Received: September 19, 2023. Revised: November 4, 2023.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Utilizing Probability Distribution for Selecting Optimal and Minimal Replicas to Achieving Fault Tolerance in a Distributed System

Mahdi S. Almhanna[1]* **Ahmed M. Al-Salih**[1] **Tariq A. Murshedi**[1] **Rafah M. Almuttairi**[2]

[1]*Department of Information Networks, College of Information Technology, University of Babylon, Babylon, Iraq*
[2]*Collage of Information Technology Engineering, Al-Zahraa University for Women, 56001, Karbala, Iraq*
* Corresponding author's Email: mahdi.almhanna@uobabylon.edu.iq

Abstract: This paper highlights the importance of efficient task distribution and robust fault tolerance in network systems. It emphasizes the limitations of relying on a fixed resource quantity and proposes task replication as a solution to improve data availability. The introduced algorithm dynamically determines the optimal number of replicas based on network history, response time, and joint probability of successful servers, aiming to minimize task failure rates. The algorithm's advancements in grid scheduling lie in optimal resource management, fault-aware job placement, adaptability to changing conditions, and efficient fault tolerance through redundancy planning. The algorithm outperforms three other algorithms, showcasing significant enhancements.

Keywords: Grid computing, Replication, Distributed systems cloud computing.

1. Introduction

The utilization of computing allows organizations to integrate geographically dispersed resources from different administrative regions into a unified system. This consolidation facilitates the resolution of large-scale problems across scientific, human, and social domains [1-5]. These resources include diverse components like computers, storage, peripherals, and applications. To cater to the heterogeneous and dynamically changing nature of network resources, a middleware layer needs to be implemented to provide essential services to users.

Network computing environments are susceptible to various failures and outages, primarily due to the abnormal characteristics of the network infrastructure. Failures can manifest in different ways [6, 7], including computer failures, connection issues, network bottlenecks, excessive power consumption, software malfunctions caused by workload, potential software deletions, and other factors arising from the diversity of networks and applications. So, ensuring fault tolerance is essential for maintaining uninterrupted network operation and delivering reliable services.

In essence, reliable applications within the network should be capable of automatically mitigating failures with minimal losses, while maintaining performance and quality of service (QoS) [8-10]. In simpler terms, the network should have the ability to minimize and overcome failures, ensuring uninterrupted functionality. Fault tolerance in the network empowers it to continue operating even in the presence of significant errors or failures, without disrupting overall functionality.

Furthermore, handling failures can be achieved through scheduling strategies in resource scheduling. When done prior to scheduling resources, it is called proactive orientation; otherwise, it is termed post-active [11, 12]. The post-active approach is relatively easier to implement as it utilizes job monitoring techniques, while the proactive method works with probabilities and requires more information about network resources. If the proactive method, such as replication, is used, all failure-handling decisions must be made before the task begins, thereby reducing the probability of failure and increasing

Table 1. Examples of fault tolerance

System	Fault	Tolerance Method
Distributed	Computer failures	Redundancy and replication
File servers	Connection issues	Load balancing and failover
Cloud	Network bottlenecks	Traffic shaping and prioritization
Database	Excessive power consumption	Power backup and redundancy
Web server	Software malfunctions	Graceful degradation and error handling
Network	Potential software deletions	Continuous monitoring and backups
Cluster	Hardware failures	Automatic failover and fault detection
High-performance computing	Workload-induced software malfunctions	Dynamic resource allocation and load balancing
Data storage	Disk failures	RAID (Redundant Array of Independent Disks)
Communication	Packet loss and corruption	Error detection and correction techniques

productivity. Table 1 provides some examples of fault tolerance.

In grid systems, mechanisms for achieving fault tolerance can be classified into three categories [13].

The first category, known as task replication [10], [14], involves duplicating the same task and executing it on multiple independent resources to safeguard against failures at a single point.

The second category is referred to as Checkpoint redundancy [15]. It involves periodically saving the state of the running function to maintain stability. This saved state can be used later in case of any errors, allowing the system to resume execution from the last stored point instead of starting from the beginning.

The final category is Adaptive, which combines checkpointing and symmetrical copies to accomplish the task. The adaptive approach significantly enhances the performance of distributed systems by achieving optimal parameters for both throughput and fault tolerance.

This paper focuses on the first category, task replication, to establish a proactive scheduling system that is resilient to errors. The goal is to determine the minimum number of replicas required for each task.

2. Problem definition:

"In this study, we propose an algorithm that determines the number of replicas based on the specific type of tasks to be performed, rather than using a fixed number for all job offerings. By employing variable replicas, we aim to optimize resource utilization and enhance the overall efficiency of the system."

The problem is about optimizing resource utilization and enhancing system efficiency in a network environment with dynamic and diverse resources by determining the appropriate number of replicas for different types of tasks. The current issue is identified as inefficient resource use due to a fixed or random number of replicas for all tasks, leading to network overload with a minimum number of tasks that need to be executed. The proposed solution involves adapting the number of replicas based on the specific task requirements to achieve optimization and efficiency.

3. Related work

Detecting and predicting faults in a system before they occur is a crucial objective in system design. It involves preventing or avoiding errors that could lead to problems like deadlock and implementing recovery strategies. These strategies can be implemented through replication, improvement techniques, or a combination of both. The probability of failure at a single source is higher compared to failure at multiple sources simultaneously. The need to restart jobs from the beginning can be eliminated by implementing replication. This avoids wasting effort and time in the process, as a failure in one source does not result in a network breakdown, and the system can continue providing services.

During task implementation, the system stores all the relevant information and data at each development stage, creating checkpoints as specific reference points. If a failure occurs, the system can retrieve the stored information and resume work from that checkpoint, saving effort and time. However, there is a potential problem of unintentional or continuous repetition, which can exacerbate the issue.

In this research, the replication mechanism will be employed with a focus on minimizing the utilization of resources in replicas to optimize time and reduce system expenses.

K. Srinivasa, G. Siddesh, and S. Cherian [16] propose a middleware approach in the network where each node holds a copy of the task, and communication between nodes occurs through the TCP/IP protocol.

