Fault Tolerance in the Cloud

Kashif Bilal*[1], Osman Khalid[1], Saif Ur Rehman Malik[1], Muhammad Usman Shahid Khan[1], Samee U. Khan[1], and Albert Zomaya[2]

[1]{kashif.bilal*, osman.khalid, saif.rehmanmalik, ushahid.khan, samee.khan}@ndsu.edu
[2]albert.zomaya@sydney.edu.au

[1]North Dakota State University, Fargo, ND, USA.
[2]School of Information Technologies, Sydney University, Australia.

Keywords: Cloud Computing, Fault Tolerance, Service Level Agreement, Verification.

1. INTRODUCTION

Cloud Computing is an emerging and innovative platform, which makes *computing* and *storage* available to the end-users as *services*. The cloud is a "blob" of unstructured resources that are classified into three domains: **(a)** applications (or software), **(b)** platform, and **(c)** infrastructure. The cloud is a merger of business and computing models, which makes it a very important scientific and business medium for the end-users. Cloud Computing has established a widespread adoption in various domains, such as research, business, health, e-commerce, agriculture, and social life. Recently, cloud computing has increasingly been employed for a wide range of applications in various research domains, such as agriculture, smart grids, e-commerce, scientific applications, healthcare, and nuclear science. In the "Market Trends" report by Gartner, it is estimated that the cloud-based business services and Software-as-a-Service (SaaS) market will increase from \$13.4 to \$32.2 billion from 2011 to 2016. Similarly, Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) market is estimated to grow from \$7.6 billion to \$35.5 billion from 2011 to 2016. The cloud investments have delivered around \$4 billion benefit yield in the last five years.

As the cloud computing systems continue to grow in scale and complexity, it is of critical importance to ensure the stability, availability, and reliability in such systems. The varying execution environments, addition and removal of system components, frequent updates and upgrades, online repairs, intensive workload on servers, to name a few, are the primary reasons that can induce failures and faults in the large-scaled complex and dynamic environments of cloud computing. The reliability of such systems can be easily compromised if the proactive measures are not taken to tackle against the possible failures emerging in cloud subsystems. For instance, Google reported a 20 percent revenue loss, when an experiment caused an additional delay of 500 ms in the response time (Greenberg *et al.* 2009). Moreover, Amazon reported a 1 percent sales decrease for an additional delay of 100 ms in search results (Greenberg *et al* 2009). A minor failure in the O2 network (a leading cellular service provider in the UK) affected around seven million customers for three days (ITProPortal 2012). Similarly, a core switch failure in the BlackBerry's network left millions of customers without Internet access for three days (ITProPortal 2012).

To achieve reliability, and as a countermeasure for faults and failures, the cloud service providers adopt various mechanisms to implement *fault tolerance* at the system level. Fault tolerance is a vital issue in cloud computing platforms and applications. Fault tolerance enables a system to continue operation, possibly at a reduced level, rather than failing completely, when some subcomponent of the system malfunctions unexpectedly. The significance of the interconnection networks is obvious from the abovementioned discussion, providing adequate evidences for the fault tolerance requirement of the network.

Because of the diversity in nature and needs of various services deployed in a cloud environment, fault tolerance also has a pivotal role in maintaining the Service Level Agreements (SLAs), as well as, the desired levels of Quality of Service (QoS). The SLAs define various rules to regulate the availability of the cloud service to the end users. Virtualization in cloud assigns various services at different levels of access to numerous subscribers. Virtualization and multiple subscriptions with diversifying SLAs and QoS requirements significantly raise the complexity and unpredictability of cloud environments.

1.1. Types of Faults

The faults can be of various types including: **(a)** transient, intermittent, or permanent hardware faults, **(b)** software bugs and design errors, **(c)** operator errors, and **(d)** externally induced faults and errors. In a typical cloud environment, the faults appear as failure of resources, such as applications/hardware storage, that are being used by the end users. The two most commonly occurring faults in cloud environment are:

A) Byzantine failures

In such type of faults, the system components fail in arbitrary ways, causing the system to incorrectly behave in unpredictable manner. The system may process requests incorrectly and produce inconsistent outputs.

