

DESIGNING FAULT-TOLERANT ARCHITECTURES FOR RESILIENT ORACLE CLOUD ERP AND HCM INTEGRATIONS

Anusha Atluri

SR Oracle Techno Functional consultant at Oracle, USA

ABSTRACT

A lot of firms are now embracing cloud-based tools like Oracle Cloud ERP (Enterprise Resource Planning) and HCM (Human Capital Management) to get their most important job done. These systems are particularly crucial for a business's growth and stability because they take care of everything, including people and money. These tools are useful, but it can be challenging to keep them all up to date and working together when something goes wrong. This article talks about how to build systems that can handle problems so that Oracle Cloud ERP and HCM systems stay robust even when things go wrong. You can't stop every possible failure, but you can create systems that can fix themselves quickly and have the least impact when things go wrong. We talk about practical methods that IT teams can get things done in the real world. You could wish to employ microservices that aren't connected, event-driven workflows that change as events happen, and proactive monitoring systems that find problems before they grow worse to depend less on single points of failure. You need more than simply the right tools. You also need to get people to work together, establish processes that can be utilized over and over again, and make a place where people are always trying new things and getting better. The study talks about what usually goes wrong and what usually goes right based on real-world application, simulations, and evaluations of the design. We also speak about how to make systems that can break down without causing any complications. The system will remain working even if one part breaks. We also discuss about the best ways to try again that will make it less likely that customers will have to become involved. It's evident that the main point is that just because you have power doesn't mean you can't make mistakes. It's about making systems that can deal with surprises and do the correct thing. This article can help IT directors and business architects discover how to make Oracle Cloud connectors more stable. This will help them deal with any challenges that crop up later.

KEYWORDS

Oracle Cloud ERP, HCM, Fault-Tolerant Architecture, Integration Resilience, Cloud Architecture, Middleware Redundancy, Data Synchronization, High Availability, Disaster Recovery, Microservices

1. INTRODUCTION

1.1. Context and Background

Businesses use digital tools like HCM (Human Capital Management) and ERP (Enterprise Resource Planning) to keep things running smoothly. ERP systems help businesses with many key tasks, such as managing projects, paying bills, procuring goods, and the supply chain. HCM systems, on the other hand, are in charge of everything that happens to an employee, such as hiring them, training them, paying them, and keeping track of how well they do their duties. These technologies aren't only for major companies that work in fast-paced industries with a lot of

restrictions. They are also incredibly helpful tools that help things go smoothly every day and keep you on track with your long-term goals.

Oracle Cloud is now one of the best places to go for HCM and ERP solutions. It offers a single, scalable, and secure cloud architecture that makes it easy for organizations to run all over the world. The architecture has distinct sections, but they all work nicely together. It can observe data as it happens, gets updates all the time, and follows rules rather well. These are all good reasons for firms to adopt this to keep their back-office systems up to date. By connecting Oracle Cloud ERP with HCM, organizations can keep their financial and employee data in sync, make better decisions, and run their operations more easily on a broad scale.

1.2. Challenges in Integration

Even while there are evident benefits, it's not straightforward to combine Oracle Cloud ERP with HCM, especially with other business systems. These systems make and consume a lot of information for a lot of different business tasks. You might need to report payroll changes (HCM) very quickly in cost accounting (ERP), and information about purchases could modify how the workforce is planned. This vital link makes it possible for complicated data flows to move between systems, departments, and even across countries at times.

The worst things that can happen with these kinds of connections are system crashes, delayed data transfers, and programs that are connected having different data. You could break the regulations, lose money, or lose faith in your workers if there is a small outage or data error. It's considerably tougher to keep APIs, middleware, and third-party programs in sync because some integration need to be in sync in real time or near to real time. Changing regulations, multi-cloud environments, and hybrid IT infrastructures make these problems worse by making the attack surface wider and more likely to fail.