Received: September 19, 2023. Revised: November 4, 2023.

In [17], J. Abawajy introduced a distributed scheduling algorithm that integrates scheduling and replication features. The approach involves partitioning the network into smaller segments, each comprising a set of sites managed by a dedicated scheduler. These scheduling managers serve as backups for one another. Each user benefits from a set number of replicas. Abawajy's paper considers the possibility of faults and sets a limit on the maximum fraction of faulty processors within a specific cluster. Additionally, the paper assumes the advance identifiability of the suitable number of replicas. The job placement algorithm has a drawback related to potential inefficiency in resource utilization and responsiveness, particularly in the context of resource reservation and site selection. Here are the key drawbacks:

- Resource underutilization and inefficiency:

The algorithm reserves a fixed number of sites for job execution, even if the total available sites (RR) are less than nn. This can lead to underutilization of resources; as potentially suitable sites may remain idle due to the fixed reservation policy.

The approach of reserving n−Rn−R sites that are expected to finish soon may not always result in the best resource allocation, potentially causing idle time for resources that could have been utilized for other tasks.

- Lack of dynamic adaptability:

The algorithm lacks flexibility in dynamically adapting to changing conditions, such as varying resource availability or workload. The fixed reservation of nn sites may not be suitable for dynamic workloads or changing resource conditions.

- Potential latency and responsiveness issues:

The algorithm does not have a mechanism to handle urgent job requests efficiently. If the home SRM cannot find nn candidate sites, it still proceeds with scheduling the job. This may introduce delays or affect the responsiveness of the system, especially for critical or time-sensitive jobs.

- Suboptimal backup selection:

The algorithm designates one of the nn SRMs as a backup home SRM. However, the criteria for selecting this backup SRM are not specified. This lack of defined criteria may lead to suboptimal choices for backup, potentially affecting fault tolerance and system reliability.

M. Chetepen and colleagues [18] introduce a heuristic scheduling approach that utilizes function replication and real-time network state information to rearrange unsuccessful tasks, rather than relying solely on scheduled job data. The algorithm discussed focuses on replicating computational tasks dynamically, aiming to reduce task completion times. It operates in iterations, selecting the longest-waiting task with fewer than a specified number of replicas for distribution. The algorithm considers available resources and replica counts in site selection, aiming for efficient task allocation. A load-based metric is used to assess the workload of a site or resource. Two variations of the algorithm, adaptive task replication (ATR) and failure detection (FD), adapt to changing grid loads and handle resource failures, respectively. A combined approach, FDATR, merges adaptive replication with failed task rescheduling, improving resource utilization and task completion efficiency. The described algorithm for task replication and distribution in grid systems presents several potential drawbacks:

- Static replica counting:

The algorithm employs a predefined fixed number of replicas (LL) for tasks. However, this static approach may not be optimal for varying task requirements or changing system conditions. Task replication needs may vary based on task characteristics, resource availability, or load.

- Resource load metrics:

The algorithm uses a simplistic load calculation based on the number of tasks and million instructions per second (MIPS). This approach may oversimplify the actual load on resources, ignoring other crucial factors like memory usage, I/O operations, or network congestion.

- Limited task information:

The algorithm considers only basic task information such as task processing time and replica count. More comprehensive task information, such as task dependencies, could aid in better scheduling decisions and potentially improve overall system efficiency.

- Potential resource overloading:

The algorithm may select resources based on being "least loaded" without considering the absolute capacity of the resources. This can lead to overloading resources that are already close to their maximum capacity, impacting task performance and system stability.

- Fault resilience:

The algorithm does not explicitly address fault tolerance or recovery strategies. In grid systems where resource failures are common, lacking a robust fault handling mechanism can result in job failures and decreased system reliability.

- Limited adaptability:

The algorithm may struggle to adapt to highly dynamic and bursty workloads. Rapid changes in load conditions can make it challenging for the algorithm to efficiently manage task distribution and replication.

- Performance impact of replication:

While task replication can enhance reliability, it may also introduce additional processing and communication overhead, potentially impacting the overall performance and efficiency of the system.

C. Jiang et al. [19] suggest a fault-tolerant scheduling algorithm that considers the trust level of resources and the security of users. The number of copies is determined based on the variable security level of the network. The algorithm does not explicitly outline a method to determine the number of replicas for fault tolerance. However, it does suggest reserving sites for future placement of replicas, which implies a mechanism to potentially have multiple replicas. The drawback of the Scheduler's operation:

- **Reliance on fault rate alone**: The scheduler primarily relies on the fault rate to determine the most reliable resource. While this is a valuable metric, considering only fault rates may oversimplify the system's reliability. Other factors impacting reliability, like hardware quality, network stability, or software robustness, should also be considered for a more comprehensive assessment of resource reliability.
- **Response time as a secondary factor**: The algorithm considers response time but does not emphasize its importance. Response time is crucial for user satisfaction and meeting job deadlines. A resource with a low fault rate but a significantly higher response time may not always be the best choice, especially for time-sensitive jobs.
- **Assumption of homogeneous resources**: The algorithm assumes that resources are homogeneous in terms of their fault rates. In real-world scenarios, resources can have varying fault rates due to different hardware, software configurations, or network conditions. Failing to consider resource heterogeneity may lead to suboptimal resource allocations.
- **Lack of dynamic adaptability**: The algorithm does not account for dynamic changes in fault rates or resource conditions. A resource's fault rate can change over time due to various factors. An adaptive approach that updates fault rates dynamically would provide a more accurate reflection of resource reliability.
- **Absence of redundancy planning**: The algorithm does not incorporate redundancy planning to mitigate failures. Introducing redundancy or backup solutions for critical jobs can enhance fault tolerance and reduce the impact of resource failures.
- **Scalability challenges**: The approach may face challenges in scaling to a large number of resources and jobs. As the grid scales up, the computational complexity of calculating scheduling indicators and sorting in the SI matrix may increase significantly, potentially affecting the scheduler's efficiency.