B) Crash failures

When the crash failures occur, they cause the system components to completely stop functioning or remain inactive during failures. For instance, the failures due to power outages or hard disk crash.

As discussed earlier, the cloud computing is divided into several operational layers, such as PaaS, SaaS, and IaaS. If a failure occurs in one of the aforementioned layers, then this layer affects the services offered by the layers above it. For instance, failure in PaaS may produce errors in the software services offered by SaaS. However, if a failure occurs in physical hardware layer (IaaS), then this may negatively affect both the PaaS and SaaS layers. This implies that the impact of failures occurred at hardware level is significantly high, and it is of critical importance to devise fault-tolerant strategies at hardware level. The supporting research in the fault-tolerance computing involves system architecture, design techniques, coding theory, testing, validation, proof of correctness, modeling, software reliability, operating systems, parallel processing, and real-time processing.

1.2. Redundancy

A practical approach of implementing fault tolerance is through redundancy that involves duplication of hardware and software components, such that if a component or a process fails, the backup process or component is available to take place of primary. Some of the vendors have been involved in developing computing solutions with built-in capability of fault tolerance. For instance, *Stratus Computers* produce a duplex-self checking computers, where each computer belonging to a duplex pair is internally duplicated and runs synchronously. If one of the machine fails, the duplication allows the other machine of the pair to continue the computations without delay. Similarly, the *Tandem Computers* utilize a number of independent identical processors and redundant storage devices and controllers to provide automatic recovery in the case of a hardware or software failure.

1.3. Fault-Tolerance Validation

The cloud service providers need to perform the fault tolerance/availability analysis of the services they provide to the end users. The fault-tolerance validation of a service is of critical importance to ensure the SLAs are properly adhered. However, due to a numerous stochastic factors involved, it is quite difficult to verify that a fault-tolerant machine will meet the reliability requirements. To aid in the assessment of system reliability, a great deal of research has been conducted recently in experimental testing by making use of a methodology known as *fault-injection.* Fault injection is an important method to mimic the occurrence of errors in a controlled environment to make the necessary measurements. A number of stochastic models based on probability computations have been developed that utilize Markov and semi-Markov processes to perform the availability analysis of a fault-tolerant machine. These models have been implemented in several computer-aided design tools. Some of the well-known tools are: **(a)** HARP–Hybrid Automated Reliability Predictor (developed in Duke University), **(b)** SAVE–System Availability Estimator (IBM), **(c)** SHARPE–Symbolic Hierarchical Automated Reliability and Performance Evaluator (Duke University), **(d)** UltraSAN– (University of Illinois, UIUC), and **(e)** DEPEND – (UIUC). The aforementioned tools are utilized to perform various measures of interest, such as latency, coverage, and fault rates. We define some of the fault-tolerance measurements in the next subsection.

1.4. Fault Tolerance Measures

Fault tolerance measurement is a crucial aspect of cloud paradigm. Fault tolerance measures can be used to quantify the dependability of the cloud system. Two of the major legacy measures for fault tolerance of the system are: **(a)** availability and **(b)** reliability (Koren & Krishna 2010). Availability is the ratio between the uptime and sum of the uptime and downtime of the system. Availability can be quantified as:

$$Availability = \frac{Uptime}{Uptime + Downtime}. \qquad (1)$$

The uptime and downtime values are either predicted by using the mathematical modeling techniques, such as Markov availability model, or can be calculated from the actual field measurements (Bauer & Adams 2012). Availability can also be measured as a percentage of agreed service time and downtime of system. The agreed service time is the expected operational time of the system per month. The planned downtime of the system is explicitly excluded from the agreed service time. The availability of the system can be calculated as:

$$Availability\ (\%) = \frac{AgreedServiceTime - Downtime}{AgreedServiceTime} \times 100\,. \qquad (2)$$

