National College of Ireland

Fault Tolerance Optimization in Load Balancing using Multiple APIs in distributed Cloud environment

MSc Research Project
MSc Cloud Computing

Balavignesh Kunkulagunta Sanghameswar
Student ID: 22220445

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Balavignesh Kunkulagunta Sanghameswar
Student ID:	22220445
Programme:	MSc Cloud Computing **Year:** 2023
Module:	MSc Research Project
Supervisor:	Rashid Mijumbi
Submission Due Date:	16-09-2024
Project Title:	Fault Tolerance Optimization in Load Balancing using Multiple APIs in a distributed Cloud environment
Word Count:	8480 **Page Count** 23

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Balavignesh Kunkulagunta Sanghameswar

Date: 16-09-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Fault Tolerance Optimization in Load Balancing using Multiple APIs in distributed Cloud environment

Balavignesh Kunkulagunta Sanghameswar

22220445

Abstract

Cloud computing has a significant focus on providing fault tolerance to the systems by implementing strategies and mechanisms to cover the failure of any of the components. This is especially important in cloud environments because distributed infrastructures often contain single point of failure (SPOF). Measures like load balancing, redundancy, failover management, error control, and traffic redirection are used to partition required loads, multiple services in various locations, and restart traffic to other frames when a failure happens. With these methods, it is possible to ensure high availability, reliability as well as an optimal level of applications and service's performance in the clouds, thus keeping the rate of outages and similarly service unavailability at the level as low as possible. This paper focuses on fault tolerance in cloud environments with reference to multiple APIs to improve system reliability of high availability. This research establishes a distributed environment in different cloud platforms, which uses multiple APIs to balance the workload and handle failure conditions. The findings show that the use of geographically dispersed API endpoints, round-robin load balancing, and fault tolerance mechanisms enhance the system availability and reliability in managing the API failure without causing any interruption to the service. This helps in the case of application which require high availability such as in the case of financial and e-commerce activities.

Key words: *Fault tolerance, Multiple API, Cloud Environment, Single point of failure, Optimization*

1 Introduction

In the current ever-revolving technology of cloud computing, fault tolerance can be regarded as an important parameter that increases the reliability of distributed systems. Cloud infrastructure is scalability, flexibility, and most of the structure of cloud infrastructure is virtual infrastructure, which is distributed in multiple data centers and areas. This distributed architecture brings in certain inherent challenges and risks where failure can be in the hardware, software, or the network that can affect the service delivery. Cloud failure is the occurrence of failures in cloud environments and to counter this, fault tolerance is the process of preventing, identifying and managing such failures before they occur (Diouf, et al, 2020). Redundancy is a basic technique in which key components like the server and data storage

are maintained in pairs to enable working even when one of them fails. Load balancing is the partitioning of a workload so that no single server is loaded with a large workload while the other server is idle, but instead all the server workloads are tend to be balanced in such a way that every server is loaded in a proportionate manner to its capacity. Also, failover mechanisms redirect the traffic to the backup systems or other resources in the event of a failure to stop the service interruption. Some of the tools and services that are provided by the cloud providers to improve fault tolerance include automated scaling, replication of data and geographic redundancy across multiple zones of availability. They not only enhance the capability of the system to recover from failures but also enhance the ability of the business to continue the services during planned maintenance or any other type of outage (Yang, et al, 2020).

Cloud computing is a complex and distributed system where fault tolerance is a major concern. In the cloud computing domain, the applications and services are dependent on the network, virtualization of resources, and multiple geographic locations sharing infrastructure, the vulnerability to failures and disruptions rises dramatically. A single point of failure (SPOF) refers to a single component that if fails, causes the entire system to fail and this can be in the form of a server failure, network failure, or a software failure in a critical component. The value of fault tolerance is that it can help to avoid such risks and provide constant availability and reliability of services (Belgacem, et al, 2020). It is possible to point out that effective mechanisms of fault tolerance, which are specifically designed to deal with failures within cloud systems, can be incorporated in advance to prevent their adverse effects on system performance. Some are redundancy, load balance, and fail over, which help in distributing workload across many servers or regions, and so if one of them is down the others can well be allowed to pick the load without interruption. Moreover, with the help of fault tolerance, cloud environments become protected from various accidents which may happen in the future, for example, such as failure of the hardware equipment, natural disasters, cyber-attacks, and even maintenance operations. It gives confidence to enterprises and companies that their key applications as well as info will be available and functional, particularly within the course of disruptive elements (Sharif, et al, 2020).

1.1 Aim

The purpose of this research is to explore and propose effective fault tolerance techniques in clouds multiple APIs approach with the goal of improving the system's reliability and achieve high availability, since distributed systems are vulnerable to SPOFs. This work integrates the setup of distributed systems and applications in virtual different locations by means of cloud infrastructures as well as the construction of multiple APIs, which provide uniform interfaces for help in load dispersal and to achieve fault tolerance. The objective of the study is to implement techniques such as load balancing and failover management to manage the load and ensure that API calls keep running without disruption in the event of failure. To compare the efficiency of the mentioned mechanisms and to determine the reliability coefficient, estimation and measurement of simulated failure conditions, such as uptime and downtime and response time, will be assessed.

1.2 Research Question

What is the best approach to work with many API endpoints consolidated in a distributed cloud environments so that fault tolerance as well as high availability will be achieved?

1.3 Objectives

- To collect information about the overall functionality of the system, how fast the system has performed, the percentage of the successes and failures encountered to assess the efficiency of the fault tolerance mechanisms for data collection.
- To develop several instances of the distributed systems in different geographical areas for the testing purpose to design several APIs that interact with the distributed cloud instances.
- To formulate a function that equally divided the API calls with the availability of the API's and introduce a feature that can forward a call to another API in case one fails
- To imitate the failure of the APIs that are involved in processing the requests and capture the degree of dependency by monitoring the time that the system is available and the time.

1.4 Novelty of the project

The overall contribution of this research is derived from its fresh approach towards minimizing fault tolerance in distributed cloud settings with the help of normal API points as well as multiple API point. Through the observation of multiple API endpoints, several request points from different geographical regions; this research will enhance the feature of high availability and rely on it to enhance system reliability. In contrast to most existing approaches, which usually involve the interaction with only one API link, this study outlines a procedure in which the call to an API is divided among multiple links so that if one link fails, the others remain active. The test study shall yield ample data on this novel work stream performance in different geographical conditions to prove the tool's prowess on ensuring operations perpetuity, and service continuity in instances of disruption.