4. Proposed work vs. related work

Aspect	**Proposed Algorithm**	**[17]J. Abawajy**	**[18]M. Chetepen et al.**	**[19]C. Jiang et al.**
Purpose	Achieve fault tolerance through resource replication and scheduling optimization	Integrate scheduling and replication, limit faulty processors within clusters	Replicate tasks dynamically, reduce task completion times	Consider trust level and security for fault-tolerant scheduling
Input	Job information, QoS requirements, resource failure history	Job information, QoS requirements, resource failure history	Job information, real-time network state, task replication count	Job information, security level
Output	List of selected resources for each job based on fault rates and response times	List of reserved sites for job execution, backup scheduler assignment	List of replicas and redistributed tasks based on load and replica count	Reserve sites for potential replica placement
Key Drawbacks	Depending on the history information may be a Suboptimal backup selection.	Resource underutilization and inefficiency - Lack of dynamic adaptabil	Static replica counting - Simplistic resource load metrics - Limited task	Reliance on fault rate alone - Response time as a secondary factor - Assumption of

Aspect	Proposed Algorithm	[17]J. Abawajy	[18]M. Chetepen et al.	[19]C. Jiang et al.
		ity - Potential latency and responsiveness issues - Suboptimal backup selection	information - Potential resource overloading - Lack of fault resilience - Limited adaptability - Performance impact of replication	homogeneous resources - Lack of dynamic adaptability - Absence of redundancy planning - Scalability challenges
Consideration of Fault Rate	Yes	Yes	Yes	Yes
Consideration of Response Time	Yes	Yes	Yes	Yes
Adaptability to Workload Changes	Limited adaptability due to fixed reservation policy	Limited adaptability due to fixed reservation policy	Limited adaptability due to static replica counting	Lack of dynamic adaptability
Redundancy Planning	Yes, through replication of tasks and selection of backup resources	Yes, through the assignment of a backup scheduler	Yes, through adaptive replication and failed task rescheduling	Yes, through replica placement
Load Metrics for Resource Allocation	Yes	Yes	Yes	No
Consideration of Resource Heterogeneity	Yes	No	No	No
Scalability Considerations	Scalability challenges due to computational complexity	Scalability challenges due to computational complexity	Scalability challenges due to computational complexity	Scalability challenges due to computational complexity

5. The aim of the proposed work

Due to the dynamic and diverse nature of network resources, the likelihood of failures in the network environment is high. As a result, performing functions in such an environment requires more time [20], leading to network failures. Furthermore, when available resources fail, the system needs additional time to search for alternative resources suitable for executing these tasks.

Many algorithms that rely on symmetrical copies employ a fixed number of identical versions [21]. This approach leads to the inefficient use of resources for the same task, resulting in a network that becomes overloaded with a minimum number of tasks that need to be executed.

In this study, we propose an algorithm that determines the number of replicas based on the specific type of tasks to be performed, rather than using a fixed number for all job offerings. By employing variable replicas, we aim to optimize resource utilization and enhance the overall efficiency of the system.

6. Methods

6.1 Backup resources selection

The resource information server (RIS) is a system or component responsible for storing and managing historical data related to resource loads. It acts as a central repository that keeps track of information such as the usage, availability, and performance of various resources within a network or system. The RIS collects and maintains this data over time, allowing it to be accessed and utilized for various purposes such as resource allocation, load balancing, performance optimization, and decision-making processes. By analyzing the historical resource load data, the RIS enables a better understanding and management of resources, aiding in efficient resource utilization and overall system performance.

The backup resources will be chosen based on the requested number of replicas for the jobs. This ensures that if one resource fails, the network can still finish the job using an alternative resource. The selection of these resources in the discussed research paper relies on factors like response time and resource load. The Resource Information Server

(RIS) is tasked with storing the historical data of resource loads. This data is specifically defined as:

$$LHj = \frac{TCj}{Sj} \tag{1}$$

Where LHj is the load history of resource j, TCj is the instructions acts completed by resource j, and Sj is the rapidity of the resource j measured in seconds for a million instruction operations.

6.2 Scheduling system.

The architectural engineering of the grid scheduler (GS) is designed to accommodate a group of units, which include the following components:

User interface: This interface allows users to submit their tasks to the network. The received tasks are then directed to the available network resources [22-25].

Resource information server (RIS): One of the network resources is the RIS, which collects information about other resources, such as memory size and details about the central processing unit. This information is used by the GS to make informed decisions regarding task scheduling.

Fault handler (FH): The grid scheduler incorporates a fault-tolerant structure by utilizing the Fault Handler component. It handles cases of system failures and ensures reliable operation.

Fig. 1 demonstrates that the scheduling of resources, which are distributed across different geographical regions, is managed by a central management unit. It is important to note that each resource may exhibit different behavior when it comes to failures. Therefore, it is necessary to have a processor capable of handling errors when they occur. If the actual outcome deviates from the expected result, indicating a failure, relevant information about the failure is stored in the resource services (RIS). This information can be leveraged in future task implementations.

7. Proposed system.

The proposed system incorporates proactive scheduling to mitigate failures and minimize their impact. In the event of a failure, the system takes measures to minimize its effects. This is achieved by creating multiple replicas on different resources and executing them simultaneously. As a result, if one resource fails, it does not hinder the execution of tasks on the remaining resources, ensuring uninterrupted progress.

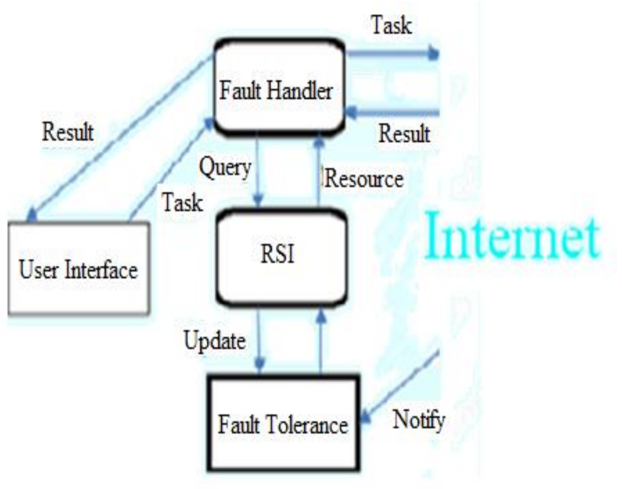

Figure. 1 Proposed system architecture

Once any replica completes its execution, all other replicas are terminated, and the network resources are released accordingly. The system determines the minimum number of replicas required for each task based on its understanding of the resources' tendency to fail. This approach helps reduce the adverse effects of failures on the network. Furthermore, the system focuses on selecting a suitable group of resources for task execution. It considers factors such as the response time of these resources, which includes the time required for transferring the task from the scheduler to the resource, waiting time in the queue, implementation time, and result transfer time from the resource back to the scheduler. By prioritizing resources with efficient response times, the system aims to optimize task execution.