Availability of the system can also be quantified using metrics, such as Mean Time to Failure (MTTF), Mean Time Between Failures (MTBF), and Mean Time to Repair (MTTR). The MTTF is the average time of the system operating accurately until a failure occurs. The MTBF is the average time between two consecutive failures of the system. The MTTR measure predicts the average time required to replace the faulty component of the system to bring the system back to operational mode. Availability of the system in terms of MTBF and MTTR can be computed as:

$$Availability = \frac{MTBF}{MTBF + MTTR} \qquad (3)$$

Reliability, denoted as R(t), is the probability of the system to work accurately as a function of time 't' (Koren & Krishna 2010). The TL9000 measurement handbook defines reliability as "the ability of an item to perform a required function under stated conditions for a stated time period" (Bauer & Adams 2012). Service reliability can be formulated as:

$$Service\ Reliability = \frac{Successful\ Responses}{Total\ Requests} \times 100\,. \qquad (4)$$

Because most of the services are highly reliable, services are quantified by the defective/ unsuccessful transactions per million attempts. Defects Per Million (DPM) can be formulates as:

$$DPM = \frac{(Total\ Requests - Successful\ transactions)}{Total\ Request} \times 1{,}000{,}000\,. \qquad (5)$$

The availability and reliability of the system holds a pivotal role in the cloud paradigm. A small fraction of downtime has severe financial impacts. It has been reported that a single hour of downtime costs around $50,000 in a data center (Bilal *et al.* 2014). Therefore, round the clock availability of the cloud services are vital.

In this chapter, we discuss fault tolerance in the cloud and illustrate various fault tolerance strategies existing in the literature. The rest of the chapter is as follows. In Section 2, we discuss the different fault tolerant strategies for cloud and provide taxonomy of fault-tolerant approaches. Section 3 concludes the book chapter.

2. FAULT TOLERANT STRATEGIES IN CLOUD

Crash and failures are very common in the cloud, such as disk failure or link failure. Moreover, the number of nodes (servers) involved in the cloud has an order of tens of thousands or more servers that increases the probability and cost of the failures. The jobs executing over the cloud may have a time span that may evolve for few days. The effect of failure on medium or long running jobs can lead to jeopardizing the fulfillment of SLA contract and wastage of computation time. For instance, a task requires 24 hours to complete, and if after 23 hours the node that was executing the task crashes, then almost one day of execution will be wasted in the aforementioned scenario and it will also lead to the violation of the SLA. One way to tackle such problem is to execute a backup process on a different system for each primary process. The primary process in this case is responsible for checkpointing the current state to redundant disks. Should the primary process fail, the backup process can restart from the last checkpoint. Generally, the fault-tolerant systems characterize the recovery from errors as either roll-forward or roll-back. Roll forward mechanism takes the system state at the time when the error was detected in order to correct the error, and from there the system moves forward. Alternatively, the roll-back mechanism utilize checkpointing to revert the system state to some earlier correct state, and the system moves forward from that state. The operations between the checkpoint and the erroneous state can be made idempotent, as per requirement of roll-back recovery. Some systems make use of both roll-forward and roll-back recovery for different errors or different parts of one error. We discuss checkpointing in detail in the next subsection.

2.1. Checkpoint-based Fault Tolerance

To avoid the aforesaid problem, checkpoint mechanisms have been proposed and implemented on the cloud. This mechanism records the system state periodically after certain time limit so that if a failure occurs, the last checkpoint state of the system is restored and the task execution is resumed from that point. However, significant overheads are associated with the application of checkpoint strategy as they can be expensive in terms of performance. In case of virtualized environment, such as the cloud, the checkpoint strategy becomes more challenging, where huge Virtual Machine (VM) images needs to saved and restored (Goiri *et al.* 2010). In the said perspective, several researchers have proposed different approaches to make the use of checkpoint efficient in the cloud. In the following subsections, we will discuss and highlight some of the recent checkpoint based fault tolerant approaches that are deployed in the cloud.

