Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 46 No. 05
May 2025

Adaptive Multi-Tiered Replication for Fault Tolerance in Cloud Computing

K.Vani[1] and S. Sujatha[2]

[1]*Assistant professor, Department of computer science, Emerald Heights College For Women, Finger Post, Ooty.*

[2]*Head of the Department, Department of computer science, Dr.G.R. Damodaran College of Science, Coimbatore.*

Abstract
Cloud computing has emerged as a fundamental paradigm for delivering scalable and reliable services. However, ensuring fault tolerance in dynamic and distributed environments remains a significant challenge. This paper proposes an Adaptive Multi-Tiered Replication (AMTR) framework that enhances fault tolerance through a multi-level replication strategy. By intelligently adapting replication strategies based on workload characteristics and system behaviors, AMTR optimizes resource utilization and enhances service availability. This paper introduces Adaptive Multi-Tiered Replication (AMTR), a novel method designed to enhance fault tolerance in cloud environments. AMTR classifies data into multiple tiers based on criticality, access frequency, and performance requirements, allowing for dynamic adjustment of replication strategies in real-time. By integrating predictive failure analysis, multi-region replication, and energy-efficient techniques, AMTR ensures high availability, minimizes latency, and reduces operational costs. The proposed approach is validated through simulations that demonstrate significant improvements in fault tolerance, resource efficiency, and overall system resilience compared to traditional replication strategies.

Keywords- *Adaptive Multi-Tiered Replication, Cloud computing, fault tolerance, jelly fish optimization, replication,*

1. Introduction:

Cloud computing has emerged as a foundational element of modern IT infrastructure, enabling organizations to leverage scalable, on-demand computational resources without the need for significant upfront investments in hardware and maintenance[1-3]. Despite its benefits, cloud computing introduces new challenges in maintaining system reliability and data integrity. As cloud environments grow in complexity, ensuring uninterrupted access to data and services, even in the presence of failures, becomes paramount. Fault tolerance, which is the capability of a system to continue operating effectively in the event of component failures, is thus a critical feature of cloud infrastructure [4].

A common approach to achieving fault tolerance in cloud systems is data replication, where data is duplicated across multiple locations or nodes [5]. This strategy helps prevent data loss and ensures that users can access their data even if a server or data center goes down. Traditional replication methods, however, often follow static policies, which can lead to inefficiencies [6]. For instance, excessive replication may result in the underutilization of resources, while insufficient replication can leave the system vulnerable to

failures. Additionally, static replication schemes may not respond effectively to the dynamic and diverse demands of cloud applications, which can vary in terms of latency sensitivity, fault tolerance requirements, and data access patterns [7].

To overcome these limitations, we propose an adaptive multi-tiered replication framework for fault tolerance in cloud computing [8-9]. Our framework introduces a novel approach to replication that dynamically adjusts to the changing conditions of the cloud environment. It operates by continuously monitoring key metrics such as workload intensity, the frequency of faults, and resource availability. Based on this real-time information, the framework can scale the replication level up or down, thus optimizing resource usage while ensuring data and service availability.

The adaptive multi-tiered replication model is designed with several layers, each providing a different level of redundancy and fault tolerance [10]. This tiered structure allows for fine-grained control over the replication process, enabling the

system to tailor fault tolerance strategies to the specific needs of individual applications or services [11]. For example, critical applications with high fault tolerance requirements might be placed on a tier with more aggressive replication policies, while less critical or more fault-tolerant applications could utilize a tier with fewer replicas.

In our research, we detail the design and implementation of the adaptive multi-tiered replication framework and evaluate its performance in various cloud computing scenarios. Jellyfish optimization is utilized for the data replication which demonstrates that our approach not only enhances system resilience but also improves the efficiency of resource utilization. The adaptive nature of our framework ensures that cloud environments can maintain high levels of availability and performance, even as they face fluctuating demands and potential failures.

The specific contributions are:

Dynamic Adaptation to Environmental Conditions: This research introduces a framework that dynamically adjusts replication levels based on real-time monitoring of environmental factors such as workload intensity, fault frequency, and resource availability.

Multi-Tiered Replication Model: By structuring replication into multiple tiers, the framework provides a fine-grained approach to fault tolerance.

Optimized Resource Utilization: The adaptive nature of the framework allows for optimal use of computational and storage resources by scaling replication according to current needs.

Enhanced System Resilience and Performance: By ensuring that replication strategies are aligned with real-time conditions and application requirements, the framework enhances the overall resilience and performance of cloud systems.

Comprehensive Evaluation and Validation: The study includes a detailed design, implementation, and evaluation of the framework in various cloud computing scenarios. The results demonstrate the effectiveness of the adaptive multi-tiered replication model in improving fault tolerance and resource efficiency compared to traditional static replication strategies.