The job scheduling process revolves around selecting a job from the job queue based on the desired user service quality. Subsequently, the resource information server (RIS) is consulted to acquire a suitable list of resources that meet the user's quality of service criteria [26].

The primary function of the resource information server (RIS) is to furnish a list of resources along with their estimated response times for task completion. This list is then organized by the scheduler based on the response times of each server. The server with the highest rank is chosen as the primary server responsible for executing the function.

However, there is a possibility that the primary resource may encounter a failure during task execution. To address this situation, the system implements a replication phase where certain resources from the available roster are designated as duplicates of the task. These resources are known as backup or reserve resources. There are two methods to improve performance:

By executing a job simultaneously on multiple resources, the response time to complete the task on the first resource serves as the benchmark. This response time may vary depending on server loads, requests, network conditions, and server capacity. Implementing the task on multiple resources has the potential to enhance the system's response time.

Employing repeated functions can help mitigate failures. It is sufficient to complete the implementation of a single replica to accomplish the task. This minimizes the impact of failure that may occur if there is only one replica.

Determining the number of replicas is critical. Increasing the number can significantly reduce the likelihood of task failure, but it also results in higher resource consumption and increased response time. On the other hand, an inadequate number of replicas may lead to task failure. Therefore, selecting the appropriate number of identical replicas should be proportional to both cases mentioned above. This ensures a high likelihood of task implementation while minimizing the impact on network stability.

7.1 Adaptive job replication

The main objective of the proposed algorithm is to determine the optimal number of replicas, which depends on the likelihood of resources experiencing failures. This relationship follows proportion: as the number of resource failures increases, the need for more replicas becomes higher, whereas as the number of resource failures decreases, the need for more replicas decreases as well.

As a result, the optimal number of replicas can be derived by analyzing the frequency of resource failures, which can be computed based on historical data. This number will vary depending on the type of task assigned to each resource.

Let's assume that "Nf" represents the count of failures experienced by a resource during the execution of its assigned tasks, and "Ns" denotes the count of successful completions by the resource. Whenever a resource fails to complete its assigned task, the value of "Nf" increments by one, and the task originally assigned to that resource is reassigned to another suitable resource within the network. Conversely, if the resource successfully completes its task, the value of "Ns" increases by one. Therefore, the propensity of resources to fail, denoted as "FTj," can be expressed as follows:

$$FTj = \frac{Nf}{Ns+Nf} \times 100\% \quad (2)$$

Thus, the possible success of the mission's implementation for resource j can be as follows:

$$Bj = 1 - FTj \quad (3)$$

Assuming that the resources R1, R2, ..., RN are dedicated to task j, then the inclination rate for failure in these resources is:

$$FTn = \frac{\sum_{j=1}^{N} FTj}{N} \times 100\% \quad (4)$$

The number of replicas of the task, k, was determined to be commensurate with the value of FTn. The minimum number of replicas should be at least one, and the number of resources available and suitable for the task should not exceed N. Accordingly, the highest limit of the number of replicas will be N. Thus, the possible success of the mission's implementation for all resources combined can be as follows:

$$Bn = 1 - FTn \quad (5)$$

7.2 Binomial probability distribution

The binomial probability distribution is a discrete probability distribution that describes the number of successes in a fixed number of independent Bernoulli trials, where each trial has only two possible outcomes: success or failure. The distribution is named after the binomial coefficient, which appears in the formula for calculating the probabilities.

The binomial probability distribution is characterized by two parameters:

1, The number of trials, denoted as "n," represents the fixed number of independent trials or experiments.

2, The probability of success on a single trial, denoted as "p," which represents the probability of the desired outcome occurring in each individual trial.

Using these parameters, the probability of obtaining exactly "k" successes in "n" trials can be calculated using the following formula:

$$P(X = k) = C(n,k) \times p^{k} \times (1 - p)^{n-k} \quad (6)$$

Where:

P(X = k) is the probability of obtaining exactly "k" successes.

C(n, k) is the binomial coefficient, also known as "n choose k," which represents the number of ways to choose "k" successes from "n" trials. It can be calculated as

$$\mathrm{C(n,k)} = \frac{n!}{\mathrm{k!} \times (\mathrm{n-k})!} \quad (7)$$

Received: September 19, 2023. Revised: November 4, 2023.

Table 2. RTT in ms for 20 servers

R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
159	169	175	175	190	210	200	190	150	132
R11	R12	R13	R14	R15	R16	R17	R18	R19	R20
145	235	212	160	112	1000	99	123	191	102

Table 3. RTT for remaining servers

R1	R2	R3	R4	R7	R8	R9	R10
159	169	175	175	200	190	150	132
R11	R14	R15	R17	R18	R19	R20	
145	160	112	99	123	191	102	

p^k represents the probability of "k" successes occurring, and (1 - p)^(n - k) represents the probability of "n - k" failures occurring.

The binomial probability distribution is often used in various applications, such as analyzing the success/failure outcomes of experiments, estimating the likelihood of events with two possible outcomes, and conducting hypothesis testing.

7.3 Joint probability

Refers to the probability that two events will both occur. In other words, joint probability is the likelihood of two events occurring together.

A joint probability is the probability of one or more independent events occurring simultaneously and is represented as P(A∩B) or P(A and B).

It can be calculated by multiplying the individual probabilities of the events: P(A) * P(B).

A joint probability is used to determine the likelihood of multiple events coinciding, but it does not provide information about causation or influence between the events.