1.3. Need for Fault-Tolerant Architectures

It is critically crucial to make sure that the organization can stay functioning throughout these challenging times. Businesses need to make sure their connections are set up so that they can handle problems. This means that things can still go wrong and things can still happen. A fault-tolerant design can withstand shocks, get back on its feet quickly, and keep essential processes going, which makes interruptions less destructive to the organization. By embracing DevOps and fast delivery methods, businesses may make their systems stronger, more reliable, and able to fix themselves. Integration designs that need scripts that are hard-coded or connections that aren't all in one location anymore. Businesses are increasingly needing event-driven architectures, services that aren't too closely linked, and full monitoring and alerting systems to stop problems before they happen and speed up recovery.

1.4. Objectives

This article speaks about the best ways to set up systems that operate with Oracle Cloud ERP and HCM, as well as how to fix problems when they crop up. The major purpose is to give IT architects, business leaders, and integration engineers a method to make systems better without making them slower or less adaptable.

The major goals of this study are to uncover the most prevalent problems that happen when Oracle Cloud ERP and HCM are integrated and see how they effect enterprises.

- To study architectural patterns like decoupling, asynchronous messaging, and redundancy that make systems less likely to fail.
- To examine at tools and services that help make powerful integrations possible, like Oracle Integration Cloud, event brokers, and monitoring platforms.
- To provide a framework or reference model for putting fault-tolerant architecture into action.

The study used a number of different ways to attain these goals. This article talks about modern integration frameworks and architectural approaches, and it also gives real-life examples of firms that have successfully set up long-term Oracle Cloud connections. Also, system design simulations and failure testing scenarios are utilized to check the proposed architectural methodologies. This study builds on what we currently know by looking into how to make enterprise cloud systems that are naturally strong and can grow.

2. CORE CONTENT

2.1. Oracle Cloud ERP and HCM Integration Landscape

Oracle Cloud ERP and HCM are important parts of modern business IT systems. They help manage money, operations, and people in different ways. These systems must be able to work with other platforms in order to keep data consistent, make it easier to get real-time insights, and improve operational efficiency. Oracle has a lot of ways to connect that make it easier and more reliable to share data.

The Oracle Integration Cloud (OIC) is the heart of Oracle integrations. It is a strong platform-as-a-service (PaaS) solution that lets you use prebuilt adapters, orchestrate APIs, and automate business processes. The OIC makes it possible for businesses to use different integration methods based on their needs by allowing both synchronous and asynchronous connections.

Oracle offers a wide range of REST and SOAP APIs that let developers work directly with both ERP and HCM modules. These APIs make it easier to do a number of things, such as getting GL entries and changing payroll records. Companies that want on-premises or hybrid integration solutions still use Oracle SOA Suite.

Standard integration patterns include:

- **Batch Integration:** Used for non-time-critical processes like daily financial reconciliations or bulk payroll uploads.
- **Real-Time Integration:** Vital for operations like employee onboarding, where immediate reflection across systems is necessary.
- **Event-Based Integration:** Leveraging Oracle Business Events and Streaming services, systems can publish and consume events like requisition approvals or timecard submissions for immediate action.

This landscape provides the tools, but fault-tolerant architecture decisions determine whether these integrations withstand real-world disruptions.

2.2. Defining Fault Tolerance in Cloud Integrations

The system will keep running even if one portion of a fault-tolerant cloud integration stops working. It keeps tiny problems from getting worse, including system failures, network problems, or API timeouts.

Key concepts include:

- **Redundancy:** Building in spare capacity (e.g., multiple data centers or servers) to take over when a primary resource fails.
- **Failover:** Seamlessly switching to a standby system or network when the active one fails.
- **Self-Healing Systems:** Architectures designed to detect and resolve issues autonomously, such as restarting services or rerouting traffic without manual intervention.