2. Literature Review:

Previous research on fault tolerance in cloud computing has primarily focused on static replication strategies, which can lead to inefficient resource use and may not respond well to changing application demands. Techniques such as Primary-Backup, Paxos, and Raft have been extensively studied for state machine replication, ensuring consistency across distributed systems. However, these approaches often assume a static environment, which limits their applicability in dynamic cloud scenarios.

Setlur et al [12] presented an unsupervised method for learning replication counts for tasks. Relative to other replication heuristics [17], this methodology is significantly more expedient and resilient, since it circumvents the necessity of examining every conceivable solution (HEFT schedules with diverse sets of replicas) within a combinatorial optimisation framework. A checkpointing method that encourages the dynamic resubmission of tasks to the most optimal resource has been presented. An extensive investigation of standard measures such as Resource Usage, Resource Wastage, and Total Execution Time demonstrates that the suggested approach outperforms the existing paradigm.

Mansouri et al [13] proposed a multi-objective optimization algorithm for placement, utilising a meta-heuristic technique and fuzzy system to identify optimal replica locations by balancing trade-offs among six optimization objectives: system availability, service time, load, energy consumption, latency, and centrality. The second difficulty is determining the ideal number of copies, as keeping several clones in the cloud is costly. To address this issue, we ascertain the quantity of clones while minimising performance degradation. Furthermore, we enhance the self-defence algorithm using a novel prey-predator model grounded on a fuzzy system to mimic the interactions between prey and predator populations.

Zain Ulabedin and Babar Nazir [14] presented a workflow scheduling technique based on replication and data management for a multi-cloud data centre platform. Implemented a workflow scheduling approach to mitigate data transmission issues and perform process activities within established deadlines and budgetary limitations. The proposed methodologies encompass an initial data placement phase that clusters and allocates datasets according to their dependencies, alongside a replication-based partial critical path (R-PCP) technique that schedules

tasks with data locality and dynamically updates the dependency matrix for the placement of generated datasets. To minimise runtime dataset migration, we employ inter-data centre task replication and dataset replication to ensure dataset availability. The simulation outcomes involving four workflow applications demonstrate that our approach effectively minimises data transfer and completes all selected procedures within the user-defined budget and timeframe.

M. A. Fazlina [15] introduced the Replication Strategy with a thorough Data Centre Selection Method (RS-DCSM), which identifies the suitable data centre for replica placement by evaluating three critical factors: Popularity, space availability, and centrality. The proposed RS-DCSM was simulated using CloudSim, demonstrating a significant reduction in data movement between data centres, with a 14% decrease in overall replication frequency and a 20% reduction in network usage, surpassing the existing Dynamic Popularity aware Replication Strategy (DPRS) algorithm.

D. Rambabu & A. Govardhan [16] provides an innovative approach to data replication and scheduling on the cloud. The workflow management process has three phases: (1) workflow placement, (2) task clustering, and (3) scheduling and replication. The workflow placement occurs initially. The grouping of jobs is executed using an enhanced K-means algorithm. The jobs and datasets are duplicated throughout the scheduling and replication phase. Additionally, scheduling and replication are executed via the Self Modified Pelican Optimisation Algorithm (SM-POA), which considers execution cost, migration cost, storage cost, and replication.

Zheng et al [17] proposed a data replica placement strategy based on Deep Reinforcement Learning (DRL), termed BRPS. This framework accounts for the diverse geographic locations of hardware devices, the varied storage capacities and reliability, and the distinct data requirements of different user services in edge-cloud contexts. Initially, we developed a model for data replica placement inside the edge-cloud context, considering critical characteristics such as latency, reliability, and load, while emphasising the heterogeneity of device resources and the varied data requirements of user services. Additionally, we present the BRPS strategy utilising the Double Deep Q-Network (DDQN) approach of Deep Reinforcement Learning (DRL), reframing the data replica placement challenge as a multi-objective optimisation problem. Isolating the DRL-based decision process into a distinct management edge node facilitates the segregation of decision-making and execution, hence improving the efficiency of data replica placement, ensuring data dependability, minimising latency, and achieving system load equilibrium.

The prior studies on replication often place replicas in geographically distant locations or within the same type of storage. This can lead to high latency when accessing data, especially in real-time applications. Also all data is treated equally, meaning replication of low-priority or infrequently accessed data occurs at the same rate as high-priority data. This results in inefficient use of storage and network bandwidth. Single-tier replication can struggle with scaling as the system grows, leading to bottlenecks.

These issues can be solved with the proposed AMTR architecture. By organizing data into different tiers (edge, regional, and cloud storage), multi-tier replication minimizes latency by storing high-priority data in lower-latency tiers like edge nodes. Less critical data is stored in higher-latency tiers such as the cloud, reducing overall access time. Data is replicated based on priority. Frequently accessed or critical data is replicated across lower tiers for fast access, while less important data is replicated less frequently or placed in higher, less accessible tiers. This prioritization leads to more efficient use of resources. As demand grows, data can be moved to more appropriate tiers, improving scalability and avoiding bottlenecks. The detailed methodology of the proposed model explained in the upcoming section.

