A. Forster and A. L. Murphy, "Froms: A failure tolerant and mobility enabled multicast routing paradigm with reinforcement learning for WSNs," *Ad Hoc Networks*, vol. 9, no. 5, pp. 940 – 965, 2011.

[24] Y.-C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 1, pp. 73–82, 2005.

[25] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Design, Automation & Test in Europe Conference & Exhibition*, 2015, pp. 1521–1526.

[26] R. Jhawar and V. Piuri, "Chapter 1 - fault tolerance and resilience in cloud computing environments," in *Cyber Security and IT Infrastructure Protection*, J. R. Vacca, Ed. Boston: Syngress, 2014, pp. 1 – 28.

[27] H. Okamura, Y. Nishimura, and T. Dohi, "A dynamic checkpointing scheme based on reinforcement learning," in *IEEE Pacific Rim International Symposium on Dependable Computing*, March 2004, pp. 151–158.

[28] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka, "Fti: High performance fault tolerance interface for hybrid systems," in *ACM/IEEE International Conference for High Per-*

formance Computing, Networking, Storage and Analysis, 2011, pp. 32:1–32:32.

[29] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[30] S. Di, L. Bautista-Gomez, and F. Cappello, "Optimization of a multilevel checkpoint model with uncertain execution scales," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 907–918.

[31] D. Singh, J. Singh, and A. Chhabra, "High availability of clouds: Failover strategies for cloud computing using integrated checkpointing algorithms," in *International Conference on Communication Systems and Network Technologies*, May 2012, pp. 698–703.

[32] A. J. Oliner, L. Rudolph, and R. K. Sahoo, "Cooperative checkpointing: A robust approach to large-scale systems reliability," in *ACM Annual International Conference on Supercomputing*, 2006, pp. 14–23.

[33] S. Di, M. S. Bouguerra, L. Bautista-Gomez, and F. Cappello, "Optimization of multi-level checkpoint model for large scale hpc applications," in *IEEE International Parallel and Distributed Processing Symposium*, May 2014, pp. 1181–1190.

[34] B. Mohammed, M. Kiran, K. M. Maiyama, M. M. Kamala, and I.-U. Awan, "Failover strategy for fault tolerance in cloud computing environment," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1243–1274, 2017.

[35] I. Jangjaimon and N. F. Tzeng, "Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 396–409, Feb 2015.

[36] J. Zhao, Y. Xiang, T. Lan, H. H. Huang, and S. Subramaniam, "Elastic reliability optimization through peer-to-peer checkpointing in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 491–502, Feb 2017.

[37] M. Amoon, "Adaptive framework for reliable cloud computing environment," *IEEE Access*, vol. 4, pp. 9469–9478, 2016.

[38] B. Balasubramanian and V. K. Garg, "Fault tolerance in distributed systems using fused data structures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 4, pp. 701–715, 2013.

[39] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: High availability via asynchronous virtual machine replication," in *USENIX Symposium on Networked Systems Design and Implementation*, 2008, pp. 161–174.

[40] W. Zhao, P. Melliar-Smith, and L. Moser, "Fault tolerance middleware for cloud computing," in *IEEE International Conference on Cloud Computing (CLOUD)*, July 2010, pp. 67–74.

[41] T. Wood, R. Singh, A. Venkataramani, P. Shenoy, and E. Cecchet, "Zz and the art of practical bft execution," in *ACM EuroSys Conference on Computer Systems*, 2011, pp. 123–138.

[42] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, Nov 2002.

[43] P. Costa, M. Pasin, A. Bessani, and M. Correia, "Byzantine fault-tolerant mapreduce: Faults are not just crashes," in *IEEE International Conference on Cloud Computing Technology and Science*, Nov 2011, pp. 32–39.

[44] K. Plankensteiner, R. Prodan, and T. Fahringer, "A new fault tolerance heuristic for scientific workflows in highly distributed environments based on resubmission impact," in *IEEE International Conference on e-Science*, Dec 2009, pp. 313–320.