1.5 Outline

The first chapter establishes the background by presenting fault tolerance as a crucial feature of cloud computing and statistics accuracy and dependability of distributed systems. This section provides an understanding of Application Programming Interfaces (APIs) and raises the importance of load balancing as a vital tool in the determination process of resource usage to avoid performance problems. The chapter also defines the study's purpose, research questions and objectives and serves as a guide toward identifying efficient fault tolerance techniques. Next, the literature review considers the latest studies on fault tolerance in cloud computing, pinpoints the researchers' shortcomings, and explains the selected approaches. In the Research method section, there is an explanation of why AWS was chosen as the cloud

environment for the project, and why FastAPI and REST API were used in API creation, as well as the overall workflow of the project, the stages of system design, function development, testing, and general considerations about ethics in research for the result to be stronger and more reliable.

2 Literature review

The aspect of clouds that is widely discussed in connection with dependability and high availability is fault tolerance, since distributed computing is always prone to single points of failure (SPOFs). This chapter explains the recent trends in fault tolerance solutions able to provide load balancing, fail over mechanisms, and adaptive scheduling. It reveals different approaches such as Fourier Transform based load distribution, anticipative fault handling through CPU temperature probe, and other structures based on the deep reinforcement learning model. However, there is still little research dedicated to the combined usage of multiple APIs to increase the fault tolerance and provide the load distribution across the cloud environment. Therefore, to enhance the progress and construction of the distributed cloud systems configuration, it is crucial to fill the above-discussed gap.

2.1 Fault tolerance in cloud

Shahid, et al. (2020) explore load balancing, a key cloud computing challenge. LB uniformly distributes computing effort among cloud servers to avoid underloading or overloading. Academic researchers have employed load balancing algorithms in order to solve the customer problems for appropriate cloud nodes, to improve cloud service efficiency, and boost user satisfaction. Effective load balancing algorithms split load according to system nodes, improving resource efficiency and optimization. This research paper provides a comprehensive foundation on load balancing strategies which aims to improve load balancing concerns such throughput, efficiency, migration time, reaction time, extra time, utilization, scalability, fault tolerance, and energy optimization. The research paper on load balancing in cloud-computing services discusses current challenges and identifies the need for a new fault tolerance-based solution because typical LB algorithms do not handle FT efficiency metrics during work, they are inadequate. The research found a significant FT efficiency indicator insufficiency in LB algorithms, which is a key worry for cloud systems.

Ray et al. (2020) offers a proactive fault tolerance method that detects and handles federation failures and CPU temperature. Fault tolerance in the federation may be modelled as a multi-objective optimization problem. The system migrates virtual machines from problematic CSPs to non-faulty federation members to maximize utility and minimize migration costs. To address this issue, the authors offer Preference Based Fault Management (PBFM) to manage federation when problems develop. The authors conduct rigorous experiments to test the proposed system and compare it to MCAFM and PAFM. The Profit-Based Fault Management (PBMF) technique finds the best solution to controlling profit and migration cost for problematic CSP applications.

Devi et al. (2021) devised a multi-level fault tolerance scheduling strategy that can safely manage real-time system failures. In the first step, non-functional testing and decision- making algorithms determine virtual machine reliability. The dependable Decision K-Nearest Neighbour (RDK-NN) technique meets reliability indicators, which consider just the most trustworthy virtual machine. A scheduling strategy ensures excellent availability in the second phase. We propose a Teaching-Learning Optimization (TLBO) scheduling approach to increase user friendliness and assignment organization. The recommended approach is evaluated using Cloudsim. Scheduler performance is graded on length, failure ratio, augmentation of performance rate, response time, and rejection ratio. As shown, multi-tiered design improves system reliability and cloud data availability.

The Yuan, et al. (2020) research examines cloud network failure-tolerant VNF implementation. It recommends putting VNFs in pre-configured standby VMs for instantiation as needed. The major issue is that VNFs maintain state information. FT solution must manage stand-by instances carefully since they need updates from current instances. The location, request route, and state transfer of active/stand-by VNF instances must be evaluated jointly. The authors solved this problem by creating an effective heuristic for placing fault-resistant VNFs. In addition to this the authors also provide two bi-criteria approximation techniques that have proven approximation ratios for the problem without computing or bandwidth constraints. We next discuss the second component of dynamic fault recovery in VNFs, focusing on problematic VNFs' active instances. The authors propose an approximation technique for traffic processing handoffs from malfunctioning VNFs to backups. Estimating prospective outcomes with actual parameter values shows that the algorithms may greatly boost request admission compared to present practices while making it more efficient in certain areas. The authors used a test bed with physical and virtual switches to evaluate the solution's dynamic fault recovery capabilities. The research shows that the strategies work, therefore they may be used in real life.

2.2 Studies on load balancing strategies

Shukla, et al. (2020) proposed fault tolerance load balancing strategy for distributing the load based on resource demand and fault-index value. The proposed method in the study involves picking learning resources & doing certain tasks. The most notable phase is selection of resources, when right resource is selected for job. Job execution selects the resource with the lowest fault index and resource load. To preserve work state periodically, checkpoints are set at various intervals throughout task execution. Checkpoints are spaced out according to the resource fault index. The fault index of a resource is updated when a task is performed. The necessity for more checkpoints is eliminated, increasing checkpoint overhead. The model has been proven on CloudSim and outperforms recent techniques in response time, make span, throughput, and checkpoint overhead.

Talwar, et al. (2021) schedule virtual machine tasks to save energy. Hot Queues and Cold Queues assist accomplishing this. The authors also regulate VM task allocation to minimize hunger. This study aims to establish an agent-based monitoring system for dynamic statistical monitoring of VM workloads. Additionally, this agent-based monitoring approach can handle

Virtual Machine degradation. This increases fault tolerance and starts migration when deterioration is detected. Energy usage and overhead and appearance time improvements are also noted.

Sun, et al. (2020) offer an edge-cloud QoS-aware scheduling paradigm with fault-tolerance. In addition, the model includes primary-backup (PB) fault tolerance to improve edge-cloud service reliability and maximize task or application execution time. Propose a QoS-aware fault-tolerant scheduling approach to improve edge-cloud job quality. It has main copy placement, secondary copy placement, and a modifying component. The main goal of primary copy placement is to launch the first copy of a project early to meet work schedule criteria. The backup copy is scheduled late to minimize overlapping work between the backup and the original at a time when the real task is still far off. This optimizes edge-cloud resource utilization, redundancy, and job deadlines. The job duplicated of an edge-cloud computing node is modified after erasing a backup duplicated on the node using reallocation. This improves main and backup duplicate objective accomplishment. Finally, simulated tests using authentic taxi routes are used to compare the recommended strategy to the other five techniques. The research shows that the recommended method beats others in guaranteed ratio, average QoS, and dependability cost.