Fault tolerance in Oracle ERP and HCM ensures that payroll operations keep going even if one API fails, and the procurement process doesn't halt when there are delays at integration points. This is very critical for projects that need to be done quickly or that have a lot of rules to follow. Businesses are finding it difficult and tougher to migrate data. If one node has a problem, such as an API request that doesn't work for cost reimbursement, it could spread to other modules and make reporting, compliance, and employee happiness worse. Not just a nice thing to have, every business needs fault tolerance.

2.3. Key Components of a Fault-Tolerant Design

2.3.1. Redundancy and Load Balancing

Redundancy, which means spreading out copies of services across several zones or regions, is a key part of fault tolerance. In Oracle Cloud Infrastructure (OCI), multi-region deployments make sure that if one area goes down, another can take on the load.

Enterprises may opt for:

- **Active-Active Setups:** All instances handle traffic concurrently. Offers high availability but requires complex synchronization.
- **Active-Passive Setups:** The secondary remains idle until the primary fails. Simpler but may introduce slight recovery delays.

Load balancing across these instances ensures no single system is overwhelmed, enhancing both performance and reliability.

2.3.2. Retry and Circuit Breaker Patterns

It will still be hard to get to the API. Retry logic enables you start the call over again after a certain length of time if there is a problem, such as a timeout. People can't use a service that isn't working because of how the circuit breaker is set up. This keeps systems safe. It finds failures, and when it reaches a certain level, it cuts off the circuit to stop calls for a short time, which helps the downstream service get back up and running. Together, these patterns help avoid cascading failures and improve system robustness.

2.3.3. Decoupled Architecture

A good design has loose coupling. Message brokers like Oracle Messaging, Apache Kafka, or other systems that act as middlemen make it easier to keep data producers and consumers separate. The queue protects communications, so if one system stops working or slows down, the other one keeps going.

Rod, this kind of asynchronous integration makes it easier to expand, makes you less dependent, and gives you more chances to get back on track after a defeat.

2.3.4. Monitoring and Observability

Fault-tolerant systems require **deep visibility**. Tools like **OCI Logging, Oracle Cloud Monitoring, Splunk, and Prometheus** play critical roles in observability.

Key strategies include:

- **Dashboards** showing system health and message throughput.
- **Centralized logs** for forensic analysis.
- **Real-time alerts** for latency spikes, error rates, or dropped messages.

Proactive monitoring helps teams detect and resolve issues before they impact business.

2.3.5. Secure Data and Authentication Redundancy

Security should never be the single point of failure. Ensuring fault tolerance requires strategies like:

- **Token Refresh Mechanisms:** Automatically renewing expired tokens.
- **Credential Rotation:** Seamlessly updating API keys without downtime.
- **SSO Redundancy:** Using federated identity providers that support failover (e.g., Azure AD with Oracle IDCS).

Secure design must go hand-in-hand with availability.

2.4. Practical Implementation Approaches

2.4.1. Middleware Strategies

Oracle Integration Cloud (OIC) serves as the core middleware layer, enabling low-code workflows and system connectors. It supports:

- Custom **REST connectors** for third-party systems.
- **Prebuilt adapters** for Oracle and non-Oracle apps.
- Error-handling flows (e.g., re-routing failed transactions to quarantine queues).

Paired with **custom APIs**, OIC acts as the central nervous system for integration orchestration.

2.4.2. Leveraging OCI Native Services

Oracle Cloud offers several native tools for enhancing fault tolerance:

- **Load Balancers:** Distribute traffic intelligently across backends.

- **Autonomous DB with Data Guard:** Enables high-availability databases with automatic failover.
- **Failover Groups:** Automate redirection of resources during region failures.

These services help maintain service continuity with minimal manual effort.

2.4.3. Disaster Recovery Scenarios

No system is immune to large-scale outages. A well-architected disaster recovery (DR) plan involves:

- **Regular backups** of configuration and transactional data.
- **Cross-zone failover** using OCI's Availability Domains.
- **Playbooks and simulations** to ensure teams know how to execute recovery quickly.