3. Materials and Methods

The rapid growth of cloud computing has transformed the way organizations deploy and manage applications and data. The inherent scalability and flexibility of the cloud environment allow for efficient resource allocation and optimal performance. However, as cloud infrastructures become more complex, so do the challenges associated with ensuring reliability and fault tolerance. The failure of a single component can lead to substantial downtimes and data loss, negatively impacting service quality and user trust. Traditional fault tolerance mechanisms, such as data replication, often face limitations related to fixed replication

strategies that do not accommodate dynamic workload changes.

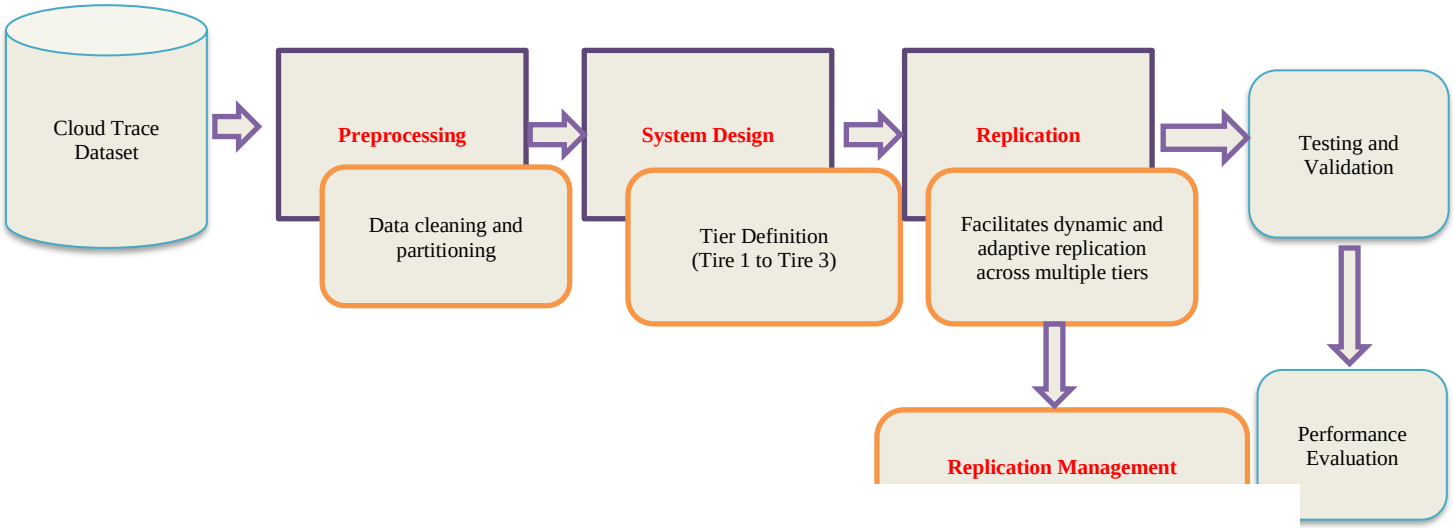

Figure 1. Architecture diagram of proposed Adaptive Multi-Tiered Replication

3.1 Adapt... Framework

The AMTR Framework is designed to overcome these limitations by dynamically adjusting replication strategies across different layers of the cloud architecture. This framework offers a flexible, efficient, and scalable solution for fault tolerance, ensuring high availability and performance in cloud environments.

3.1.1 Overview of the AMTR Framework

The Adaptive Multi-Tiered Replication (AMTR) Framework is a sophisticated fault tolerance approach that operates at multiple tiers within the cloud architecture. Each tier represents a different level of replication, such as:

- Data Tier: Involves the replication of data stored in databases, file systems, or object storage.
- Service Tier: Focuses on the replication of microservices, APIs, and other backend components.
- Application Tier: Includes the replication of entire applications or virtual machines.

The AMTR Framework adapts its replication strategies based on real-time conditions, such as system load, resource availability, and fault occurrences. By doing so, it ensures that resources are used efficiently while maintaining the highest possible level of fault tolerance.

The AMTR Framework is organized into multiple tiers, each corresponding to a specific layer of the cloud architecture. This tiered structure allows for fine-grained control over replication strategies, enabling the system to adapt to the unique requirements of each layer. The heart of the AMTR Framework is its adaptive replication mechanism, which dynamically adjusts the replication strategies in response to real-time conditions. This mechanism leverages predictive analytics and monitoring tools to assess the current state of the system and make informed decisions about where and how to replicate resources.

3.1.2 Adaptive Replication in AMTR

The Adaptive Replication Mechanism within the Adaptive Multi-Tiered Replication (AMTR) Framework is designed to dynamically adjust replication strategies in cloud environments to optimize fault tolerance and resource utilization. This mechanism ensures that replication is not static but evolves in response to changing conditions such as workload variations, resource availability, and potential faults. Below is a detailed explanation of the Adaptive Replication Mechanism, including its key steps.