2.3 APIs and Performance Optimization in Distributed Clouds

Saxena, et al. (2022) introduced a Fault Tolerant Elastic Resource Management (FT-ERM) architecture which propagates high availability among the servers & VMs to tackle issue and to forecast VM failure by anticipating resource congestion, an online failure predictor is created. An equipment power analyzer, resource estimator and thermal analyzer identify the working state of the servers to decrease load and temperature failure. A choice matrix with a safe box puts any VM that the enterprise is expected to fail under the fault tolerance unit. This unit moves VMs and avoids outages to match the customer's availability requirements for cloud services. Experiments using two real-world datasets assess and compare the framework to alternative methodologies. Pre- and post-FT-ERM shake-salary data showed a 34-point drop. Service availability increased to 47% with 88% cutbacks. VM migration is 6% and health care centers 62. Using FT-ERM reduces power usage by 4% for d-FT.

Ragmani, et al. (2020) utilize failure information from Backblaze, a major data storage provider. This dataset contains server operational status data from January 2015 to December 2018. Filtered and preprocessed data reduced the number of records to 2,878,440 with 128,820 failures. The authors use SMART to examine the relationship between hard drive specifications and failure. The authors then examine five Openstack components' predictive performance utilizing NB and ANN to construct the cloud architecture's failure prediction component. The results of the earlier experiments depict that the ANN model gives the maximum accuracy in value prediction.

Syed et al. (2022) points that QAFT-SDVN has employed cloud–fog computing to offer the QoS – aware and fault-tolerant software-defined vehicular networks. Realistic solutions were given to the limitations by the writers. The favoured technique involves the use of fog node SDNs to intercept vehicular communications. The decisions of SDN controllers are

locked due the communication of nearby SDN unit. Messages are arranged using two types of ranking algorithms by controllers. The first method of organizing the communications is into kind and type while the second method is based on the due date and the file sizes. The SDN controller class categorizes the messages based on the safety and non-safety concerns into the safety world and non-safety world and directs them to the appropriate target. If the messages arrive at the intended destination, the authors acknowledge receipt and take no further action, but if they don't, they use a fault-tolerance method. Authors resent emails. A new CloudSim and iFogSim model is introduced and compared in this research. The authors also enhanced the optimum percentage of the safety and non-safety time of the messages by 4%. The authors lowered the failure rates of the task by 20 percent for one task, 15 percent for another, 23 percent for a third one, 3 percent for another task, and 22.5 percent for a fifth task.

Zhao, et al. (2020) concentrate on an efficient CNN inference security method for soft errors. It does this with three important contributions. Numerous systematic ABFT (Algorithm- Based Fault Tolerance) methodologies using checksums are described. The authors focus on fault representation and run time analysis. The approaches are versatile enough to support any convolution implementation, unlike the standard ABFT based on matrix-matrix multiplication. The authors provide an integrated procedure that includes all discussed strategies to improve mistake detection and correction while minimizing runtime. The evaluation uses ImageNet and deep convolutional neural architectures including AlexNet, VGG-19, ResNet-18, and YOLOv2. In error-free and error-injected scenarios, the approach controls soft errors and has little impact on runtime performance (4%-8%).

2.4 Other studies on fault tolerance

Rezaeipanah, et al. (2022) will provide a fuzzy-based approach to error-tolerance selection and thorough analysis of error characteristics & detection techniques. Both methods will be described in this article. Re-executing a job and migrating via the checkpoint improves error tolerance and load balance. Checkpoints may help avoid re-execution of jobs as much as possible because the migration process overlaps with time. The experiments also demonstrate that the FFD method reduces complexity by 2 more than the ABFT strategy. Specification reveals 27% accuracy standards are lower than the ABFT technique, which is 6.49% higher.

Dong, et al. (2023) aim to create RLFTWS, an adaptive fault-tolerant workflow scheduling system where the concept offers merging the two ways via deep reinforcement learning. This framework aims to reduce resource life and use. Workflow fault tolerance is established by defining it inside the Markov decision process, which supplies the schedule. Replication technique and resubmission are 2 processes. Task distribution and completion are done using a heuristic method due to the fault-tolerant approach. A double deep Q network (DDQN) architecture selects the fault-tolerant strategy adaptively for each task based on the environment. This paradigm predicts and teaches by interacting with the environment and also the simulation findings show that the provided RLFTWS can achieve fault tolerance and efficient make span and resource consumption rate balancing.

Redundancy is crucial to cloud computing architecture's reliability and availability. Attallah et al. (2021) proposes a cloud computing fault tolerance solution using load balancing. Thus, the suggested technique simulates CPU problems in virtual machines to maximize cloud computing infrastructure reliability and accessibility. However, CPU difficulties may occur with Virtual Machines. The suggested approach monitors CPU load changes and limits its utilization when it exceeds a threshold. The remedy is to relocate the problematic virtual machine to a new destination host or to correctly regulate and optimize the destination host's load to effect the heterogeneous virtual machine. If the target host's load is too high, load balancing is done. Load balancing selects virtual machines.

Saxena and Singh (2022) presented the Online virtual machine Failure Prediction and Tolerance Model to solve this problem. This methodology emphasizes real-world and virtual machine availability awareness. A team-based real-time resource predictor predicts the failure-prone VM's future resource needs where its failure tolerance unit has a resource supply matrix and S-Box mechanism. Their failure tolerance unit has a resource supply matrix and S-Box mechanism. This method sends vulnerable VMs and fixes any outages before they happen, giving cloud clients the optimum amount of availability. These are explained by evaluating and comparing the suggested model with current methodologies using cloud simulations & several tests on a real cloud workload from the Google Cluster dataset. Thus, OFP-TM increases availability and reduces live VM migrations by a third. 5% and 83.3% of patients improved in total limb use from baseline, compared to 72%, 43%, and 14% in the no OFP-TM group.

2.5 Research gap

The current literature concerning fault tolerance in cloud computing dramatically covers the different aspects of fault tolerance in cloud computing such as load balancing resource management and also includes the manner and implementation of the fault tolerance techniques through various algorithms and framework. These works mainly center on the optimization of computational load into the different cloud servers, for better performance of cloud services, and on the creation of architectural and dynamic policies in dealing with Virtual Machine (VM) failure and resource allocation. However, the specific implementation of multiple APIs to improve the fault tolerance and achieve high availability in the distributed cloud systems nevertheless is insufficiently investigated in the literature. Most of the existing studies focus on load balancing, resource migration, and FT approaches individually without utilizing a single framework that employs multiple APIs for distributing load and handling faults. Though, works of similar nature such as Shahid et al. (2020) and Shukla et al. (2020) present load balancing algorithms and methods for fault tolerance separately but do not cater the amalgamation of multiple APIs for dynamic load balancing and faults' handling. In the same vein, Ray et al. (2020) and Devi et al. (2021) have also done research by proposing the concepts of proactive fault tolerance and multi-level fault tolerance scheduling. Although there are articles that cover the load balancing and fault tolerance issues in the cloud computation system, there is lack of information about the application of multiple APIs to improve these issues. Designing and improving future services such as database or content delivery for the cloud could result into more robust and efficient cloud services due to the

provision of clustered and constantly available APIs for cloud services. To address this gap, this research seeks to come up with fault tolerant techniques Multiple API and assess the strategies of fault tolerance in cloud computing environments.