DR readiness should be tested at least quarterly, simulating real-world failure modes.

2.4.4. Hybrid and Multi-Cloud Integrations

Enterprises increasingly operate in **multi-cloud environments**, integrating Oracle Cloud with AWS, Azure, or on-prem systems. Use cases include:

- Integrating Oracle HCM with **Workday** for talent data exchange.
- Bridging ERP financials with **Salesforce** for revenue recognition.
- Syncing procurement data with **SAP Ariba**.

These require robust **API management**, **security policies**, and **network design** to ensure resilience across vendors.

2.5. Case Studies and Use Cases

2.5.1. Enterprise Retailer Deployment

A large store employed Oracle Cloud ERP and warehouse systems together so they could see how much stock they have on hand at any given time. The circuit breaker delivered messages to a queue when the main API stopped working because there was too much traffic. Employees were able to keep working while the data was being synced in the background. This proved that backup plans work to keep things rolling.

2.5.2. Government Sector Implementation

A government agency set up an extra payroll stream using Oracle HCM, OIC, and secure message brokers. The active system moved to a standby zone without any problems during a planned upgrade. They used token rotation and SSO redundancy to keep the security of the system during the change.

2.5.3. Healthcare Integration

A medical system in the area employed Oracle ERP and HCM to handle billing for patients and scheduling staff. They made each site stronger by putting up local data stores. The localized cache kept giving out data even when the main Oracle APIs stopped operating. This mixed

strategy made sure that patient care went well, even when individuals were working in the cloud.

3. FIGURES AND TABLES

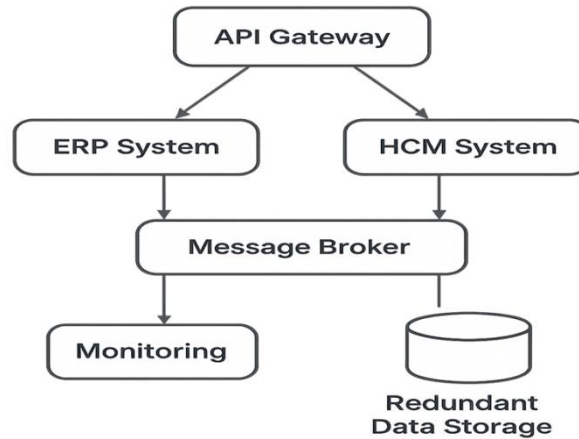


Figure 1: High-Level Fault-Tolerant ERP-HCM Architecture

This graphic displays the essential pieces and how data flows in a fault-tolerant architecture for putting together Oracle Cloud ERP and HCM applications. It is all about being strong, flexible, and easy to switch over.

1. Oracle Cloud Infrastructure (Top Layer)

The key place to host important *services is Oracle Cloud*. It has the security, scalability, and availability zones that enterprise applications need to be able to withstand failures.

2. API Gateway

The API Gateway gets service requests from both inside and outside the system. It manages traffic, keeps it in check, verifies identities, and maintains track of different versions. It also goes around backend services. In this design, the API Gateway sends requests to the Oracle Integration Cloud (OIC). It is more reliable because it has ways to try again and limits on how fast it can go.

3. Oracle Integration Cloud (OIC)

The OIC is the hub of integration, making sure that systems like ERP and HCM can communicate with one other. It makes it easy to send queries to the cloud, create custom APIs, and automate processes. It also comes with adapters for apps and services from Oracle and other firms. It can put transactions in a queue and give transformation logic, which makes it easier to discover and remedy mistakes.

4. Message Queues (Asynchronous Decoupling)

The message queue layer is the most crucial aspect of making sure things don't go wrong. It usually uses something like Oracle Messaging, Kafka, or something else. It makes it easier to chat to each other while you're not online. Messages are kept safe until the target system is ready, so data isn't lost when services go down.