Statisticians utilize joint probability to understand the probability of two or more events occurring together, considering their independent probabilities. However, it does not capture the relationship or influence between these events.

Section 4.7. explains the algorithm employed to decide the number of replicas for every submitted task involves a comparison between the values of "Bj" and "Bn" starting with the first resource on the dedicated list.

If "Bj > = Bn" an additional replica is added. The new list "Bt = Bt + Bj ",

The algorithm stops if it is "Bt > Bj."

8. Proposed algorithm.

The following steps outline the proposed algorithm:

- ❖ Receive tasks submitted by the user.
- ❖ For each task:
- ❖ Request "FT" for all resources from the Fault handler.
- ❖ Request "LH" for all resources from the Resource Information Server.
- ❖ Send packets to the servers in the obtained list to calculate the Round-Trip Time (RTT).
- ❖ Select a list of servers with the highest response time from the Resource Information Server.
- ❖ Calculate FTj, Bj, FTn, and Bn for all resources in the grid with the highest response time.
- ❖ If Bj > = Bn, add resource j to the new list "Bt"; otherwise, do not take any action.
- ❖ Arrange the resources in descending order based on their failure percentage.
- ❖ Calculate the average probability of successful resources using a binomial distribution.
- ❖ Count the number of replicas for the tasks.
- ❖ Define backup resources.
- ❖ End.

The algorithm continues these steps until the condition "Bt > Bj" is met. When this condition is satisfied, the algorithm stops, and the resulting list "Bt" represents the selected resources that satisfy the failure probability criteria.

9. Case study

Assume that we have 20 resources R1, R2..., R20, as shown in Table 2. First, calculate the LH (using formula 1) for each server and neglect the servers that have more than medium. (according to Microsoft [25] CPU utilization < = 66% and memory utilization < = 62%).

The summation of round trip time for all servers is 4129 ms, therefore the mean will be 206.45 ms.

The second step is to neglect all the servers that have round trip more than the mean so the result is shown in Table 3.

The following Fig. 2 represents the response time of all resources before any of them are excluded, while Fig. 3 represents those with a higher response time.

Third, Calculate FTj, Bj, FTn, and Bn, for all 15 resources in Table 3. such that, if Bj > Bn, then add resource j to the list otherwise no action, do the procedure to all the 15 resources of step two.

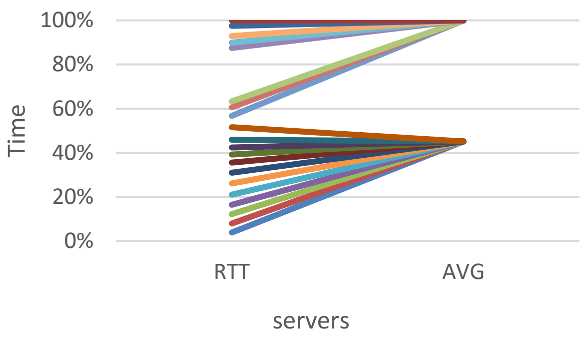

Figure. 2 Round trip time for all resources

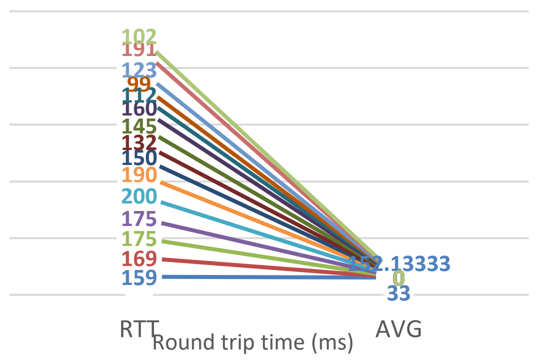

Figure. 3 Excluding all resources that have more value than average

Assume that "Nf" represents the count of failures experienced by a resource while executing its assigned tasks is 5, and "Ns" denotes the count of successful completions by the resource is 45.

$$propensity\ to\ fail = FTj = \frac{5}{5+45} \times 100\% = 10\%$$

$Bj = 1 - FTj$ =1-10/100 = 0.9

So, in the same way, the calculation of the tendency to failure is calculated using Formula 2 for all the remaining resources, suppose the tendency to failure of these resources is as follows in Table (4): then the summation of the tendency to failure of these resources is 122%, so the average is 0.8%, the discard all the server having a tendency to failure more than the average such as shown in Table 5.

Then the inclination rate for failure in these resources is 122/15 = 8.1%, then discard all the resources having more than this value as shown in Table 6.

After that will check the remaining resources to calculate the probability of failure using binomial distribution and arrange them descending as shown in Table 6.

Table 4. FT for 15 servers

R1	R2	R3	R4	R7	R8	R9	R10
10%	5%	7%	4%	15%	20%	9%	5%
R11	R14	R15	R17	R18	R19	R20	
4%	20%	11%	2%	1%	7%	2%	

Table 5. FT for 9 servers

R2	R3	R4	R1 0	R1 1	R1 7	R1 8	R1 9	R2 0
5 %	7 %	4 %	5%	4%	2%	1%	7%	2%

Table 6. Descending FT for 9 servers

R1 8	R2 0	R1 7	R4	R1 1	R2	R1 0	R3	R1 9
1%	2%	2%	4 %	4%	5 %	5%	7 %	7%

$$FTn = \frac{\sum_{j=1}^{N} FTj}{N} \times 100\% \quad = 122/15 \times 100\% = 8{,}1\%$$

The total number of servers available is 9, if we use only 0ne servers, then the probability of failure is 0.01. and 0.00000004 for using two servers, for three servers is 6.4E-17, and for four servers is 6.55E-28,

Therefore, the possible success of the mission's implementation using formula 5 is 0.99, 0.99999996, 1, and 1, respectively, as shown

The statement describes the calculation process for determining the success probability in a distributed system. It considers scenarios where all servers fail to complete their assigned tasks. The goal is to find the probability of at least one server succeeding in completing its job.

To calculate this probability, the following steps are taken:

First, the calculation considers the situation where all servers fail to complete their tasks. This is important to understand the complementary scenario to the desired outcome (at least one server succeeding).