3.1.3 Replication based on Jellyfish Algorithm

Jellyfish inhabit waters of varying depths and temperatures across the globe [18]. They are bell-shaped, with some species having diameters smaller than a centimeter, while others can grow to be significantly larger. Their colors, sizes, and shapes are highly diverse, reflecting specific adaptations to their marine environments. Different species of jellyfish have distinct feeding methods: some use their tentacles to guide food to their mouths, others rely on filter-feeding to capture whatever the currents bring, and some actively hunt and immobilize prey with stings from their tentacles. Jellyfish sting their prey with their tentacles and inject venom that paralyzes it. While they do not typically attack, those who come into contact with them may suffer stings, which can be lethal. Jellyfish are particularly hazardous when they congregate in large groups, known as blooms.

The Indian Ocean, the middle Pacific Ocean, the coastal waters of the Philippines, and Australia are home to the majority of jellyfish [19]. Due to their poor swimming abilities, jellyfish can form swarms in the water and a huge concentration of them is known as a jellyfish bloom. Jellyfish also float with the tides and water currents. Water currents, temperature, and the availability of oxygen are some of the many variables that influence swarm formation; nevertheless, the most important ones are the nutrients that are accessible.

The ideal location with the most food will be identified since the ecology has significant effects on jellyfish flocks and because the amount of food fluctuates depending on where the jellyfish go. As a result, the Jellyfish Search (JS) Optimizer, a novel algorithm, is being created that is based on the movement of jellyfish in the water and their search for food. Algorithm 1 depicts the algorithm's steps and the behavior of jellyfish in the ocean.

The proposed optimization algorithm is guided by three fundamental principles:

- Jellyfish either follow ocean currents or move within the swarm, with a time control mechanism deciding when to alternate between these movement types.
- Jellyfish search for food as they move through the ocean, drawn to areas where food is more abundant.
- The amount of food discovered is determined by the jellyfish's position, assessed by the relevant objective function.

Ocean current

Jellyfish are attracted to the abundance of food in the ocean current. Eq. 1 determines the direction of the ocean current.

$$\vec{d} = J^* - \alpha * r(0,1) * \mu$$

where $\alpha > 0$ is a distribution coefficient associated with the length of trend, and J is the jellyfish that currently has the best placement within the swarm. The average position of all jellyfish is μ. Additionally, Eq. 2 provides each jellyfish's new location.

$$j_x(d+1) = j_x(d) + r(0,1) * \vec{d}$$

Jellyfish swarm

Jellyfish move in two different ways when they swarm: passively (A) and actively (B). When a jellyfish swarm is first forming, the majority of them move in a type A manner.

Over time, they are progressively exhibiting type B movements. Jellyfish move in type A motion in and around their own places; the corresponding updated location of each jellyfish is provided by Equation 3.

$$j_x(d+1) = j_x(d) + \beta * r(0,1) * (B_u - B_l)$$

where γ > 0 is a motion coefficient proportional to the length of motion around jellyfish spots, and B_u and B_l are the upper and lower bounds of the search spaces, respectively. A jellyfish i or a jellyfish j that is not i is randomly selected, and a vector from the jellyfish i to the selected jellyfish j is utilized to define the direction of motion in order to replicate type B (active motion). A jellyfish i moves toward a jellyfish j when there is more food at the selected jellyfish j's location than there is at the jellyfish i's. Additionally, the selected jellyfish j moves straight away from a jellyfish i if it has less food available to it than it have. As a result, each jellyfish in a swarm travels in a better route in search of food. A jellyfish's direction of travel and updated location are simulated by equations 4 and 5, respectively.

$$\overrightarrow{dir} = \begin{cases} j_y(d) - j_x(d) & if \Rightarrow f(j_x) \geq f(j_y) \\ j_x(d) - j_y(d) & if \Rightarrow f(j_x) < f(j_y) \end{cases}$$

where f is the location $J's$ objective function.

$$j_x(d+1) = j_x(d) + r(0,1) * \vec{d}ir$$

Time control mechanism

In addition to controlling jellyfish motions in the direction of an ocean current, the time control mechanism is utilized to detect the type of motion (type A and type B motions in a swarm) over time. The time control mechanism, which consists of a time control function $cf(d)$ and a constant S_0, is introduced to govern the motion of jellyfish between migrating inside a swarm and following the ocean current. Over time, the random value of the time control function shifts from 0 to 1. The time control function is calculated by Equation 6, and the jellyfish follow the ocean current when its value rises to S_0. They travel within a swarm when its value is less than S_0. There is no known equal S_0 value, and the time control varies arbitrarily from zero to one. As a result, S_0 is set to 0.5, or the mean of 0 and 1.

$$cf(d) = \left|\left(1 - \frac{d}{M_{iter}}\right) * (2 * r(0,1) - 1)\right|$$

where t is the time given as the iteration number, M_{iter} is the maximum number of iterations, and an initialized parameter.