5. ERP and HCM Systems

OIC and queues deliver messages to the ERP and HCM systems, which then do what they need to do with them. The architecture keeps the upstream and downstream systems separate, which makes it less likely that issues in the upstream systems would spread to the downstream systems. When each part can work on its own, it's easier to find and correct faults, and the system is more modular.

6. Monitoring and Observability

A different layer of monitoring collects logs, metrics, and events from different parts of the system, such as the API Gateway, OIC, queues, and application layers. Tools like OCI Monitoring, Splunk, and Prometheus give you instant access to information, let you know about problems before they happen, and help you understand out what's causing them.

7. Autonomous Database & Failover Layer

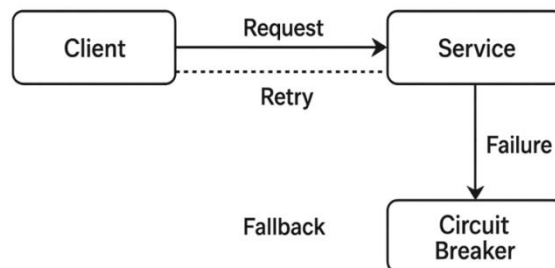
Oracle Autonomous Database keeps data safe for a long time. It was built to fix itself and always be there. Failover logic makes it easy to transition to a backup area if a region or service goes down. This is achievable because of multi-region deployments. The integration, storage, and computing layers make it easier to switch to a backup path.

8. Multi-Region Support

Oracle's multi-region features make it easier to keep backups in more than one place. This makes sure that all of these things are possible: recovering from a disaster, following the regulations, and keeping operations running during long outages.

Key Benefits Demonstrated in the Architecture

- **Resilience to Partial Failures:** Queues and circuit-breakers buffer and reroute traffic during service interruptions.
- **Decoupling & Modularity:** Each service can evolve, scale, or fail independently without collapsing the entire integration.
- **Observability:** Real-time monitoring enables quick detection and remediation of issues.
- **High Availability & Disaster Recovery:** Multi-region setup and database failover mechanisms guarantee minimal downtime.



Figures 2: Message Flow with Retry and Circuit Breaker.

This picture demonstrates a message processing mechanism that can handle problems and stop them from happening. It has a way to break the circuit and start afresh. This is something that

happens all the time when you integrate Oracle Cloud ERP and HCM apps, for example. The process makes sure that service is always available, even when there are problems or protracted outages.

1. Client to Service Request

A client, such Oracle Integration Cloud or another microservice, makes a request to a downstream service, like the HCM payroll endpoint or the Oracle ERP API. You need to do this first. Sometimes, it's hard to obtain this service because of network problems, traffic surges, or problems with the backend.

2. Retry Logic

Should the first request fail due to a timeout or transitory problem, the system initiates a retry process. This logic often utilizes exponential backoff and limited retry efforts to allow the target service to recover before concluding that it is unreachable. This is crucial for transient network or API gateway issues.

- **Advantages:** Reduces false alarms for brief outages.
- **Configuration Parameters:** Max retries, delay intervals, exponential backoff.

3. Circuit Breaker

If retries consistently fail, the circuit breaker pattern is activated. This method protects the system by halting additional requests to the defective service for a designated cooling-off period.

- **Open State:** The circuit "breaks" after a failure threshold is met, redirecting calls to a fallback path.
- **Half-Open State:** The system occasionally tests the downstream service to see if it has recovered.
- **Closed State:** Normal operation resumes when stability is detected.

Circuit breakers **prevent system-wide cascading failures**, protect dependent services, and allow time for graceful recovery.

4. Fallback to Message Queue

In case of persistent failure, the system **routes messages to a fallback mechanism**—typically a **message queue** like Oracle Messaging, Kafka, or OCI Queue. This ensures that:

- No messages are lost.
- Processing can resume once the service is back online.
- Downstream systems are not overwhelmed.

The message queue acts as a durable buffer and enables asynchronous retry later, preserving data and maintaining integration flow integrity.