A) Disk- and diskless-based checkpointing schemes

Message Passing Interface (MPI) is generally used to achieve parallelism in high performance computing. The MPI is a language-independent communications protocol that supports both point-to-point and collective communications. The goal of MPI is to attain high performance, scalability, and portability. To achieve fault tolerance in MPI-based applications, several strategies have been proposed, such as BlobCR (Nicolae & Cappello 2011) that are specifically optimized for tightly coupled scientific applications written using MPI and are needed to be ported to IaaS cloud. Two techniques are widely adopted to achieve fault tolerance in cloud, replication (redundancy) and checkpoint. For tightly coupled applications, redundancy implies that all components of the process (which is itself part of a distributed application) must also be replicated. The reason for such replication is that a failure of one process results in a global failure of all processes and eventually leads to process termination. Therefore, in tightly coupled applications, checkpoint approaches provide a feasible solution than replication. The BlobCR use a dedicated repository of checkpoints that periodically takes the snapshots of the disk attached to the virtual machine. The aforesaid approach allow using any checkpointing protocol to save the state of processes into files, including application level (where the process state is managed by the application itself) and process level mechanism (where the process state is managed transparently at the level of the message passing library). Moreover, the BlobCR also introduces a support that allows I/O operations performed by the application to rollback. The BlobCR brings a checkpointing time speedup of up to 8x as compared to the full VM snapshotting based on qcow2 over Parallel Virtual File System (PVFS) (Gagne 2007). The checkpoints performed on the whole state of VM are expensive. However, using only a snapshot of the portion of disk is smaller and faster, even if a reboot is needed.

The disk-based checkpointing strategies are widely used. However, the applications that require checkpointing frequently, leads to several disk accesses that results in a performance bottleneck. In the said perspective, several diskless-based checkpoint strategies have been proposed, such as a multi-level diskless checkpointing (Hakkarinen & Chen 2013). In multi-level checkpointing, there are *N*-level of diskless checkpoints. Multiple levels of checkpoints reduce the overhead for tolerating a simultaneous failure of *N* processors by layering the diskless checkpointing schemes for a simultaneous failure of *i* processors. For example, an *N*-failure checkpoint can recover any number of failures from *1, 2,…, N*. The multi-level diskless checkpointing scheme can significantly reduce the fault tolerance time as compared to a one-level scheme. To perform the multi-level diskless checkpointing, a schedule for diskless checkpoints must be developed for each level of recovery, such as one, two, or *N* simultaneous failures. When the checkpoints are scheduled, the processor takes specific steps to perform the recovery. For instance, if one-level checkpoint is scheduled, then processor will take only one step back for the recovery. The coordination of the checkpoints among the processors is important for the consistency of the checkpoints. If a failure is detected, then an *N*-level diskless checkpointing mechanism will attempt to use the most recent checkpoint to recover the state. However, if another failure occurs during the recovery, then the mechanism will use most recent two-level checkpoint to restore the state. If the number of failure exceeds the number that is supported, then the system needs to restart the computation. For any diskless checkpoint there is an overhead of both communication and calculation. The difference between the disk and diskless checkpointing is prominent when a failure occurs during a recovery. The system simply restarts from the same checkpoint in disk-based checkpointing. However, in diskless checkpointing, an earlier checkpoint is used to restore the state of the system.

B) Checkpoint placement scheme

The number of checkpoints inserted can significantly decrease the performance of the system during recovery. Moreover, the number of checkpoints must be optimal to minimize the storage. In the said perspective, several optimized and efficient checkpoint placement strategies are proposed. One such optimal checkpoint strategy that maximizes the probability of completing all tasks within the deadline is presented in (Kwak & Yang 2012). In real time systems, the occurrence of faults is normal rather than exceptions. Every fault is detected at the checkpoint that comes first after the fault occurrence. In such strategies, the checkpoints are inserted constantly after certain interval at the execution of the tasks. However, the time limit for the interval may vary task to task in general. The slack time of each task is identified first and then the maximum number of re-executable checkpoints are determined that can meet the deadline of the task. In multi-tasking environment, the slack time is calculated not only by the execution time of the task but also of execution of the other tasks. Based on the information of the slack time, formulas are derived that computes the number of checkpoints needs to be re-executed to meet the deadline of the task. The significance of such checkpoint schemes are that they provide an integrated solution to multiple real-time tasks by solving a single numerical optimization problem.