[45] A. Zhou, S. Wang, C.-H. Hsu, M. H. Kim, and K.-s. Wong, "Network failure-aware redundant virtual machine placement in a cloud data center," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, pp. e4290–n/a, 2017.

[46] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "Component ranking for fault-tolerant cloud applications," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 540–550, Fourth 2012.

[47] R. Jhawar, V. Piuri, and M. Santambrogio, "Fault tolerance management in cloud computing: A system-level perspective," *IEEE Systems Journal*, vol. 7, no. 2, pp. 288–297, June 2013.

[48] C. Wang, L. Xing, H. Wang, Z. Zhang, and Y. Dai, "Processing time analysis of cloud services with retrying fault-tolerance technique," in *IEEE International Conference on Communications in China*, Aug 2012, pp. 63–67.

[49] G. Ramalingam and K. Vaswani, "Fault tolerance via idempotence," *SIGPLAN Not.*, vol. 48, no. 1, pp. 249–262, Jan 2013.

[50] A. Liu, Q. Li, L. Huang, and M. Xiao, "Facts: A framework for fault-tolerant composition of transactional web services," *IEEE Transactions on Services Computing*, vol. 3, no. 1, pp. 46–59, Jan 2010.

[51] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *ACM SIGMOD Record*, vol. 34, no. 3, pp. 44–49, Sep 2005.

[52] I. Hernandez and M. Cole, "Reliable dag scheduling on grids with rewinding and migration," in *ICST International Conference on Networks for Grid Applications*, 2007, pp. 3:1–3:8.

[53] F. Saffre, R. Tateson, J. Halloy, M. Shackleton, and J. L. Deneubourg, "Aggregation dynamics in overlay networks and their implications for self-organized distributed applications," *The Computer Journal*, vol. 52, no. 4, pp. 397–412, Jul 2009.

[54] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *IEEE International Conference on Advanced Information Networking and Applications*

Workshops, Apr 2010, pp. 551–556.

[55] M. Randles, A. Taleb-Bendiab, and D. Lamb, "Scalable self-governance using service communities as ambients," in *World Conference on Services - I*, Jul 2009, pp. 813–820.

[56] O. Rahmeh, P. Johnson, and A. Taleb-Bendiab, "A dynamic biased random sampling scheme for scalable and reliable grid networks," *INFOCOMP Journal of Computer Science*, vol. 7, no. 4, 2008.

[57] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud," in *USENIX Conference on Security Symposium*, 2008, pp. 91–106.

[58] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian, "Virtualized in-cloud security services for mobile devices," in *ACM Workshop on Virtualization in Mobile Computing*, 2008, pp. 31–35.

[59] K.-L. Peng, C.-Y. Huang, P.-H. Wang, and C.-J. Hsu, "Enhanced n-version programming and recovery block techniques for web service systems," in *ACM International Workshop on Innovative Software Development Methodologies and Practices*, 2014, pp. 11–20.

[60] A. Avizienis, "The n-version approach to fault-tolerant software," *IEEE Transactions on Software Engineering*, vol. 11, no. 12, pp. 1491–1501, Dec 1985.

[61] P. Hosek and C. Cadar, "Varan the unbelievable: An efficient n-version execution framework," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015, pp. 339–353.

[62] R. Hanmer, "Software rejuvenation," in *ACM Conference on Pattern Languages of Programs*, 2010, pp. 21:1–21:13.

[63] M. Melo, J. Araujo, R. Matos, J. Menezes, and P. Maciel, "Comparative analysis of migration-based rejuvenation schedules on cloud availability," in *IEEE International Conference on Systems, Man, and Cybernetics*, Oct 2013, pp. 4110–4115.

[64] F. Xin-yuan, X. Guo-zhi, Y. Ren-dong, Z. Hao, and J. Le-tian, "Performance analysis of software rejuvenation," in *International Conference on Parallel and Distributed Computing, Applications and Technologies*, Aug 2003, pp. 562–566.