Use Case Relevance

This pattern is especially critical for Oracle ERP-HCM integrations where:

- **Real-time payroll or finance operations** must not fail silently.

- API endpoints may experience throttling or maintenance windows.
- Data integrity and sequence must be preserved even during failure windows.

Key Benefits Highlighted by This Architecture

- **Improved resilience** through controlled retries.
- **Failure isolation** with circuit breakers to avoid cascading disruptions.
- **Data durability** ensured via message queuing fallback.
- **Enhanced observability** as circuit states and retries can be monitored.

Table 1: Comparison of Fault Tolerance Techniques

Technique	Purpose	Reliability	Latency Impact	Implementation Complexity	Use Cases
Retry Logic	Automatically reattempt failed operations	Medium	Low to Medium	Low	Transient API failures, network glitches
Circuit Breaker	Prevent repeated failures from overwhelming systems	High	Low	Medium	Downstream system outages, rate limit breaches
Redundancy (Multi-Region)	Ensure service continuity during outages	Very High	Low to Medium	High	Disaster recovery, active-passive or active-active setups
Message Queuing	Buffer and decouple system communications	High	Medium	Medium to High	Asynchronous data transfer, order processing, event triggers
Failover Mechanism	Redirect traffic to standby system upon failure	High	Medium	High	Regional outages, system upgrades, maintenance windows
Monitoring & Alerts	Detect and notify system anomalies early	Medium to High	None	Medium	Performance degradation, failure detection, security alerts
Token Refresh / Credential Rotation	Maintain secure access without downtime	Medium	None	Low	Authentication continuity, compliance readiness

Table 2: Downtime Impact Analysis for Oracle Integrations

Integration Type	Primary Function	Estimated Downtime Tolerance	Impact Severity	Business Consequences
Payroll Integration	Synchronizes employee time, benefits, and pay data	≤ 1 hour	Critical	Delayed salary processing, regulatory non-compliance, employee dissatisfaction
Procurement Integration	Automates purchase orders and supplier invoicing	4–6 hours	High	Halted supply chain operations, delayed vendor payments, inventory stockouts
Finance/GL Integration	Aggregates transactions and journal entries	12–24 hours	Medium	Inaccurate reporting, delayed audits, potential SOX non-compliance
Talent Management (HCM)	Candidate on boarding and employee lifecycle data	12–24 hours	Medium	Onboarding delays, HR operational bottlenecks, hiring slowdowns
Expense Management	Reimbursement and approval workflow syncing	24–48 hours	Low	Delayed reimbursements, employee frustration
Real-Time Alerts & Events	Operational triggers for approvals/notifications	≤ 15 minutes	Critical	Missed escalations, delayed approvals, disruption to workflows
Cross-Module Reporting	Unified dashboards across ERP and HCM	12–24 hours	Medium	Stale analytics, decision-making based on outdated data

4. CONCLUSION

It's not only a good idea to have a sound integration architecture in today's business climate; it's a must. Oracle Cloud ERP and HCM make it easier to do important tasks like recruiting and firing people, paying them, establishing budgets, and buying items. The risk goes up a lot when these systems are linked. If one service goes down, it could slow down work, damage other departments, or cost a lot of money to fix compliance problems. This is why it is so important to have architecture that can tolerate problems. You need to have the correct attitude and plan out the design if you want to develop a system that can manage problems, as this study shows. Redundancy, message queuing, retry systems, and circuits breakers all help things run smoothly and stop little problems from getting worse. These patterns make the architecture harder to deal with, but the extra work is worth it. Fault tolerance allows you respond straight away, increases trust in the system's stability, and protects a business's ability to service its employees, customers, and partners. Fault-tolerant systems don't just fix things when they break; they also aim to keep them from breaking in the first place. Parts that mend themselves correct themselves when something goes wrong. Tools for monitoring and observability help teams find problems quickly. These skills are especially important now because downtime costs are increasing up and digital operations have to always meet or beat performance standards. There is optimism for the future of integration that can handle faults. Finding problems before they grow worse is becoming more and more important with the help of artificial intelligence and machine learning. Oracle's better tools will let developers construct integrations that are faster, easier, and more flexible. It will work with event-driven structures and serverless processes. The capacity to find and fix problems in real time will be what makes a business strong as systems become more dynamic. In brief, adding fault tolerance to Oracle Cloud ERP and HCM connections is a way to invest in long-term