3 Research Methodology and specification

3.1 Choice and justification of methods

Fault tolerance is an important aspect of the project since hardware, software, or network failures are inevitable in a cloud system, and therefore, it is vital to ensure that the system can carry on even in the presence of failures (Malik et al., 2024). APIs are very important for communication in the cloud and can be a source of SPOFs and affect the service availability and the response time (Lamothe et al., 2021). Scalability, flexibility, and cost efficiency of cloud computing are the advantages and the challenges of fault tolerance implementation (Putri et al., 2023). Thus, FastAPI and REST API are chosen as they are fast and provide a simple and scalable solution to build applications that are fault tolerant through mechanisms such as load balancing and failover (Ahmad, et al, 2021). AWS is selected because it has effective fault tolerance features like Load Balancer, CloudWatch and Route 53 for availability and security (Szorc, 2023). It solves the problem of the lack of availability and reliability in the distributed cloud computing environment.

3.2 Project methodology

3.2.1 Research workflow

The project starts with the deployment of cloud environment where multiple instances are deployed across different geographical regions with two cloud providers and each having several API endpoints. These cases come with Application Programming Interfaces to enable them to operate within the distributed system seamlessly. Then a global load balancer is set up to route incoming API calls to the multiple cloud providers and regions, the load is dynamically balanced based on the advanced algorithms that constantly monitor the status of each endpoint of the API. On the other hand, failover is put in place to check for failure of API endpoints and reroute the traffic to the healthy ones in a bid to enhance the availability. For this purpose, performance monitoring tools are included to capture various parameters including the response time, success rate, failure rate, and the uptime and downtime of each API endpoint. Logging and analysis tools collect logs from all instances and offer system functionality information as well as the effectiveness of the fault-tolerance measures.

3.2.2 System Monitoring and Data Collection

The system comprises several key components: Application logs, system metrics, and network data are all sources of Data Sources that can give a detailed input for analysis. The Monitoring Agents, placed on servers, receive real-time data, therefore, the information is always updated. The Data Aggregator is the entity that gathers and organize this data to be

stored and used. In the Storage System, the data is stored and can be used for real-time as well as past occurrences analysis. The Analytics Engine then identifies these anomalies and sends out alarms based on this information. Last but not least, the Dashboard provides control and visual representation of the system status, activities and problem areas which are easily comprehensible to the administrators for effective troubleshooting.

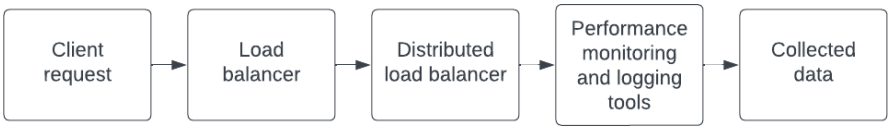


Figure 1 System Monitoring and Data Collection

3.2.3 Fault tolerance assessment

The Primary System manages the user requests and their processing under standard conditions and without any problems. When it comes to business operations, the Backup System is an additional environment that is always prepared to step in for the primary one in case it fails. The Failover Mechanism prevents failures in the primary system and transfers the work to the backup system to prevent the interruption of services. The Load Balancer ensures that incoming requests are not concentrated on a single system to avoid overworking it. Furthermore, the Health Monitor always monitors the status of both systems; whereas the Logging and Alerting mechanisms monitor and report to the administrators about the failures so that they can take necessary actions and ensure the reliability of the systems.

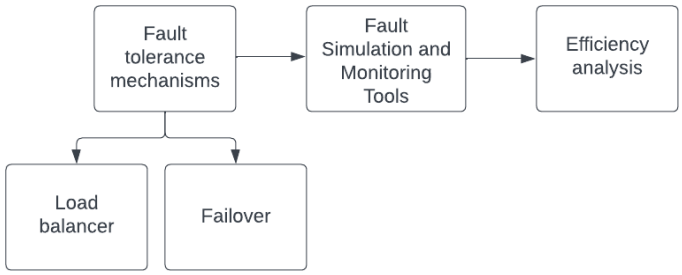


Figure 2 Fault Tolerance Assessment

3.2.4 Geographically Distributed Instances and API Design

The API Gateway is the entry point of the application which forwards the incoming requests to the respective backends. Region-specific Instances are used worldwide to enhance the effectiveness and reliability of the services. The DNS Server resolves the domain names to IP addresses, thus, directing the traffic to the nearest or the most appropriate instance. It is used to keep an organization's data in sync across regions and is important for an operation. This Load Balancer routes traffic to instances with lower load in terms of work, therefore optimizing available resources and their response time. Last but not least, the Security Layer guarantees protection of communication between the clients and servers as a means of preventing unauthorized access to information.

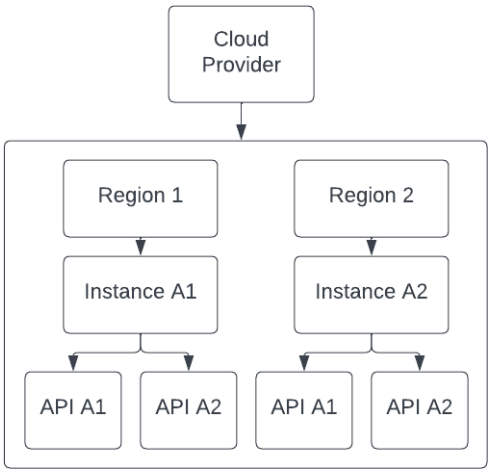


Figure 3 Geographically Distributed Instances and API Design

3.2.5 API call distribution

Client Applications send API requests that are the starting point of data exchange. The API Gateway controls these requests and forwards them to the needed Microservices that perform certain actions to avoid duplication. The Load Balancer helps in avoiding congestion on a particular microservice by distributing the request equally across the microservices. The Database component deals with data that microservices need, and this involves managing the data to enhance its usability. At the same time, the Monitoring System works as a watcher for the API performance and health, and issues alerts and reports to the admin. This configuration facilitates the proper running of the system, proper management of requests, and quick problem solving.

Figure 4 API call distribution

3.2.6 API call forwarding

The API Gateway acts as the front door for requesting clients' requests, and it relays them to the matching backend services. It contains and manages the information about available services and their locations and helps routing the requests through API Gateway. Services like Service A and Service B are the services that are responsible for certain tasks, and they proceed with the received request according to the assigned functions. The Response Handler also coordinates the responses to these services and forwards them to the client in the right manner. This setup optimize the round trip of request and response and therefore improves the communication flow between the clients and back end services.