Formula 5 is used to compute the success probability. This formula takes into account the probability of success for individual servers and calculates the probability of having at least one success among multiple servers.

The reason for this approach is that for the distributed system to achieve fault tolerance, it should be capable of successfully completing tasks even if some servers fail. By calculating the probability of at least one server succeeding, the system can be assessed for its ability to handle failures while still accomplishing its assigned jobs. So,

For one server use (R18), P(x=0) = 1C0 p0 q1-0 = 1! / 1! 0! × (99/100)0 × (1 /100)1 = 0.01.

Then the success probability for R18 using formula 5 is

Bn = 1-0.01= 0.99.

The joint probability of failure for the first two servers (R18, R20) is 1% **×** 2% = 0.0002

P(x=0) = 2C0 p^0 q^{2-0} = 2! / 2! 0! × $(99.98/100)^0$× $(0.02 /100)^2$ = 0.00000004.

Then the success probability for both of them using formula 5 is

Bn = 1- 0.00000004 = 0.99999996 = close to 1

The joint probability of failure for the first three servers (R18, R20, R17) is 1% × 2% × 2% = 0.000004

P(x=0) = 3C0 p^0 q^{3-0} = 3! / 3! 0! × $(0.999936)^0$× $(0.000004)^3$ = 6.4E-17 = close to 0.

Then the success probability for all three servers (R18, R20, R7) using formula 5 is

Bn = 1- 0 = 1

The joint probability of failure for the first four servers (R18, R20, R17, R4) is 1% × 2% × 2% × 4% = 0.00000016.

P(x=0) = 3C0 p^0 q^{3-0} = 3! / 3! 0! × $(0.99999984)^0$ × $(0.00000016)^4$ = 6.55E-28 = very close to 0.

Then the success probability for all four servers (R18, R20, R17, R4) using formula 5 is

Bn = 1- 0 = 1

The algorithm stops working because when adding one more server (R4) the probability of failure becomes close to zero. also, it is noted that each of the above resources shown in Table 6 represents a high probability of success in implementing the task, Accordingly, only two resources can be satisfied to represent the resources of the replication, and the first three are the best of these resources, due to the high possibility of carrying out this task.

Result and discussıon.

Independent events refer to occurrences that do not influence each other. When event Q is considered independent of event K, it means that the probability of event Q happening is unaffected by the occurrence of event K.

If Q and K are independent events in a random experiment, the probability of both events occurring simultaneously, denoted as P(Q⋂K), can be calculated by multiplying their individual probabilities, represented as P(Q) and P(K):

$$\mathbf{P(Q\cap K) = P(Q) \times P(K)} \tag{8}$$

In the case of multiple independent events, let's say Q1, Q2, ..., Qn, associated with a random experiment. The probability of all these events happening simultaneously, represented as P(Q1⋂Q2⋂Q3⋯⋂Qn), can be calculated by multiplying the individual probabilities of each event:

$$\mathbf{P(Q1\cap Q2\cap Q3 \cdots \cap Qn) = P(Q1) \times P(Q2) \times P(Q3) \times \ldots \times P(Qn).} \tag{9}$$

The likelihood of successfully accomplishing the task increases with additional resources, but this approach has drawbacks due to resource consumption, network instability, and potential network collapse [27, 28].

To ensure fault tolerance, at least one replica should be added. For instance, when considering the first two servers, the probability of both failing simultaneously is incredibly low (0.00000004) according to the binomial probability distribution. Individually, the first server's failure probability is 0.01, and the second server's failure probability is 0.02, with respective success probabilities of 0.99 and 0.98. Therefore, their joint probability of failure is 0.01 * 0.02 = 0.0002.

On the other hand, when there are three working servers, the probability of all of them failing together is nearly zero, and their individual failure probabilities are 0.01, 0.02, and 0.02. Combining these probabilities using joint probability gives a failure rate of 0.01 * 0.02 * 0.02 = 0.000004.

From the above explanation, it becomes evident that as the number of servers increases, the probability of failure significantly decreases, approaching close to zero. For instance, with only two servers, the failure rate was 0.0002, but with the addition of another server, it decreased to 0.000004. When a fourth server is added, the tendency towards failure becomes exceptionally low, almost reaching zero. Fig. 4 illustrates the failure probability of joint

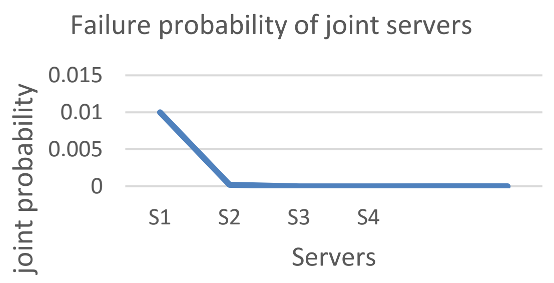

Figure. 4 Failure probability of joint servers

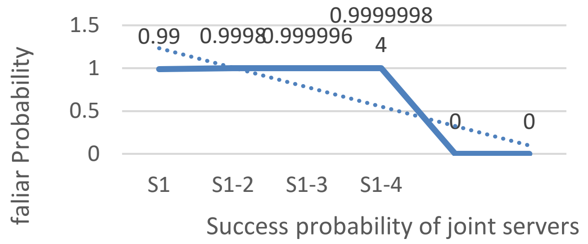

Figure. 5 Success probability of joint servers

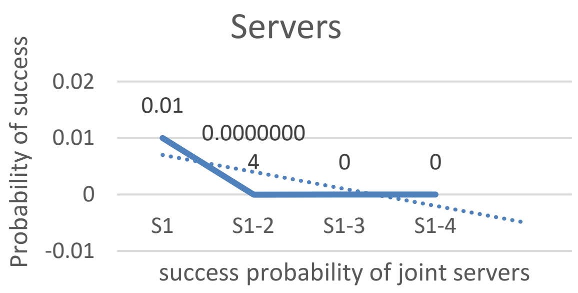

Figure. 6 Failure probability of binomial probability

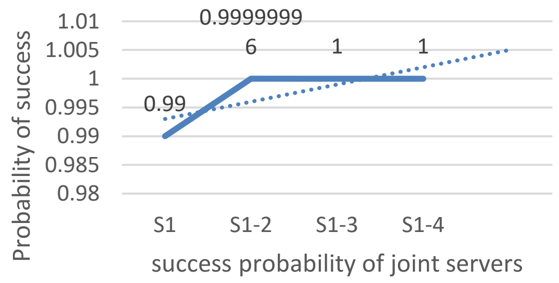

Figure. 7 Success probability binomial probability

servers, while Fig. 5 represents the probability of their success.