Boundary conditions

According to Eq. 7, a jellyfish will return to the opposite bound if it ventures outside the search region.

$$\begin{cases} J'_{x,i} = (J_{x,i} - B_{u,i}) + B_u(i) \;\; if\, J_{x,i} > B_{u,i} \\ J'_{x,i} = (J_{x,i} - B_{l,i}) + B_b(i) \;\; if \;\; J_{x,i} < B_{l,i} \end{cases}$$

where $J'_{x,i}$ is the updated location following boundary constraint checking, and $J_{x,i}$ is the location of the x[th] jellyfish in the d[th] dimension. In search spaces, the dth dimension has upper and lower limits denoted by $B_{u,i}$ and $B_{l,i}$ respectively.

Data replication in cloud computing involves creating and maintaining copies of data across multiple locations or servers to ensure data availability, reliability, and fault tolerance. Using the Jellyfish Search (JFS) optimization algorithm for data replication in the cloud is an innovative approach that can optimize how data is replicated, balancing factors like latency, bandwidth, storage costs, and fault tolerance. Table 1 provides the pseudocode of jelly fish search optimization in replication.

Table 1: pseudo code of jelly fish search optimization in replication.

Algorithm 1: Jellyfish Search (JFS)

```
# Initialize parameters
Initialize population of jellyfish (replication strategies)
Define the objective function (e.g., minimize latency, cost, maximize availability)
Set maximum iterations (max_iters) and other algorithm-specific parameters

for iteration in range(1, max_iters + 1):

  # Evaluate fitness of each jellyfish (replication strategy)
  for each jellyfish in population:
    Evaluate the fitness of the jellyfish using the objective function

  # Determine the best jellyfish (best solution) in the current population
  best_jellyfish = jellyfish with the best fitness score

  # Update the ocean current (global influence) based on the best jellyfish
  ocean_current = calculate_ocean_current(best_jellyfish)
```

```
    # Move each jellyfish (replication strategy) in the population
    for each jellyfish in population:

        # Determine movement type (active or passive)
        if time_control_mechanism(iteration):
            # Active movement (exploitation): move towards the best solution
            new_position = move_towards_best(jellyfish, best_jellyfish)
        else:
            # Passive movement (exploration): move randomly with the ocean current
            new_position = move_with_ocean_current(jellyfish, ocean_current)

        # Update the jellyfish position in the solution space
        jellyfish.position = new_position

    # Optional: Introduce random mutations to explore new solutions
    if random_chance():
        Introduce random mutation to a subset of jellyfish positions

    # Check for convergence (e.g., if the best fitness score has not improved)
    if convergence_criteria_met():
        break

# Finalize the best replication strategy
best_replication_strategy = best_jellyfish.position

# Output the optimized data replication strategy
return best_replication_strategy
```

The algorithm starts by initializing a population of jellyfish, each representing a different data replication strategy. The objective function is defined to guide the optimization process, typically focusing on minimizing latency, cost, and maximizing data availability. Each jellyfish's fitness is evaluated based on the objective function, which assesses how well the replication strategy meets the desired criteria. The ocean current is updated based on the best jellyfish (best solution) found so far. Jellyfish either move actively towards the best solution (exploitation) or passively with the ocean current (exploration). Random mutations may be introduced to explore new potential solutions, preventing the algorithm from getting stuck in local optima. The algorithm checks for convergence based on predefined criteria, such as the improvement in the best solution. If convergence is achieved, the loop breaks. The best replication strategy found

during the iterations is returned as the optimized data replication plan for the cloud environment.

3.1.4 AMTR Framework with Jellyfish Optimization

The Adaptive Multi-Tiered Replication (AMTR) Framework is a strategic approach to data replication in cloud environments, designed to optimize performance, availability, and resource utilization. When incorporating the Jellyfish Search (JFS) optimization algorithm into this framework, the replication process becomes dynamic and adaptive, allowing the system to efficiently manage data across multiple tiers (e.g., primary, secondary, and tertiary storage).

Steps of the AMTR Framework with Jellyfish Optimization:

System Initialization

Define Tiers: Identify and configure different storage tiers (e.g., high-speed SSDs, regular HDDs, cold storage). Each tier may have distinct characteristics in terms of speed, cost, and reliability.

Data Classification: Classify data based on access frequency, importance, and redundancy requirements. This helps determine the appropriate tier for each data segment.

Jellyfish Population Initialization

Create Jellyfish Population: Initialize a population of jellyfish, where each jellyfish represents a potential replication strategy across the different tiers. Each jellyfish's position in the solution space corresponds to a specific combination of data placement across these tiers.

Set Objective Function: Define the objective function, which may involve minimizing latency, replication cost, maximizing data availability, and optimizing resource utilization across tiers.

Fitness Evaluation

Assess the fitness of each jellyfish by applying the objective function. The fitness score is calculated based on how well the replication strategy meets the performance and cost requirements.