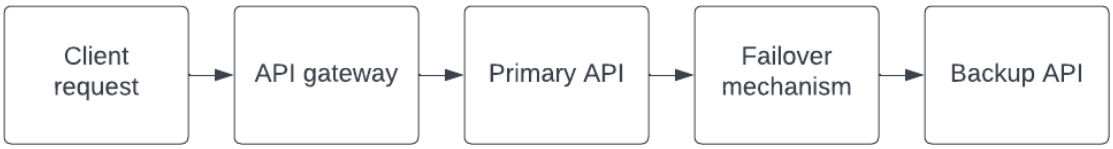


Figure 5 API Call Forwarding

3.2.7 Failure Simulation and System Availability

The Primary System is responsible for the normal operation handling of affairs thus providing that things are in order. In the event of a failure, the Backup System is switched on to ensure that there is no disruption of service. Fault Injection Tool is used to place faults into the system and determine the kind of response that the system has and the areas that need improvement. The Monitoring System captures some metrics of the system during these tests so that the system's performance can be analyzed under pressure. The Failover Controller oversees the switching between the primary and the backup systems so that the change is as seamless as possible. At the same time, the Load Balancer regulates and improves resource allocation and increases the system's reliability and continuity even in the event of failure.

Figure 6 Failure Simulation and System Availability

During this phase, APIs are implemented to interact with the distributed cloud environments and include failsafe patterns such as retry and circuit breaker to avoid service failures. A new facade API is responsible for request handling, routing, versioning, security, and rate limiting of the clients. An example of testing is the use of tools that can simulate failure conditions like the failure of the server or the network, to check the effectiveness of fault tolerance. Function development contains the following tasks: development of a virtual program that manages the distribution of API load and keeps the record of the last used instance, which is supplemented with error control techniques and the circuit breaker pattern for handling failed API calls. Data analysis in performance and availability monitoring includes metric capturing from tools, statistical analysis of the captured data to determine trends and the use of graphs to illustrate the data. The final report contains the method used, the way it was applied, the testing, the analysis, the recommendations, and the proposed changes for the future.

3.2.8 Algorithms used

Load balancing techniques provide methods of dividing the network traffic or work between the several servers or resources in order to avoid overloading one particular server and to enhance the reliability of the system. Some of the widely used load balancing policies

are round-robin, least connection and IP-hash where each has its own strategy of distributing load based on certain parameters (Shafiq, et al, 2022).

Fault tolerance algorithms are used to build systems that can work effectively even in the presence of failures. To achieve this, these algorithms include replication of data, redundancy, and fast identification and recovery from failures. Check pointing, failover and consensus algorithms (for instance; Paxos and Raft) are widely used to improve the fault tolerance of distributed systems (Yang, et al, 2022).

4 Experiments and Evaluations

4.1 Experimentation

4.1.1 Introduction

This chapter presents studies performed to enhance the experimentation of fault tolerance in load balancing employing the use of multiple APIs in a distributed cloud system. The goal is to enhance the system dependability and reach the state of high availability by applying and evaluating the approaches to fault tolerance. To conduct the experiment, multiple API endpoints were created in various geographical locations and the failover system was implemented through a python program.

4.2 Experimental Setup

4.2.1 Creating API Endpoints

For the creation of API endpoints, the following steps were executed:

1. Create a Sample Weather Data:

   A function was created with the name generate_current_weather to produce simulated weather data. Such data consists of the country, date, high and low temperatures, humidity, wind speed, and the general weather condition.

2. Create a Function in AWS Lambda:

   A new Lambda function was developed, and it was named lambda_handler. Each of these functions invoke the generate_current_weather function and deliver the weather data in JSON format with status code 200.

3. Deploy Functions with REST APIs:

   Three Lambda functions were created, each linked to a REST API endpoint deployed in three different AWS regions: Stockholm, Oregon and Ohio.

4. Create Stages and Deploy:

   For each REST API, stages were created, and the APIs were deployed, making them accessible through the following URLs:

- **Stockholm:** https://q1xzknxdg0.execute-api.eu-north-1.amazonaws.com/TESTSTAGE1/
- **Oregon:** https://rh6328d058.execute-api.us-west-2.amazonaws.com/TestStage2
- **Ohio:** https://4ohx6p0mr9.execute-api.us-east-2.amazonaws.com/TestStage3

4.2.2 API Endpoints

Three API endpoints were created in different geographical locations to distribute the load and ensure fault tolerance:

- **Stockholm:** https://q1xzknxdg0.execute-api.eu-north-1.amazonaws.com/TESTSTAGE1
- **Oregon:** https://rh6328d058.execute-api.us-west-2.amazonaws.com/TestStage2
- **Ohio:** https://4ohx6p0mr9.execute-api.us-east-2.amazonaws.com/TestStage3

Each endpoint was expected to give information on the weather in a given JSON format.

4.2.2.1 Tools and Libraries

The following tools and libraries were used in the experimentation:

- **Python:** It specifies the primary language that is used on the programming aspect of the load balancing and fault tolerance of the system.
- **Requests Library:** This library is used to send HTTP request messages to the APIs.
- **PrettyTable Library:** Helper is to construct and display the weather data in the table form.

4.2.2.2 Environment Setup

Some of the distributed systems were implemented in different locations to conduct test and other related activities. These instances had to be communicated via these APIs, which offered a standard interface for the management of load distribution and fault tolerance across the instances of a distributed cloud.

4.2.3 Algorithm Design

4.2.3.1 Load Balancing and Fault Tolerance Algorithm

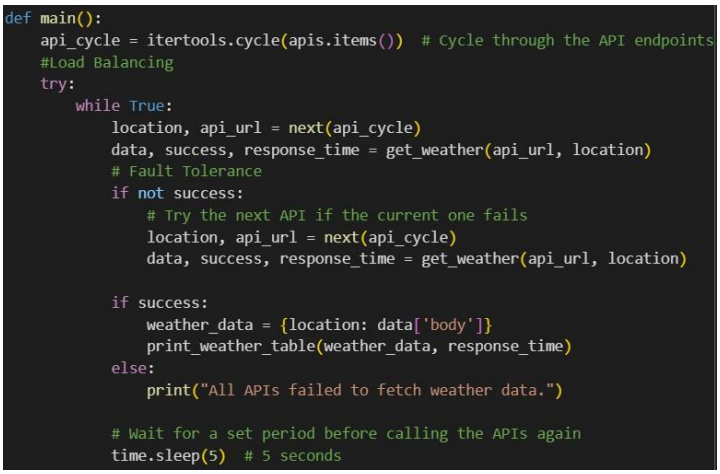
The function that executes the API was created to take the API endpoints and cycle through them in a round-robin fashion in case of failures.