business stability, flexibility, and continuity. Businesses that use these principles now will be far better able to handle issues in the future.

REFERENCES

- [1] Tandon, Righa, Ajay Verma, and P. K. Gupta. "Fault tolerant and reliable resource optimization model for cloud." *Reliable and Intelligent Optimization in Multi-Layered Cloud Computing Architectures*. Auerbach Publications, 2024. 67-101.
- [2] Kumari, Priti, and Parmeet Kaur. "A survey of fault tolerance in cloud computing." *Journal of King Saud University-Computer and Information Sciences* 33.10 (2021): 1159-1176.
- [3] Pasham, Sai Dikshit. "Fault-Tolerant Distributed Computing for Real-Time Applications in Critical Systems." *The Computertech* (2020): 1-29.
- [4] Cheraghlou, Mehdi Nazari, Ahmad Khadem-Zadeh, and Majid Haghparast. "A survey of fault tolerance architecture in cloud computing." *Journal of Network and Computer Applications* 61 (2016): 81-92.
- [5] Amin, Zeeshan, Harshpreet Singh, and Nisha Sethi. "Review on fault tolerance techniques in cloud computing." *International Journal of Computer Applications* 116.18 (2015): 11-17.
- [6] Kirti, Medha, Ashish Kumar Maurya, and Rama Shankar Yadav. "Fault-tolerance approaches for distributed and cloud computing environments: A systematic review, taxonomy and future directions." *Concurrency and Computation: Practice and Experience* 36.13 (2024): e8081.
- [7] Mukweho, Mukosi Abraham, and Turgay Celik. "Toward a smart cloud: A review of fault-tolerance methods in cloud systems." *IEEE Transactions on Services Computing* 14.2 (2018): 589-605.
- [8] Shahid, Muhammad Asim, et al. "Towards Resilient Method: An exhaustive survey of fault tolerance methods in the cloud computing environment." *Computer Science Review* 40 (2021): 100398.
- [9] Liu, Ameen Alkasemand Hongwei. "Research article a survey of fault-tolerance in cloud computing: Concepts and practice." *Research Journal of Applied Sciences, Engineering and Technology* 11.12 (2015): 1365-1377.
- [10] Ledmi, Abdeldjalil, Hakim Bendjenna, and Sofiane MounineHemam. "Fault tolerance in distributed systems: A survey." *2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)*. IEEE, 2018.
- [11] Farid, Shahid, and Mariya Hussain. "Fault tolerance techniques in cloud and distributed computing: A review." *Technical Journal* 22.IV (2017).
- [12] Almufti, Saman M., and Subhi RM Zeebaree. "Leveraging Distributed Systems for Fault-Tolerant Cloud Computing: A Review of Strategies and Frameworks." *Academic Journal of Nawroz University* 13.2 (2024): 9-29.
- [13] Poola, Deepak, et al. "A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments." *Software architecture for big data and the cloud* (2017): 285-320.
- [14] Prathiba, Soma, and S. Sowvarnica. "Survey of failures and fault tolerance in cloud." *2017 2nd International Conference on Computing and Communications Technologies (ICCCCT)*. IEEE, 2017.
- [15] Jhavar, Ravi, Vincenzo Piuri, and Marco Santambrogio. "Fault tolerance management in cloud computing: A system-level perspective." *IEEE Systems Journal* 7.2 (2012): 288-297.