Some smart infrastructures are also proposed, such as in (Goiri *et al.* 2010) that uses Another Union File System (AUFS), which differentiate between the read-only and read-write parts of the virtual machine image file. The goal of such infrastructures is to reduce the time needed to make a checkpoint that will in return reduce the interference time on task execution. The read-only parts can be checkpointed only once and the read-write checkpoints are incrementally checkpointed, which means the modifications from the last checkpoints are restored. The aforesaid checkpoint mechanism can also be implemented in virtualized environment using Xen hypervisor, where the tasks can be executed on VMs created on demand. Making the checkpoint of tasks running within a VM may involve moving tons of GBs data as it may include the information to resume the task on another node, such as task and memory content, and disks information (Malik *et al.* 2013). Some checkpoints mechanism mounts the base system as a read only file system considering the fact that only a small portion of the aforementioned data is changing with respect to the VM startup. Moreover, the user modifications are stored in an extra disk space called delta disk. Besides delta disk, which contains user modification data, there is another read-only disk that contains the base system. The aforementioned disks are merged together to form a root file system to start the VM. Once the VM is booted, then the user can work with the file system without worrying about the underlying structure. The checkpoints are compressed and are stored in Hadoop File System (HDFS) so that the checkpoints are distributed and replicated in all the nodes. Moreover, by doing the aforesaid the possibility of single point of failure can also been eliminated.

C) Failures/Faults those are hard to recover

Among all different kind of failures or faults, such as node failures and link failures, the faults that are really costly and takes longer time to mitigate are Hardware Failures. In case of cloud computing, several VMs are running on a single physical machine. In such an environment, if a hardware failure occurs, then all the VMs have to be migrated that incur longer downtime as compared to a software or application failure. Moreover, the hardware device may require to be replaced, resulting in longer repair times. The reason for the hardware failures to have a significant affect is the fact that it may involve **(a)** device replacement, and **(b)** migrations, including VM migrations, which causes the recovery time to increase. Some faults and errors are hard to detect, such as routing and network misconfigurations. Such kind of faults becomes hard to recover as it is hard to detect those failures. However, once detected the aforesaid faults becomes easy to solve and handle.

2.2 Adaptive Fault Tolerance Techniques

Adaptive fault tolerance techniques help the system to maintain and improve the system's fault tolerance by adapting to the environmental changes. The adaptive fault tolerance techniques for the cloud computing, monitors the state of the system and reconfigure the cloud computing system configurations for the stability of the system in case of error detections. In this subsection, we will overview, some of the recently proposed adaptive fault tolerance techniques for the cloud computing paradigm.

A) Fault Tolerance for real time applications (FTRT)

The real-time applications that execute on cloud environments range from small mobile phones to large industrial controls. The highly intensive computing capabilities and scalable virtualized environment of the clouds help the systems to execute the tasks in real-time. Most of the real-time systems require high safety and reliability. A fault tolerance model for real-time cloud computing is proposed in (Malik *et al.* 2011). The real-time cloud fault tolerance model revolves around the reliability of the virtual machines. The reliability of the virtual machines is adaptive and changes after ever computing cycle. The proposed technique depends on the adaptive behavior of the reliability weights assigned to each processing node. The increase and decrease in the reliability depends upon the virtual machines to produce the results within the given time frame. The technique uses a metric to evaluate the reliability. The metric assesses the reliability level of node against a given minimum reliability threshold. The nodes are removed if the processing nodes fail to achieve the minimum required reliability level. The primary focus of the system is on the forward recovery mechanism although both forward and backward recovery techniques.