By applying Eq. (6) we can deduce the probability of successful execution of the task for all servers. Fig. 6 illustrates the Failure probability of binomial probability and from Fig. 7 it turns out that the best case is to choose only three servers for execution since the probability of execution on one of them will be very high and close to the confirmed execution of the task, therefore, execution with a minimum number of resources, the result is not consuming system resources and ensuring execution with a high probability of task execution.

10. Result and discussion

The proposed system introduces proactive scheduling techniques to address failures and minimize their impact on task execution within the network. This section discusses the key results and the effectiveness of the system in achieving its objectives.

10.1 Mitigating failures through replication

The system's proactive approach to mitigating failures involves creating multiple replicas of tasks on different resources and executing them simultaneously. This redundancy ensures that if one resource fails during task execution, it does not disrupt the progress of the tasks on the remaining resources. This strategy effectively minimizes the impact of failures and maintains uninterrupted task execution.

Additionally, the system incorporates an intelligent approach to determining the minimum number of replicas required for each task. It analyzes historical data on resource failures to assess the likelihood of failures occurring. This adaptive job replication strategy ensures that an appropriate number of replicas are created, balancing the need for failure mitigation with resource efficiency.

10.2 Resource selection for task execution

The system goes further to optimize task execution by selecting resources based on their response times. Factors such as the time required for transferring tasks to resources, queue waiting times, implementation times, and result transfer times are considered. By prioritizing resources with efficient response times, the system aims to enhance overall task execution efficiency.

10.3 Adaptive job replication

The algorithm used to determine the optimal number of replicas for each task demonstrates a data-driven approach. It considers the historical failure rates of resources and adapts the replication strategy accordingly. This ensures that resources with a higher likelihood of failure receive more replicas, reducing the risk of task failure.

10.4 Binomial probability distribution

The use of the binomial probability distribution to calculate the probability of task success is a statistically sound approach. It takes into account the number of replicas and the historical failure rates to estimate the likelihood of successful task execution. This method provides a solid foundation for decision-making regarding replication levels.

10.5 Joint probability

While joint probability is discussed, it's essential to clarify that this concept is not directly applied to the algorithm but is mentioned to explain the probability of two events occurring together. It's a theoretical concept and does not play a direct role in the replication decision-making process.

10.6 Effectiveness of the proposed algorithm

The proposed algorithm effectively combines historical data on resource failures, response times, and statistical methods to determine the optimal number of replicas for each task. By doing so, it ensures a high likelihood of task implementation while minimizing the impact on network stability. This proactive approach significantly reduces the risk of task failures and improves overall network reliability.

11. Conclusion

To avoid errors and minimize time and effort wasted in redoing tasks, most existing methods utilize a fixed number of resources, disregarding the task's type and size. Despite the abundance of resources, the selection process lacks careful consideration, resulting in uncertain outcomes during implementation.

This paper introduces a fault-tolerant strategy for scheduling functions in cloud/grid computing. The proposed approach specifically focuses on intentionally choosing a specific number of resources based on their known response times. This deliberate selection leads to exceptional performance by minimizing the impact on network resources and ensuring uninterrupted task execution. Consequently, the number of resources needed for task execution is reduced, significantly increasing the probability of completing tasks and approaching a level close to certainty.

By executing tasks on meticulously selected resources with known response times, highly favorable outcomes are achieved. This approach optimizes performance, reduces resource utilization, enables seamless task completion with minimal disruption, and maintains network efficiency. Consequently, the probability of successfully executing tasks becomes significantly high, reaching a level that approaches certainty.

Conflicts of interest

There is no conflict of interest regarding the publication of this paper

Authors' contributions are as follows:

Mahdi S. Almhanna conceptualized the methodology, authored the primary manuscript, and created all figures except (Figures 1, 2, and 3), as well as all tables.

Tariq A. Murshedi offered valuable insights into the proposed methodology, generated Figures 1, 2, and 3, and reviewed the manuscript.

Ahmed M. Al-Salih contributed to the concept of incorporating Joint Probability in section 4.6. • Rafah M. Almuttairi conducted the experiments, meticulously reviewed and enhanced the English language across the manuscript, and oversaw its development at various stages.

Acknowledgements

The authors express their gratitude to the editors and reviewers for their valuable and constructive feedback. Additionally, we would like to extend our thanks and appreciation to the University of Babylon and the College of Information Technology for their unwavering support of the staff.

References

[1] A. Mohammed, “A fault-tolerant scheduling system for computational grids”, *Computers & Electrical Engineering*, Vol. 11, No.2, pp. 115-121, 2012.

[2] M. S. Almhanna, “Minimizing replica idle time”, In: *Proc. of 2017 Annual Conference on New Trends in Information & Communications Technology Applications (NTICT)*, pp. 128-131, 2017, doi: 10.1109/NTICT.2017.7976134.

[3] R. M. Almuttairi, R. Wankar, A. Negi, R. R. Chillarige, and M. S. Almahna, “New replica selection technique for binding replica sites in Data Grids”, In: *Proc. of 2010 1st International Conference on Energy, Power and Control (EPC-IQ)*, pp. 187-194, 2010.

[4] K. H. Anun and M. S. Almhanna, “Web Server Load Balancing Based on Number of Client Connections on Docker Swarm”, In: *Proc. of 2021 2nd Information Technology To Enhance*

e-learning and Other Application, pp. 70-75, 2021, doi: 10.1109/IT ELA 52201.2021.9773748.

[5] S. A. Abbas and M. S. Almhanna, “Distributed Denial of Service Attacks Detection System by Machine Learning Based on Dimensionality Reduction”, *J. Phys.: Conf. Ser.1804012136*, Vol. 1804, No. 1, pp. 012136, 2021, doi: 10.1088/17426596/1804/1/012136.