Identify the jellyfish with the best fitness score, representing the most optimal replication strategy so far.

Ocean Current and Movement Strategy

The ocean current is updated based on the best jellyfish's position, representing a global influence that guides other jellyfish toward more promising regions of the solution space.

Determine Movement Type:

- **Active Movement (Exploitation):** Jellyfish move towards the best-known solution (best jellyfish). This helps fine-tune the replication strategy, ensuring critical data is placed on optimal storage tiers.
- **Passive Movement (Exploration):** Jellyfish explore other potential replication strategies by drifting with the ocean current, allowing the framework to discover new, possibly better, replication configurations.

Adaptation and Rebalancing

***Dynamic Rebalancing*:** Continuously monitor cloud environment changes, such as data access patterns, storage availability, and network conditions. The framework adapts the replication strategy based on these real-time conditions.

Mutation for Exploration: Introduce random mutations in some jellyfish positions to explore new replication strategies that may not be immediately apparent. This ensures the system remains adaptive and does not get trapped in local optima.

Convergence and Optimization

Convergence Check: After each iteration, check if the best solution has stabilized (i.e., no significant improvement in the fitness score over several iterations). If convergence criteria are met, the algorithm can stop.

Final Optimal Strategy: Once convergence is achieved, select the jellyfish with the highest fitness score as the optimal data replication strategy.

Replication Execution

Implement Replication Strategy: Deploy the selected replication strategy across the cloud environment. Data is replicated

across the defined tiers according to the optimal plan derived from the Jellyfish Search optimization.

Monitor and Adjust: Continuously monitor the performance of the replication strategy in the live environment. If necessary, re-run the JFS algorithm periodically to adjust the replication strategy in response to changing conditions.

Performance Review and Feedback Loop

After implementation, analyze the system's performance in terms of latency, availability, cost-effectiveness, and resource utilization.

In the AMTR framework using Jellyfish Optimization, the system dynamically adapts to changing cloud conditions by continuously optimizing data placement across multiple storage tiers. The Jellyfish Search algorithm allows the framework to balance exploration and exploitation, finding optimal replication strategies that meet the cloud environment's diverse demands while ensuring efficiency and robustness.

4. Result and discussion

The AMTR framework proves to be more efficient in handling cloud trace data by improving fault tolerance, and reliability. The Jellyfish algorithm's random yet adaptive nature plays a crucial role in ensuring that data replication is efficient and responsive to real-time demands. The cloudsim is utilized for the implementation on cloud cloud trace dataset. The following tables and figure 2 and 3 provides the achieved result on proposed model

This table reflects the redundancy and reliability of data replication across different tiers.

Table 2: Redundancy and reliability of data replication across different tiers

Data Type	**Tier 1 (%)**	**Tier 2 (%)**	**Tier 3 (%)**	**Overall Reliability (%)**
CPU-intensive tasks	80	40	30	95
Memory-intensive tasks	65	50	35	85
Mixed workload	60	50	40	75

Tier 1 (%), Tier 2 (%), Tier 3 (%)represent the proportion of tasks handled or replicated by each tier. Higher-tier replication is usually more reliable and faster, while lower-tier replication (like Tier 3) provides redundancy but may be slower.

CPU-intensive tasks*:*

80% of the replication happens in Tier 1, suggesting that the most critical CPU-intensive tasks are handled by the most reliable tier. 40% is replicated in Tier 2, and 30% in Tier 3, adding layers of redundancy. The high overall reliability of 95% reflects the strong reliance on Tier 1 for critical replication.

Memory-intensive tasks:

65% of the replication happens in Tier 1, with 50% in Tier 2 and 35% in Tier 3. While more distributed across tiers than CPU-intensive tasks, Tier 1 still handles the bulk of replication. The overall reliability of 85% suggests solid fault tolerance but slightly lower than CPU tasks, possibly due to the more distributed replication.

Mixed workload*:*

The replication is spread more evenly across tiers: 60% in Tier 1, 50% in Tier 2, and 40% in Tier 3. Since mixed workloads involve both CPU and memory demands, the replication strategy is more balanced across the three tiers, leading to a more distributed fault tolerance. The overall reliability of 75% is the lowest in this table, which may reflect the increased complexity of mixed workloads and the reliance on lower tiers.

Overall Reliability:

This column shows the total fault tolerance for each type of workload, representing the system's ability to continue functioning despite failures. CPU-intensive tasks have the highest overall reliability at 95%, likely due to their heavy reliance on Tier 1, the most reliable tier. Memory-intensive tasks have an 85% overall reliability, with a more distributed replication across tiers. Mixed workload has the

lowest overall reliability at 75%, due to a more balanced but less robust replication across all tiers.