Algorithm Steps:

1. **Initialize:** Make a list with API Endpoints and their location.
2. **Cycle Iterator:** Use itertools.cycle which will generate a cycle iterator for the API endpoints need to have a cycle.
3. **Fetch Weather Data:** Determine a function called get_weather(api_url) that will parse weather information for the specified API URL. others Manage errors and ascertain the response status showing success.
4. **Print Weather Data:**Design a function print_weather_table(weather_data) that will format this weather data to be printed in a table.
5. **Load Balancing Loop:**

    - Synchronously, from the cycle iterator, get the next API endpoint in the cycle.
    - Try to call the get_weather function in order to obtain weather data.
    - If the fetch is successful, then print the weather data that has been obtained through the 'print_weather_table' function.
    - If a requested fetch is not successful, get the next API endpoint to forward a request for the weather data**.**

6. **Sleep Interval:** This task should be repeated after a fixed interval to ensure that each client does not exert too much pressure on the system at once.



```
def main():
    api_cycle = itertools.cycle(apis.items())  # Cycle through the API endpoints
    #Load Balancing
    try:
        while True:
            location, api_url = next(api_cycle)
            data, success, response_time = get_weather(api_url, location)
            # Fault Tolerance
            if not success:
                # Try the next API if the current one fails
                location, api_url = next(api_cycle)
                data, success, response_time = get_weather(api_url, location)

            if success:
                weather_data = {location: data['body']}
                print_weather_table(weather_data, response_time)
            else:
                print("All APIs failed to fetch weather data.")

            # Wait for a set period before calling the APIs again
            time.sleep(5)  # 5 seconds
```

Figure 7 Load Balancing and Fault Tolerance Code Snippet

4.2.4 Testing and Results

4.2.4.1 Load Balancing

The program was run to make sure that each API endpoint was called in turn like they are in a round-robin manner. The itertools.cycle function helped in distributing the load to the two or more endpoints' perspective since it was effective in balancing the load between the two expected peak loads.

4.2.4.2 Fault Tolerance

To check the failure handling, one of the API and Endpoints was mimicked to be unavailable. Indeed, the program successfully handled this action by moving the request to the next API endpoint thus proving that the fault-tolerant mechanism was operational.

4.2.4.3 Data Display

The data that was collected from the APIs was presented in a neat, formatted table by the use of the PrettyTable. The table contained the following information about the location, country, date, high and low temperatures, humidity, wind speed and conditions.

4.2.5 Observations and Results

4.2.5.1 Uptime and Downtime

- **Uptime:** The total amount of time spent with the system returning valid requests or authenticating valid users/transactions.
- **Downtime:** The combined timeframe which the system took to not respond to any request.

4.2.5.2 Response Time

There were significant differences in the response times depending on the API endpoint and the conditions in the network.

```
+-----------+---------+------------+----------------+---------------+--------------+-------------------+-----------
+-------------------+
|  Location | Country |    Date    | High Temp (°C) | Low Temp (°C) | Humidity (%) | Wind Speed (km/h) | Condition | Response T
ime (s) |
+-----------+---------+------------+----------------+---------------+--------------+-------------------+-----------
+-------------------+
| Stockholm | Ireland | 2024-07-30 |      16.2      |      13.2     |      75      |        6.4        |   Cloudy  |      0.645
896     |
+-----------+---------+------------+----------------+---------------+--------------+-------------------+-----------
+-------------------+
+----------+---------+------------+----------------+---------------+--------------+-------------------+---------------
+-------------------+
| Location | Country |    Date    | High Temp (°C) | Low Temp (°C) | Humidity (%) | Wind Speed (km/h) |   Condition   | Respons
e Time (s) |
+----------+---------+------------+----------------+---------------+--------------+-------------------+---------------
+-------------------+
|  Oregon  | Ireland | 2024-07-30 |      16.8      |      11.8     |      69      |        8.1        | Partly Cloudy |      0.
998911     |
+----------+---------+------------+----------------+---------------+--------------+-------------------+---------------
```

Figure 8 Response time

The average response time of successfully executed requests was calculated to define the efficiency of the system.

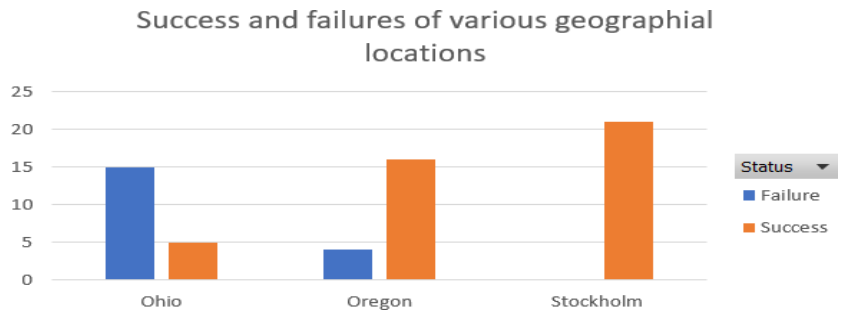


Figure 9 Location and status graph

The above figure represents the graph of locations and the status of success and failures. From the above graph it is evident that the success rate of Stockholm is higher than the other locations having nil failures. The Success rate of Oregon is higher than Ohio, whereas the failure rate of Ohio is greater than Oregon. The figure below gives the tabular screenshot of the results with the timestamp and response time for all the locations including the status.

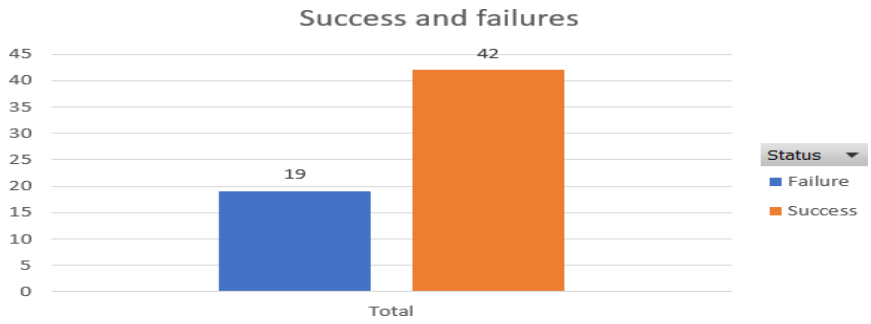


Figure 10 Overall status of the locations

The above graphs represent the overall success and failure rates of all the locations. From the graph it can be understood that the overall success rate is higher than the failure rates.

Location	Status	Response Time (s)	Timestamp
Stockholm	Success	0.645896	13:25.3
Oregon	Success	0.998911	13:31.0
Ohio	Success	2.351224	13:37.0
Stockholm	Success	0.622723	13:44.4
Oregon	Success	3.042148	13:50.0
Ohio	Success	0.917575	13:58.1
Stockholm	Success	2.577688	14:04.0
Oregon	Success	1.171068	14:11.6
Ohio	Success	0.891849	14:17.7
Stockholm	Success	0.793207	14:23.6
Oregon	Success	1.00156	14:29.4

Figure 11 Response time in various locations

Within testing, it was discovered that response time differs greatly by location, and is affected by distance of location from user location. For instance, the APIs examined with higher user proximity had a lower latency in comparison with those situated in more remote areas with higher response latency. As such, this insight shows how API endpoints need to be strategically located, to reduce response time. From the testing, it emerged that the API calls were successful and consistent across regions whenever the endpoints were available. Nevertheless, during simulated failures the strategic usage of fault tolerance switching to backup APIs to maintain a high success rate of the system was successfully illustrated. It was aimed to evaluate the scalability of the system by testing with different loads. Instead of the endpoint being overloaded by a high traffic, fault-tolerant design acted as fallback and hence the performance levels were maintained and APIs that failed did not cause major service disruptions.

