Fault Tolerant Task Scheduling for Distributed Systems Based on Genetic Algorithm

Shigan YU, Xiaoling RU*
Fuyang Normal University, Fuyang, Anhui 236041, China
yushigan@163.com, ruxiaoling@163.com (*Corresponding author*)

Abstract: When handling large-scale tasks in distributed systems, a critical challenge arises as the system scales up and the number of processors grows: the increasing prevalence of failures, which jeopardizes the timely completion of tasks. To address this issue, a fault-tolerant distributed system task scheduling model was developed, and the scheduling sequence was optimized by using a genetic algorithm aimed at minimizing the task completion time. The obtained results demonstrate that, under normal conditions, the model can achieve a resource utilization rate as high as 99.1%. Specifically, for a total of four faulty processors and 10,000 tasks, the proposed model's task completion time was 21,571 ms, outperforming other comparative models such as the Quartz model. Furthermore, when the task volume was 200, an analysis of the optimal scheduling sequence derived from the genetic algorithm revealed a task scheduling length ratio of 3.21, which was lower than the value achieved by other methods such as the ascending-order sorting method. This paper proves that the proposed model effectively reduces the task completion time and enhances resource utilization in distributed system task scheduling. Additionally, by employing genetic algorithms for solving this model, the optimal processor scheduling sequence can be determined. This research approach improves a distributed system's stability and task completion efficiency, offering a novel strategy for fault-tolerant scheduling in distributed systems.

Keywords: Distributed system, Fault tolerance, Task scheduling, GA, Optimal order, Completion time.

1. Introduction

With the rapid development of big data and artificial intelligence technology, distributed systems have been widely applied in various fields. A distributed system consists of many individual nodes that interact and work together via a network to provide efficient and reliable services to users. However, due to the complexity, heterogeneity, and exposure of distributed systems to harsh environments, fault-tolerant task scheduling has always been an important research topic (Abedpour et al., 2024). Specifically, distributed systems suffer from long scheduling delays, a low scheduling accuracy, and low fault tolerance success rates when executing tasks, resulting in a poor task execution efficiency.

The Genetic Algorithm (GA), as an optimization algorithm, has received widespread attention in the area of fault-tolerant task scheduling in distributed systems. However, in distributed systems, the communication and collaboration between nodes require a significant amount of computation and data processing, and as the amount of nodes increases, the overhead of such computation and communication also increases. Meanwhile, more tasks need to be allocated and scheduled, which also increases the complexity of the system (Palle, 2023). Therefore, when dealing with large-scale problems, GA is prone to problems such as an excessive search space and a slow convergence speed. Meanwhile, the communication and collaboration between nodes in distributed systems, as well as the interaction between nodes and the external environment, allow the system to face more complex dynamic environments (Krishnamurthy, 2023). How to make GA improve its adaptability and self-learning ability in this dynamic environment is a prerequisite for a good application effect (Reghenzani, Guo & Fornaciari, 2023). A GA-based fault-tolerant task scheduling method for distributed systems is proposed to address the above issues, in order to provide theoretical support for the stable operation and efficient service of distributed systems. The innovation brought about by this research lies in the introduction of dynamic fault-tolerant scheduling algorithms and GA-based scheduling sequence optimization methods for distributed system fault-tolerant task scheduling models, thereby ensuring the stability of system operation.

The remainder of this paper is structured as follows. Section 2 presents a literature review, which sorts out the existing research in the field of fault-tolerant task scheduling in distributed systems and points out the deficiencies of the current research in terms of its adaptation to complex dynamic environments and solution efficiency. Section 3 elaborates on the research methods, including the fault-tolerant task scheduling model with a star structure and the scheduling sequence optimization method based on the GA. Further on, Section 4 includes experimental comparisons with the purpose of analyzing the performance of the proposed model in terms of task completion time, resource utilization and others, as well as the advantages of the GA-based scheduling optimization method in terms of scheduling length ratio and acceleration ratio. Finally, Section 5 summarizes the research results, points out the shortcomings of not fully considering the complex network environment, and proposes future research directions.

2. Literature Review

In distributed systems, it is necessary to allocate and schedule tasks reasonably to guarantee the stable operation and efficient performance of the system in response to possible failure situations. Indhumathi et al. (2023) proposed a task scheduling algorithm grounded on gray wolf optimization, which achieved lower failure rates and higher throughput by simultaneously performing task scheduling and fault detection. In comparison with the traditional techniques, this method proved to be advantageous regarding the overall project duration, operation time, and utilization rate. Xu et al. (2023) presented a fault-tolerant aware optimization model for virtual machine scheduling in cloud data centers.