B) Dynamic adaptive fault tolerant strategy (DAFT)

The dynamic adaptive fault tolerant strategy (Sun *et al.* 2013) observes a mathematical relationship between the failure rates and the two most common fault tolerance techniques (checkpoints and replications). The historical data of failure rates helps the cloud computing system to configure itself for the checkpoints or the replicas. A dynamic adaptive checkpoint and replication model is made by combining the above mentioned two techniques to achieve the utmost level of service availability and service level objectives (SLOs). The dynamic adaptive fault tolerant strategy had been evaluated in large-scale cloud data center on fault tolerance degree, fault tolerance overhead, response time, and system centric parameters. The theoretical and experimental results presented in the paper (Sun *et al.* 2013) demonstrate that DAFT provides highly efficient fault tolerance and great SLO satisfaction.

C) Fault and Intrusion tolerant cloud computing Hardpan (FITCH)

The novel fault tolerant architecture for cloud computing, named FITCH (Fault and Intrusion tolerant cloud computing Hardpan) supports the dynamic adaptation of replicated services (Cogo *et al* 2013). The FITCH provides a basic interface for adding, removing, and replacing replicas. The FITCH interface also provides all the low level actions to provide end-to-end service adaptability. The technique was originally designed for the two replication services: a crash fault tolerant web service and a Byzantine fault tolerant (BFT) key-value store based on state machine replication. Both the services when deployed with FITCH are easily extendable and adaptable to various workloads through horizontal and vertical scalability. The number of computing instances that are responsible for providing the service are increased or decreased in the horizontal scalability. When there is a requirement to handle the peak users request or to handle as many faults as possible, the number of computing resources is increased. The decrease in the number of resources is adapted when there is a requirement to save resources and money. The vertical scalability is achieved by increasing or decreasing the size and capacity of the allocated resources. The FITCH adapts the horizontal and vertical scalability depending on the requirement.

D) Byzantine fault tolerance cloud (BFTCloud)

The BFTCloud is a fault tolerant architecture for the voluntary-resource cloud computing (Zhang *et al.* 2011). In voluntary-resource cloud computing, infrastructure consists of numerous user-contributed resources unlike the well managed architecture provided by a large cloud provider. The architecture of BFTCloud is based on a Byzantine fault tolerance approach. The architectures operate on five basic operations, the primary node selection, replica selection, request execution, primary node updating, and replica updating. The primary node is selected based on QoS requirements. The request for the service is handled by the primary node. The primary node also selects the *3f+1* replicas from the pool based on QoS requirements. All the replicas and primary node perform the operation on the request and send back the result to the primary node. Based on the result, the primary node decides to update the other primary node or update the replicas. In primary updating, one of the replicas is updated to primary node. In replica updating, the faulty replica is replaced with a new one. The BFTCloud provides high reliability and fault tolerance along with better performance.

E) Low latency fault tolerance (LLFT) middleware

A low latency fault tolerance (LLFT) middleware uses the leader/follower replication approach (Zhao *et al.* 2010). The middleware consists of a low latency messaging protocol, a leader-determiner membership protocol, and a virtual determiner framework. The low latency message protocol provides a reliable and ordered multicast service by communicating a message ordering information. The ordering is determined by the primary replica in the group. The technique involves fewer messages as compared to prior fault tolerance systems. A fast reconfiguration and recovery service is provided by the membership protocol. The reconfiguration service is required whenever the fault occurs at the replica or some replica joins or leaves the group. The membership protocol is faster as it finds the primary node deterministically based on the rank and the degree of the backups. The virtual determiner framework takes the ordering information from the primary replica and ensures that all the backups get the same ordering information. The LLFT middleware provides a high degree of fault tolerance and achieves low end-to-end latency.

F) Intermediate data fault tolerant (IFT)

The term *intermediate data* is defined as the data that is generated during the parallel dataflow program (Ko *et al.* 2010). The technique considers the intermediate data as of high priority (first class citizen). The other techniques either use the store-local approach or distributed file system (DFS) approach. In store-local approach the data is not replicated and is used in Map outputs in Hadoop. Although the approach is efficient but is not fault tolerant. In case of a failure of a server that stores the intermediate data, results in the re-execution of the tasks. In the DFS approach the data is replicated, but causes too much network overhead. The network overhead results in the delay of jobs completion time. The DFS approach is used for reduce outputs in Hadoop. There are three techniques for intermediate data fault tolerance: 1) asynchronous replication of intermediate data, 2) replication of selective intermediate data, and 3) exploiting the inherent bandwidth of the cloud data center topology. A new storage system termed as Intermediate Storage System (ISS) implements the above mentioned techniques for Hadoop. Hadoop with ISS outperforms the base Hadoop under failure scenarios.