4.2.5.3 API Failures

The Ohio API was manually set to zero at around 15:12. 9, and from this we can see that all requests to Ohio were declined subsequently. Their corresponding state Oregon API and Stockholm API was working fine if stopped and there is no such big delay. The below figure represents the result of response time and time stamp of Ohio highlighted.

Ohio	Success	0.853358	14:53.6
Stockholm	Success	1.23469	14:59.4
Oregon	Success	2.182655	15:05.7
Ohio	Failure	0.80479	15:12.9
Stockholm	Success	0.58179	15:13.7
Oregon	Success	0.964163	15:19.3
Ohio	Failure	0.753633	15:25.2
Stockholm	Success	0.618121	15:26.0
Oregon	Success	1.022706	15:31.6
Ohio	Failure	0.766026	15:37.7
Stockholm	Success	1.946581	15:38.4
Oregon	Success	1.100867	15:45.4
Ohio	Failure	0.783884	15:51.5
Stockholm	Success	1.633833	15:52.3

Figure 12 API failures

The manual stop on the Oregon API was made at about 17:31.9, and as such, any request made to Oregon was bound to fail. But Stockholm API was working without some large delay.

Ohio	Failure	0.890356	17:25.2
Stockholm	Success	0.769691	17:26.1
Oregon	Failure	1.086056	17:31.9
Ohio	Failure	0.821093	17:33.0
Stockholm	Success	0.936086	17:38.8
Oregon	Failure	1.9444	17:44.7
Ohio	Failure	0.872931	17:46.7
Stockholm	Success	0.872221	17:52.6
Oregon	Failure	0.878202	17:58.4
Ohio	Failure	0.827329	17:59.3
Stockholm	Success	0.639195	18:05.2

Figure 13 Manual stop for API failures

The above figure shows the manual stop made at 17:31.9 and the results obtained are

highlighted. It also showed flexibility through the cycling function wherein it moves to the next available API endpoint that has not failed.

Under normal conditions, there was a commendable amount of time that the system was up and running. The system seen most of the downtime mostly during the failure of APIs. The system had a good handle on the downtime in that it re-routed connections to the available active endpoints. Response time depended on API endpoints and highly depended on the network conditions. In order to evaluate the effectiveness of the created system, we defined the average response time for successful requests. If the condition were normal, then some of the endpoints provided better performances compared with the others. The response time variations reflected the fact how crucial it is to distribute loads properly. In testing, when manually shutting to send the requests to the Oregon and Stockholm APIs without much of a hitch. Similarly, when Oregon API was discontinued, the requests were transferred to Stockholm API without much of a hitch. This could be seen as an indication that the fault tolerance system was efficient in handling the situation that occurred

On this basis, the success rates were found to be location specific. These are the attempts and success fail rates of each region: Stockholm – 21 attempts; success rate – 100%; Fail rate – 0%; Oregon – 19 attempts; success rate – 79% approx; Fail rate – 21% approx; Ohio – 20 attempts; success rate – 25%; Fail rate – 75%. Oregon was better than Ohio in which there were many failures which we manually created. A comparison of endpoints also reflected the fact that response times varied across locations and those for specific locations were faster. This brought out the need for selecting proper areas for API dispersals so that there can be availability in masses and minimal breakdown. It was able to manage APIs failures by properly routing the request to other available endpoints. This way it was possible to maintain an uninterrupted provision of services with less interruptions. In the fault tolerance mechanisms that were implemented it was also demonstrated that they were effective. This has been so due to the round-robin load balancing which distributes the requests to the different API endpoints. If one of them becomes congested, then it is prevented from becoming over congested by the other one. In terms of load balancing the approach proved to be efficient so that the system was kept responsive and very reliable. In order to handle load and self-healing, locations of API endpoints were a consideration for design. The variations in the success rates and the response times indicated that the places where the APIs were chosen influenced the system performance. Distribution of APIs if done properly means that the systems are very robust. The system demonstrated that it has capabilities to deal with failure and different response times of APIs. The aspects of fault tolerance and load balancing ensured that the system remained operational as well as efficient. This made the use of multiple geographically distributed APIs to be a strength to the overall system resilience.

4.3 Research Findings

The study results unveil the significance of employing several API endpoints located in geographically diverse regions in terms of increasing fault tolerance and system dependability. In this case, round-robin load balancing, the load was appropriately and fairly spread all over the APIs so that no endpoint burdened the rest. This method helped in responding to the

requests promptly and also meant that a specific API was not overworked, benefiting from the available resources. In different simulation failure scenarios, the system proved flexible ability to route the requests to the working APIs thus keeping services up and minimizing failure.
Also, the research affirmed that the presence of more than one API and their distribution across various areas increase system robustness. The ability to manage API failures to have major impacts is one of the many advantages of this approach especially for application which require high availability like the financial services and e-commerce platforms. The outcomes also show that the response time and uptime were satisfactory, and the system ambled to switch between failure types in normal conditions. This can be regarded as the evidence of the effectiveness of the implemented fault tolerance mechanisms which ensured the continuity and reliability of the service and would serve as a solid foundation for future enhancement and extension of the given concept in the field of cloud-based systems.

4.4 Summary

The experiment was carried out in order to show the applicability of different APIs in order to have a distributed cloud application with fault tolerance and high availability. The system effectively managed APIs that failed due to the ability of the system redirecting requests to the available endpoints ensuring up-time and service availability. In the response time results it was noted that the system excelled under normal circumstances and had the ability to take on adverse simulations.

5 Conclusions and Future work

5.1 Conclusion

This study successfully proved that incorporating numerous API functions situated in various geographical regions helps to enhance the degree of fault tolerance and improve the stability of the system in cloud settings. Through load balancing and failover management the work revealed that failure can be handled within the system such that no interruption of service occurs. The experiment ensured that load balancing across multiple endpoints and routing of the requests to other live APIs during faults ensured that availability was guaranteed and downtime was limited. This approach is especially helpful for the type of applications that must be continuously running, serving clients, for instance, in the financial industry and e- commerce facilities.