[65] R. Angarita, M. Rukoz, M. Manouvrier, and Y. Cardinale, "A knowledge-based approach for self-healing service-oriented applications," in *ACM International Conference on Management of Digital EcoSystems*, 2016, pp. 1–8.

[66] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[67] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, "Fulfilling the vision of autonomic computing," *Computer*, vol. 43, no. 1, pp. 35–41, Jan 2010.

[68] S. George, D. Evans, and L. Davidson, "A biologically inspired programming model for self-healing systems," in *ACM Workshop on Self-healing Systems*, 2002, pp. 102–104.

[69] C. Engelmann, G. Vallee, T. Naughton, and S. Scott, "Proactive fault tolerance using preemptive migration," in *Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Feb 2009, pp. 252–257.

[70] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for hpc with xen virtualization," in *ACM Annual International Conference on Supercomputing*, 2007, pp. 23–32.

[71] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," in *ACM Workshop on Virtualized Infrastructure Systems and Architectures*, 2009, pp. 37–44.

[72] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines," *ACM SIGPLAN Notices*, vol. 46, no. 7, pp. 121–132, Mar 2011.

[73] P. Lu, A. Barbalace, and B. Ravindran, "Hsg-lm: Hybrid-copy speculative guest os live migration without hypervisor," in *ACM International Systems and Storage Conference*, 2013, pp. 2:1–2:11.

[74] G. Dhiman, G. Marchetti, and T. Rosing, "vgreen: A system for energy-efficient management of virtual machines," *ACM Transactions on Design Automation of Electronic Systems*, vol. 16, no. 1, pp. 6:1–6:27, Nov 2010.

[75] T. Knauth and C. Fetzer, "Vecycle: Recycling vm checkpoints for faster migrations," in *ACM Annual Middleware Conference*, 2015, pp. 210–221.

[76] J. Li, C. Pu, Y. Chen, V. Talwar, and D. Milojicic, "Improving preemptive scheduling with application-transparent checkpointing in shared clusters," in *ACM Annual Middleware Conference*, 2015, pp. 222–234.

[77] A. Polze, P. Troger, and F. Salfner, "Timely virtual machine migration for pro-active fault tolerance," in *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, Mar 2011, pp. 234–243.

[78] Y. Zhong, J. Xu, Q. Li, H. Zhang, and F. Liu, "Memory state transfer optimization for pre-copy based live vm migration," in *IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, Sep 2014, pp. 290–293.

[79] C. H. A. Costa, Y. Park, B. S. Rosenburg, C.-Y. Cher, and K. D. Ryu, "A system software approach to proactive memory-error avoidance," in *IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 707–718.

[80] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into hpc systems," in *IEEE International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 77:1–77:11.

[81] O. Hannache and M. Batouche, "Probabilistic model for evaluating a proactive fault tolerance approach in the cloud," in *IEEE International Conference on Service Operations And Logistics, And Informatics*, Nov 2015,

pp. 94–99.

[82] R. Rajachandrasekar, X. Besseron, and D. K. Panda, "Monitoring and predicting hardware failures in hpc clusters with ftb-ipmi," in *IEEE International Parallel and Distributed Processing Symposium Workshops PhD Forum*, May 2012, pp. 1136–1143.

[83] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys*, vol. 42, no. 3, pp. 10:1–10:42, Mar 2010.

[84] A. Tikotekar, G. Vallee, T. Naughton, S. Scott, and C. Leangsuksun, "Evaluation of fault-tolerant policies using simulation," in *IEEE International Conference on Cluster Computing*, Sep 2007, pp. 303–311.

[85] I. Egwutuoha, S. Chen, D. Levy, and B. Selic, "A fault tolerance framework for high performance computing in cloud," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2012, pp. 709–710.

[86] T. Tung, S. Y. Chaw, Q. Xie, and Q. Zhu, "Highly resilient systems for cloud," in *IEEE International Conference on Web Services*, June 2012, pp. 678–680.