G) MapReduce fault tolerance with low latency (MFTLL)

The MapReduce fault tolerance with low latency technique is a passive replication technique on the top of re-execution of the MapReduce jobs to improve the overall execution time (Zheng 2010). The technique utilizes the extra copies for the tasks in cloud to improve MapReduce fault tolerance while keeping the latency low. The technique is referred as a passive replication technique as in passive replication, all copies do not need to be in running state as compared to the active replication technique. The proposed technique allocates a few (*k*) backup copies of the tasks. The backup assignment for each task is based on data locality and on rack locality. The placement of the back in the locality avoids the heavy network traffic. The backup copy is only executed if the primary task is failed. The resources which take longer time to execute (stragglers) are also identified and for these stragglers backups are executed in parallel. The MapReduce users or cloud providers decide the value of *k* based on the failure statistics. The technique also uses a heuristic to schedule backups, move backup instances, and select the backups upon failure for fast recovery.

H) Adaptive Anomaly Detection System for Cloud Computing Infrastructures (AAD)

An adaptive anomaly detection (AAD) system for cloud computing infrastructure ensures the availability of the cloud (Pannu *et al.* 2012). The framework uses the cloud performance data to discover the future failures. The predicted possible failures are verified by the cloud operators. The failures are marked as true or false failures on verification. The algorithm recursively learns and improve the future failure prediction on the verified data. The framework also takes into account the actual failures that were not previously detected.

In Table 1, we provide a summary of various fault tolerant techniques discussed above. The schemes are categorized on the basis of methodology, programming framework, environment, and the type of faults detected.

Table 1: Summary of fault-tolerant strategies.

Strategy	**Fault Tolerant Technique**	**Programming Framework**	**Environment**	**Faults detected**
(Nicolae & Cappello 2011)	Disk-based Checkpoint	MPI	IaaS cloud	Node/ network failure
(Hakkarinen & Chen 2013)	Diskless-based Checkpoint	NA	HPC	Process/Application failure
(Kwak & Yang 2012)	Checkpoint	Probability analytic framework	Real-time systems	Process failures
(Goiri *et al.* 2010)	Checkpoint	Java	Virtual Machine	Node failure
(Malik *et al.* 2011)	FTRT (Adaptive)	-	Real-time	-
(Sun *et al.* 2013)	DAFT (Adaptive)	Java	Large scale Cloud	Works on historical failure rate
(Cogo *et al* 2013)	FITCH (Adaptive)	Java	Large scale Cloud	-
(Zhang *et al.* 2011)	BFTCloud (Adaptive)	Java	Voluntary-resource cloud	Byzantine Problems

(Zhao *et al.* 2010)	LLFT (Adaptive)	C++	Middleware	Replication faults
(Ko *et al.* 2010)	IFT(Adaptive)	Hadoop	Hadoop	Intermediate data faults
(Zheng 2010)	MFTLL (Adaptive)	MapReduce	MapReduce	Replication faults, stragglers detection
(Pannu *et al.* 2012)	AAD (Adaptive)	---	Local Cloud	Discovers future failures

4. CONCLUSIONS

In this chapter, we study the fault tolerance and illustrate various state of the art fault tolerant strategies for the cloud environments. Fault tolerance is a major concern in the cloud environments to guarantee availability of critical services, application execution, and hardware. As the cloud computing systems continue to grow in their scale and complexity, it is of critical importance to ensure the stability, availability, and reliability in such systems. The cloud environments are susceptible to failure because of varying execution environments, addition and removal of system components, frequent updates and upgrades, online repairs, intensive workload on the servers. The reliability of such systems can be easily compromised if the proactive measures are not taken to tackle the possible failures emerging in the cloud subsystems. We discussed various types of faults, and the different ways that are in use to tackle such faults. The chapter also mentions some the methods to validate the fault-tolerance of a system and the various metrics that quantify the fault-tolerance. In the end, we discuss the state of the art techniques for fault tolerance in cloud and a taxonomy of fault tolerant schemes is also presented.