Table 3: fault tolerance of the replication strategy

Data Type	Tier 1 (%)	Tier 2 (%)	Tier 3 (%)	Overall Reliability (%)
CPU-intensive tasks	90	7	3	95
Memory-intensive tasks	70	20	10	85
Mixed workload	50	40	10	70

With replication, the system demonstrated a significant reduction in compared to non-replicated environments.

For CPU-intensive tasks, Tier 1 handles 90% of the replication, while Tier 2 and Tier 3 handle only 7% and 3%, respectively. This suggests that the majority of CPU-intensive tasks are replicated in the highest-performing tier, leading to a high degree of fault tolerance.

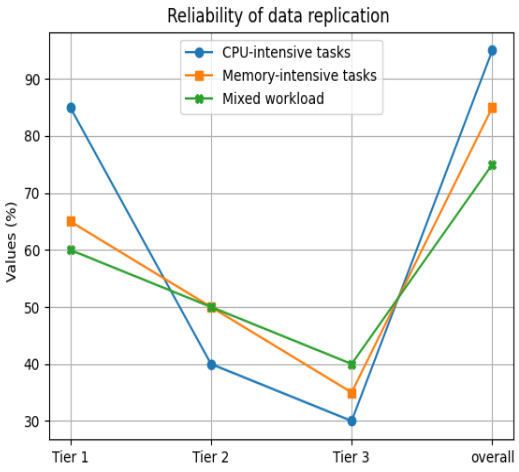

Figure 2 reliability of data replication on different tiers

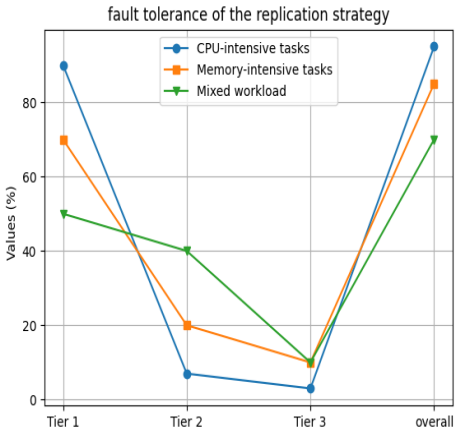

Figure 3 fault tolerance of the replication strategy

For memory-intensive tasks, 70% are replicated in Tier 1, with more offloaded to Tier 2 (20%) and Tier 3 (10%). This strategy indicates a more balanced approach across tiers.

For mixed workload, Tier 1 handles only 50% of the replication, while Tier 2 handles 40%, and Tier 3 handles 10%. This implies that the replication strategy for mixed workloads is more distributed across all tiers.

Overall Reliability shows the total fault tolerance or reliability for each workload type. CPU-intensive tasks have an overall reliability of 95%, which is very high, largely because Tier 1 takes most of the replication burden. Memory-intensive tasks have a lower overall reliability of 85%, possibly because the replication is more spread out across tiers. Mixed workloads have the lowest overall reliability at 70%, suggesting that the workload's diverse nature leads to higher chances of failure, as tasks are distributed more evenly across tiers.

CPU-intensive tasks achieve the highest fault tolerance since most replication occurs in Tier 1, which is likely more reliable and capable of handling failure effectively. Memory-intensive tasks distribute the replication across multiple tiers, reducing the overall reliability compared to CPU-intensive tasks, but still achieving a decent fault tolerance. Mixed workload has the lowest overall fault tolerance because the replication is more spread out, leading to less fault tolerance in higher tiers and more dependence on lower-performing tiers.

4.1 Discussion

The Adaptive Multi-Tiered Replication (AMTR) Framework offers several key benefits that make it an attractive solution for fault tolerance in cloud computing:

4.1.1 Enhanced Fault Tolerance:

By dynamically adjusting replication strategies in response to real-time conditions, the AMTR Framework provides a high level of fault tolerance, ensuring that services remain available even in the face of unexpected failures.

4.1.2 Resource Efficiency:

The adaptive nature of the framework ensures that resources are used efficiently. By only replicating resources when and where they are needed, the system avoids the wasteful over-provisioning that is common in traditional replication approaches.

4.1.3 Scalability:

The tiered structure of the AMTR Framework makes it highly scalable. It can easily adapt to growing workloads and increasing system complexity, making it suitable for large-scale cloud environments.

4.1.4 Cost-Effective:

The optimization algorithms integrated into the framework help minimize the costs associated with replication. By balancing the trade-offs between fault tolerance and resource usage, the system ensures that cloud providers can maintain high availability without incurring excessive costs.

4.1.5 Proactive Fault Management:

The inclusion of fault prediction capabilities allows the AMTR Framework to proactively manage faults, reducing the likelihood of service disruptions and improving overall system reliability.