[87] T. Limoncelli, "Resilience engineering: Learning to embrace failure," *Communications of the ACM*, vol. 55, no. 11, pp. 40–47, Nov 2012.

[88] A. Benso and P. Prinetto, Eds., *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*, ser. Frontiers in Electronic Testing. Springer, 2003, vol. 23.

[89] H. J. C. Berendsen, D. V. D. Spoel, and R. V. Drunen, "Gromacs: A message-passing parallel molecular dynamics implementation," *Computer Physics Communications*, vol. 91, pp. 43–56, 1995.

[90] J. Park, H. Yu, K. Chung, and E. Lee, "Markov chain based monitoring service for fault tolerance in mobile cloud computing," in *IEEE Workshops of International Conference on Advanced Information Networking and Applications*, Mar 2011, pp. 520–525.

[91] C. Colman-Meixner, C. Develder, M. Tornatore, and B. Mukherjee, "A survey on resiliency techniques in cloud computing infrastructures and applications," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2244–2281, Third 2016.

[92] H. Li and S. Venugopal, "Using reinforcement learning for controlling an elastic web application hosting platform," in *ACM International Conference on Autonomic Computing*, 2011, pp. 205–208.

[93] R. S. Sutton and A. G. Barto, *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[94] T. G. Dietterich and X. Wang, *Support Vectors for Reinforcement Learning*. Berlin, Heidelberg: Springer, 2001, pp. 600–600.

[95] Y. Li, "Deep Reinforcement Learning: An Overview," *ArXiv e-prints*, Jan. 2017.

[96] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A Brief Survey of Deep Reinforcement Learning," *ArXiv e-prints*, Aug. 2017.

[97] A. Paul and D. P. Mukherjee, "Reinforced random forest," in *ACM Indian Conference on Computer Vision, Graphics and Image Processing*, 2016, pp. 1:1–1:8.

[98] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, no. 10, pp. 78–87, Oct. 2012.

[99] K. M. Sim, "Agent-based cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 564–577, Oct.-Dec. 2012.

[100] K. M. SIM, "Agent-based approaches for intelligent intercloud resource allocation," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[101] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2017, pp. 64–73.

[102] D. Cui, Z. Peng, x. jianbin, b. xu, and W. Lin, "A reinforcement learning-based mixed job scheduler scheme for grid or iaas cloud," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[103] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2015.

[104] A. Dalília and L. R. Coutinho, "A fault tolerance approach based on reinforcement learning in the context of autonomic opportunistic grids," in *International Conference on Autonomic and Autonomous Systems*, 2014.

[105] J. F. Murray, G. F. Hughes, and D. Schuurmans, "Machine learning methods for predicting failures in hard drives: A multiple-instance application," *Journal of Machine Learning Research*, vol. 6, p. 816, 2005.

[106] Y. Zhao, X. Liu, S. Gan, and W. Zheng, "Predicting disk failures with hmm- and hsmm-based approaches," in *International Conference on Advances in Data Mining. Applications and Theoretical Aspects*, 2010, pp. 390–404.

[107] B. Zhu, G. Wang, X. Liu, D. Hu, S. Lin, and J. Ma, "Proactive drive failure prediction for large scale storage systems," in *IEEE Symposium on Mass Storage Systems and Technologies*, 2013, pp. 1–5.

[108] Y. Wang, Q. Miao, E. Ma, K.-L. Tsui, and M. Pecht, "Online anomaly detection for hard disk drives based on mahalanobis distance," *IEEE Transactions on Reliability*, vol. 62, no. 1, pp. 136–145, Mar 2013.

[109] J. Li, X. Ji, Y. Jia, B. Zhu, G. Wang, Z. Li, and X. Liu, "Hard drive failure prediction using classification and regression trees," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 383–394.

[110] X. Qin, W. Zhang, W. Wang, J. Wei, H. Zhong, and T. Huang, "On-line cache strategy reconfiguration for elastic caching platform: A machine learning approach," in *IEEE Annual Computer Software and Applications Conference*, July 2011, pp. 523–534.