[6] S. S. Sathya and K. S. Babu, “Survey of fault-tolerant techniques for the grid”, *Computer Science Review*, Vol. 4, No. 2, pp. 101-120, 2010.

[7] Q. Zheng and B. Veeravalli, “On the design of communication-aware fault-tolerant scheduling algorithms for precedence constrained tasks in grid computing systems with dedicated communication devices”, *J. Parallel and Distributed Computing*, Vol. 69, pp. 282-294, 2009.

[8] J. Singh, “An Optimal Resource Provisioning Scheme Using QoS in Cloud Computing Based Upon the Dynamic Clustering and Self-Adaptive Hybrid Optimization Algorithm”, *International Journal of Intelligent Engineering and Systems*, Vol. 15, No. 3, 2022, doi: 10.22266/ijies2022.0630.13.

[9] P. Kumari and P. Kaur, “A survey of fault tolerance in cloud computing”, *Journal of King Saud University - Computer and Information Sciences*, Vol. 33, Issue 10, pp. 1159-1176, 2021.

[10] F. Khan, K. Qureshi, and B. Nazir, “Performance evolution of fault tolerance techniques in grid computing system”, *J. Computing, and Electrical Engineering*, Vol. 36, pp. 1110-1122, 2010.

[11] L. Yao, X. Wang, Q. Z. Sheng, S. Dustdar, and S. Zhang, “Recommendations on the Internet of Things: Requirements, Challenges, and Directions”, *IEEE Internet Computing*, Vol. 23, No. 3, pp. 46-54, 2019, doi: 10.1109/MIC.2019.2909607.

[12] A. Litke, K. Tserpes, K. Dolkas, and T. H. Varvarigou, “A Task Replication and Fair Resource Management Scheme for Fault Tolerant Grids”, In: *Proc. of Advances in Grid Computing - EGC 2005*, Vol. 3470, pp. 1022-1031, 2005.

[13] S. Christian, *Specification and Analytical Evaluation of Heterogeneous Dynamic Quorum-Based Data Replication Schemes*, Springer Link, pp.13-79,2012.

[14] S. W. Kwak, K. H. You, and J. M. Yang, “Checkpoint Management with Double Modular Redundancy Based on the Probability of Task Completion”, *Journal of Computer Science and Technology*, Vol. 2, pp. 273-280, 2012.

[15] K. G. Srinivasa, G. M. Siddesh, and S. Cherian, “Fault-tolerant middleware for grid computing”, In: *Proc. of 12th IEEE International Conference on High Performance Computing and Communications, Melbourne*, Australia, pp. 635-640, Sep. 1-3, 2010б.

[16] J. H. Abawajy, “Fault-tolerant scheduling policy for grid computing systems”, In: *Proc. of 18th International Parallel and Distributed Processing Symposium*, Santa Fe, NM, USA, pp. 238-244, 2004, doi: 10.1109/IPDPS.2004.1303290.

[17] M. Chetepen, B. Dhoedt, F. Cleays, and P. V. Rolleghem, “Evaluation of replication and rescheduling heuristics for gird systems with varying resource availability”, In: *Proc. of 18th International Conference on Parallel and Distributed Computing Systems, Anaheim*, CA, USA, pp. 622-627, Nov. 13-15, 2006.

[18] C. Jiang and D. H. Zhou, “Fault detection and identification for uncertain linear time-delay systems”, *Computers and Chemical Engineering*, Vol. 30, No. 2, pp. 228–242, 2005.

[19] M. Amoon, “A fault-tolerant scheduling system for computational grids”, *Computers & Electrical Engineering*, Vol. 38, No. 2, pp. 399-412, 2012.

[20] S. Song, Y. Kwok, and K. Hwang, “Risk-Resilient Heuristics and Genetic Algorithms for Security-Assured Grid Job Scheduling”, *IEEE Transactions on Computers*, Vol. 55, No. 6, pp.703-719, 2006.

[21] Y. Zhang and N. Lu, “Parameter Selection for a Centralized Thermostatically Controlled Appliances Load Controller Used for Intra-Hour Load Balancing”, *IEEE Transactions on Smart Grid*, Vol. 4, No. 4, pp. 2100-2108, 2013, doi: 10.1109/TSG.2013.2258950.

[22] S. Jaber, Y. Ali, and N. Ibrahim, “An Automated Task Scheduling Model Using a Multi-objective Improved Cuckoo Optimization Algorithm”, *International Journal of Intelligent Engineering and Systems*, Vol. 15, No. 1, 2022, doi: 10.22266/ijies2022.0228.27.

[23] A. S. Kadhim and M. E. ManaaHybrid, “Load-balancing algorithm for distributed fog computing in the Internet of Things environment”, *Bulletin of Electrical Engineering and Informatics*, Vol. 11, No. 6, pp. 3462-3470, 2022, doi: 10.11591/eei.v11i6.4127.

[24] https://devblogs.microsoft.com/premier-developer/calculating-server-capacity-and-planning-for-future-user-growth/

Received: September 19, 2023. Revised: November 4, 2023.

[25] K. Sreenu and S. Malempati, "Multiple Resource Attributes and Conditional Logic Assisted Task Scheduling in Cloud Computing", *International Journal of Intelligent Engineering and Systems*, Vol. 16, No. 3, pp. 677-690, 2023, doi: 10.22266/ijies2023.0630.54.

[26] M. S. Almhanna, F. S. A. Turaihi, and T. A. Murshedi, "Reducing waiting and idle time for a group of jobs in the grid Computing", *Bulletin of Electrical Engineering and Informatics*, Vol. 12, No. 5, pp. 3115-3123, 2023.

[27] A. M. A. A. Muqarm, "Naseer Ali Hussies, Dynamic Cost-Optimized Resources Management and Task Scheduling with Deadline Constraint for Mobile Crowd Sensing Environment International", *Journal of Intelligent Engineering and Systems*, Vol. 16, No. 3, 2023, doi: 10.22266/ijies2023.0630.16.