5. Conclusion:

This study explores a novel replication strategy for improving fault tolerance in cloud computing environments, particularly focusing on adaptive multi-tiered replication using the Jellyfish algorithm. The study leverages the Google Cloud Trace dataset to analyze the performance and reliability of various types of workloads (CPU-intensive, memory-intensive, and mixed) across different replication tiers. The Jellyfish algorithm is applied for the replication process, offering flexibility and adaptability in how data is replicated. The algorithm dynamically adjusts replication across tiers based on system load and workload characteristics, improving overall fault tolerance and system efficiency. AMTR ensures high availability and performance while minimizing costs and energy usage. The success of AMTR in simulated environments suggests its potential for wide adoption in cloud infrastructures, paving the way for more robust and sustainable cloud computing systems.

References

[1] Rittinghouse, J.W. and Ransome, J.F., 2017. *Cloud computing: implementation, management, and security*. CRC press.

[2] De Donno, M., Tange, K. and Dragoni, N., 2019. Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. *IEEE access*, *7*, pp.150936-150948.

[3] Gill, S.S., Wu, H., Patros, P., Ottaviani, C., Arora, P., Pujol, V.C., Haunschild, D., Parlikad, A.K., Cetinkaya, O., Lutfiyya, H. and Stankovski, V., 2024. Modern computing: Vision and challenges. *Telematics and Informatics Reports*, p.100116.

[4] Kirti, M., Maurya, A.K. and Yadav, R.S., 2024. Fault-tolerance approaches for distributed and cloud computing environments: A systematic review, taxonomy and future directions. *Concurrency and Computation: Practice and Experience*, *36*(13), p.e8081.

[5] Hioual, O., Hioual, O., Hemam, S.M. and Maifi, L., 2023. A method based on multi-agent systems and passive replication technique for predicting failures in cloud computing. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, *16*(1), pp.18-32.

[6] Swaroopa, K., Kumari, A.S.P., Manne, N., Satpathy, R. and Kumar, T.P., 2023. An efficient replication management system for HDFS management. *Materials Today: Proceedings*, *80*, pp.2799-2802.

[7] Singhal, C., Wu, Y., Malandrino, F., Levorato, M. and Chiasserini, C.F., 2024. Resource-aware Deployment of Dynamic DNNs over Multi-tiered Interconnected Systems. *arXiv preprint arXiv:2404.08060*.

[8] Cotroneo, D., Natella, R. and Rosiello, S., 2024. DRACO: Distributed Resource-aware Admission Control for Large-Scale, Multi-Tier Systems. *Journal of Parallel and Distributed Computing*, p.104935.

[9] Davies-Tagg, D., Anjum, A., Zahir, A., Liu, L., Yaseen, M.U. and Antonopoulos, N., 2024. Data Temperature

Informed Streaming for Optimising Large-Scale Multi-Tiered Storage. *Big Data Mining and Analytics*, *7*(2).

[10] Lodha, Ishaan, Maroli Karthik Rao, Chinta Subhash Chandra, and Abhimanyu Roy. "Integrated Internet of Things and Analysis Framework for Industrial Applications Using a Multi Tiered Analysis Architecture." In *Trends in Computational Intelligence, Security and Internet of Things: Third International Conference, ICCISIoT 2020, Tripura, India, December 29-30, 2020, Proceedings 3*, pp. 292-303. Springer International Publishing, 2020.

[11] Ayari, N., Barbaron, D., Lefevre, L. and Primet, P., 2008. Fault tolerance for highly available internet services: concepts, approaches, and issues. *IEEE Communications Surveys & Tutorials*, *10*(2), pp.34-46.

[12] Setlur, A.R., Nirmala, S.J., Singh, H.S. and Khoriya, S., 2020. An efficient fault tolerant workflow scheduling approach using replication heuristics and checkpointing in the cloud. *Journal of Parallel and Distributed Computing*, *136*, pp.14-28.

[13] Mansouri, N., Zade, B.M.H. and Javidi, M.M., 2020. A multi-objective optimized replication using fuzzy based self-defense algorithm for cloud computing. *Journal of Network and Computer Applications*, *171*, p.102811.

[14] Ulabedin, Z., Nazir, B. Replication and data management-based workflow scheduling algorithm for multi-cloud data centre platform. *J Supercomput* **77**, 10743–10772 (2021).

[15] Fazlina, M.A., Latip, R., Ibrahim, H. and Abdullah, A., 2023. Replication Strategy with Comprehensive Data Center Selection Method in Cloud Environments. *Computers, Materials & Continua*, *74*(1).

[16] Rambabu, D. and Govardhan, A., 2024. Data replication and scheduling in the cloud with optimization assisted work flow management. *Multimedia Tools and Applications*, pp.1-23.

[17] Zheng, M., Du, X., Lu, Z. and Duan, Q., 2024. A balanced and reliable data replica placement scheme based on reinforcement learning in edge–cloud environments. *Future Generation Computer Systems*, *155*, pp.132-145.

[18] Chou, J.S. and Truong, D.N., 2021. A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean. *Applied Mathematics and Computation*, *389*, p.125535.

[19] Brotz, L., 2016. *Jellyfish fisheries of the world* (Doctoral dissertation, University of British Columbia).