REFERENCE

Greenberg, A., Hamilton, J., Maltz, D., and Patel, P. (2009) 'The Cost of a Cloud: Research Problems in Data Center Networks,' *ACM SIGCOMM Computer Communication Review,* vol. 39, no. 1, pp. 68-79.

ITProPortal, http://www.itproportal.com/2012/07/12/o2outage-latest-string-major-it-infrastructure-failures/, 2012.

Bilal, K, Malik, S., Khan, S. U., & Zomaya, A. (2014) 'Trends and Challenges in Cloud Data Centers,' *IEEE Cloud Computing Magazine*, vol. 1, no. 1, pp. 10-20, 2014.

Bauer, E., & Adams, R. (2012). *Reliability and availability of cloud computing*: John Wiley & Sons.

Koren, I., & Krishna, C. M. (2010). *Fault-tolerant systems*: Morgan Kaufmann.

Gagne, M, 2007, 'Cooking with Linux—still searching for the ultimate Linux distro?', *Linux Journal*, vol. 09, no. 161.

Goiri, I, Julia, F, Guitart, J, & Torres, J, 2010, 'Checkpoint-based fault-tolerant infrastructure for virtualized service providers', *In IEEE Network Operations and Management Symposium (NOMS)*, pp. 455-462.

Hakkarinen, D, & Chen, Z, 2013, 'Multi-Level Diskless Checkpointing', *IEEE Transactions On Computers*, vol. 62, no.4, pp. 772-783.

Kwak, SW, & Yang, JM, 2012, 'Optimal checkpoint placement on real-time tasks with harmonic periods', *Journal of Computer Science and Technology*, vol. 27, no. 1, pp. 105-112.

Nicolae, B, & Cappello, F, 2011, 'BlobCR: efficient checkpoint-restart for HPC applications on IaaS clouds using virtual disk image snapshots', *ACM International Conference for High Performance Computing, Networking, Storage and Analysis*.

Malik, S, R, Khan, S, U, and Srinivasan, S, K, 2013, 'Modeling and Analysis of State-of-the-art VM-based Cloud Management Platforms', *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 50-63.

Malik, S & Huet, F 2011, 'Adaptive Fault Tolerance in Real Time Cloud Computing', In *IEEE World Congress on Services (SERVICES),* pp. 280-287.

Sun, D, Chang, G, Miao, C, & Wang, X 2013, Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments, *The Journal of Supercomputing*, 1-36.

Cogo, V V, Nogueira, A, Sousa, J, Pasin, M, Reiser, HP, & Bessani, A 2013, 'FITCH: Supporting Adaptive Replicated Services in the Cloud', In *Distributed Applications and Interoperable Systems*, pp. 15-28. Springer Berlin Heidelberg.

Zhang, Y, Zheng, Z, & Lyu, M R 2011, 'BFTCloud: A byzantine fault tolerance framework for voluntary-resource cloud computing', In *IEEE International Conference on Cloud Computing (CLOUD),* pp. 444-451.

Zhao, W, Melliar-Smith, PM, & Moser, L E 2010, 'Fault tolerance middleware for cloud computing', In *IEEE 3rd International Conference on Cloud Computing (CLOUD),* pp. 67-74.

Ko, S Y, Hoque, I, Cho, B, & Gupta, I 2010, 'Making cloud intermediate data fault-tolerant', In *Proceedings of the 1st ACM symposium on Cloud computing,* pp. 181-192.

Zheng, Q. 2010, 'Improving MapReduce fault tolerance in the cloud', In *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW),* pp. 1-6.

Pannu, H S, Liu, J, & Fu, S 2012, ‘AAD: Adaptive Anomaly Detection System for Cloud Computing Infrastructures’, In *IEEE 31st Symposium on Reliable Distributed Systems (SRDS),* pp. 396-397.