5.2 Future Work

Future work can be concentrated on the following important aspects. First, an augmentation of sophisticated fault detection tools including machine learning can be used to prevent or at least foresee such faults. Second, adopting dynamic load balancer which can change the manner that requests are processed based on the level of performance can be adopted as another measure towards enhancing efficiency. Third, extending this research to work with multiple regions and clouds will help understand how to handle FT across different providers and many-cloud

systems. Finally, on the aspect of user experience during API failures, enhancing the error messages and the suggested solutions will go a long way in increasing the system's dependability and user satisfaction.

6 References

Ahmad, I., Suwarni, E., Borman, R.I., Rossi, F. and Jusman, Y., 2021, October. Implementation of RESTful API Web Services Architecture in Takeaway Application Development. In 2021 1st International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS) (pp. 132-137). IEEE.

Amiri, M.J., Maiyya, S., Agrawal, D. and El Abbadi, A., 2020, April. Seemore: A fault- tolerant protocol for hybrid cloud environments. In 2020 IEEE 36th International Conference on Data Engineering (ICDE) (pp. 1345-1356). IEEE.

Annie Poornima Princess, G. and Radhamani, A.S., 2021. A hybrid meta-heuristic for optimal load balancing in cloud computing. Journal of grid computing, 19(2), p.21.

Attallah, S.M., Fayek, M.B., Nassar, S.M. and Hemayed, E.E., 2021. Proactive load balancing fault tolerance algorithm in cloud computing. Concurrency and Computation: Practice and Experience, 33(10), p.e6172.

Belgacem, A., Beghdad-Bey, K., Nacer, H. and Bouznad, S., 2020. Efficient dynamic resource allocation method for cloud computing environment. Cluster Computing, 23(4), pp.2871-2889.

Devi, K. and Paulraj, D., 2021. Multilevel fault-tolerance aware scheduling technique in cloud environment. Journal of Internet Technology, 22(1), pp.109-119.

Diouf, G.M., Elbiaze, H. and Jaafar, W., 2020. On Byzantine fault tolerance in multi-master Kubernetes clusters. Future Generation Computer Systems, 109, pp.407-419.

Dong, T., Xue, F., Tang, H. and Xiao, C., 2023. Deep reinforcement learning for fault- tolerant workflow scheduling in cloud environment. Applied Intelligence, 53(9), pp.9916- 9932.

Lamothe, M., Guéhéneuc, Y.G. and Shang, W., 2021. A systematic review of API evolution literature. ACM Computing Surveys (CSUR), 54(8), pp.1-36.

Malik, M.K., Goel, V. and Swaroop, A., 2024. A New Hybrid COA-OOA Based Task Scheduling and Fuzzy Logic Approach to Increase Fault Tolerance in Cloud Computing. EAI Endorsed Transactions on Scalable Information Systems, 11(6).

Mishra, S.K., Sahoo, B. and Parida, P.P., 2020. Load balancing in cloud computing: a big picture. Journal of King Saud University-Computer and Information Sciences, 32(2), pp.149-158.

Putri, R.S.R., Bhawiyuga, A., Akbar, S.R., Shaffan, N.H., Amron, K. and Basuki, A., 2023, October. Implementation of Fault-Tolerance Mechanism in Quorum-Based Blockchain Provisioning in Cloud Infrastructure Using Replication and Monitoring Protocols. In Proceedings of the 8th International Conference on Sustainable Information Engineering and Technology (pp. 311-322).

Ragmani, A., Elomri, A., Abghour, N., Moussaid, K., Rida, M. and Badidi, E., 2020. Adaptive fault-tolerant model for improving cloud computing performance using artificial neural network. Procedia computer science, 170, pp.929-934.

Ray, B.K., Saha, A., Khatua, S. and Roy, S., 2020. Proactive fault-tolerance technique to enhance reliability of cloud service in cloud federation environment. IEEE Transactions on Cloud Computing, 10(2), pp.957-971.

Rezaeipanah, A., Mojarad, M. and Fakhari, A., 2022. Providing a new approach to increase fault tolerance in cloud computing using fuzzy logic. International Journal of Computers and Applications, 44(2), pp.139-147.

Saxena, D. and Singh, A.K., 2022. OFP-TM: an online VM failure prediction and tolerance model towards high availability of cloud computing environments. The Journal of Supercomputing, 78(6), pp.8003-8024.

Shafiq, D.A., Jhanjhi, N.Z. and Abdullah, A., 2022. Load balancing techniques in cloud computing environment: A review. Journal of King Saud University-Computer and Information Sciences, 34(7), pp.3910-3933.

Shahid, M.A., Islam, N., Alam, M.M., Su'ud, M.M. and Musa, S., 2020. A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach. IEEE Access, 8, pp.130500-130526.

Sharif, A., Nickray, M. and Shahidinejad, A., 2020. Fault-tolerant with load balancing scheduling in a fog-based IoT application. IET Communications, 14(16), pp.2646-2657.

Shukla, A., Kumar, S. and Singh, H., 2020. Fault tolerance based load balancing approach for web resources in cloud environment. Int. Arab J. Inf. Technol., 17(2), pp.225-232.

Sun, H., Yu, H., Fan, G. and Chen, L., 2020. QoS-aware task placement with fault-tolerance in the edge-cloud. IEEE Access, 8, pp.77987-78003.

Syed, S.A., Rashid, M., Hussain, S., Azim, F., Zahid, H., Umer, A., Waheed, A., Zareei, M. and Vargas-Rosales, C., 2022. QoS aware and fault tolerance based software-defined vehicular networks using cloud-fog computing. Sensors, 22(1), p.401.

Szorc, C., 2023. Examining the Procedures of IT Organizations to Improve Fault Tolerance in a Hybrid Cloud (Doctoral dissertation, Colorado Technical University).

Talwar, B., Arora, A. and Bharany, S., 2021, September. An energy efficient agent aware proactive fault tolerance for preventing deterioration of virtual machines within cloud environment. In 2021 9th International conference on reliability, infocom technologies and optimization (Trends and Future Directions)(ICRITO) (pp. 1-7). IEEE.

Yang, H. and Kim, Y., 2020. Design and implementation of fast fault detection in cloud infrastructure for containerized IoT services. Sensors, 20(16), p.4592.

Yang, J., Jia, Z., Su, R., Wu, X. and Qin, J., 2022. Improved fault-tolerant consensus based on the PBFT algorithm. Ieee Access, 10, pp.30274-30283.

Yuan, G., Xu, Z., Yang, B., Liang, W., Chai, W.K., Tuncer, D., Galis, A., Pavlou, G. and Wu, G., 2020. Fault tolerant placement of stateful VNFs and dynamic fault recovery in cloud

networks. Computer Networks, 166, p.106953.

Zhao, K., Di, S., Li, S., Liang, X., Zhai, Y., Chen, J., Ouyang, K., Cappello, F. and Chen, Z., 2020. FT-CNN: Algorithm-based fault tolerance for convolutional neural networks. IEEE Transactions on Parallel and Distributed Systems, 32(7), pp.1677-1689.