To address the optimization model, a greedy-based best fit decreasing algorithm was proposed. The experimental outcomes showed that in comparison with the other three algorithms, the overall satisfaction related to this algorithm increased by 38.3%, 20.9%, and 14.6%, respectively. Ismael et al. (2022) pointed out that distributed computing systems developed with the increase of mathematically complex algorithms, achieving a more efficient use by optimizing task allocation and scheduling methods. Li et al. (2023) proposed a metaheuristic algorithm for scheduling microservice workflow applications in the cloud environment. The algorithm minimized execution costs through greedy fault-tolerant scheduling strategies while adhering to deadlines and ensuring dependability. The experimental outcomes showed that in comparison with the existing algorithms, this algorithm could reduce the execution costs. Siyadatzadeh et al. (2023) proposed a new primary-backup task allocation approach grounded on machine learning, which achieved an excellent performance in dynamic environments by using reinforcement learning methods, reducing task dropout rates and balancing workload distribution. Compared with the existing technologies, the new method reduced the task dropout rate by up to 84% and improved system reliability by nearly 72%.

The application of GA in distributed systems is mainly reflected in task scheduling and optimization problems. By simulating the evolution process in nature, it finds solutions in a shorter period of time, and improves the parallel performance and load balancing of the system. Lu et al. (2023) proposed an instant workflow scheduling method that combined GA and deep reinforcement learning, which was divided into two stages. This method, combined with the global search capability and environmental perception decision-making ability, improved the computational efficiency, and reduced the execution costs and response time. George et al. (2023) explored load balancing and dynamic resource scheduling problems in cloud computing environments using methods such as data mining and analysis, distributed systems, fuzzy logic, and GA. Research showed that using fuzzy logic for intelligent decision-making and GA for optimized scheduling could effectively improve the general capability of cloud computing systems and reduce technical complexity. Çavdar et al. (2024) presented a GA-based virtual machine placement method aimed at improving the capability, accessibility, and dependability of data centers while minimizing the waste of resources, network burden, and energy usage. By comparing it with alternative placement techniques in terms of efficiency, network bandwidth consumption, and energy cost, the outcomes showed that this method achieved a better performance. Ge et al. (2024) proposed an information-driven distributed GA aimed at achieving optimal anonymity by balancing privacy protection and maintaining data quality. This method combined multiple operators and used a distributed population model to enhance the variety within that population.

The results confirmed the advantage of this method regarding solution precision, the efficiency of convergence, and the validity of the involved components. Singh & Chaturvedi (2024) proposed a hybrid GA improved particle swarm optimization algorithm for optimizing workflow task allocation in cloud computing environments in order to shorten the completion time, reduce costs, and decrease energy consumption. The experiment outcomes showed that this method had advantages over other algorithms in reducing the task execution time and lowering the execution costs.

In summary, the existing research has achieved significant results in fault-tolerant task scheduling in distributed systems, but there are still some shortcomings. Some methods may have limitations in handling task scheduling and faults in complex dynamic environments, while some algorithms still need to improve in terms of efficiency and accuracy. Therefore, in response to these challenges, GA-based fault-tolerant task scheduling methods for distributed systems will also be explored in the future, to further enhance the stability and capability of those systems.

3. Research Methodology

3.1 Fault-tolerant Task Scheduling Model

With the increase of data scale, the traditional centralized storage systems can no longer meet the current processing needs. Distributed systems can accelerate the computing speed and solve the problem of single machine inability by concentrating the computing power of multiple physical computers. The core of distributed computing is task scheduling, which is responsible for assigning tasks to different nodes for processing. In the face of failures, traditional task scheduling methods may not be able to guarantee the system's operational functionality (Naderi et al., 2024; Păun et al., 2024). Thus, studying fault-tolerant task scheduling methods for distributed systems is of great significance for improving the reliability and stability of the system. This paper is based on a distributed system with a star structure, as shown in Figure 1.

A distributed system with a star structure consists of multiple processors, including a central processor and other peripheral slave processors. The central processing unit is responsible for handling tasks such as data storage, processing, and forwarding, while the peripheral slave processing units are responsible for executing specific computing tasks. The central processing unit and the slave processing units are connected through a communication network to achieve fast data transmission and collaborative processing. In addition, distributed systems with star structures can expand their processing and storage capabilities by increasing the number of peripheral nodes, thereby improving their scalability and flexibility. The main processor will divide the task into multiple independent subtasks of different sizes, and then allocate them from high to low according to the computing power of the processor (Vargas, Mosquera & Trujillo, 2024; Benamor et al., 2023). During this process, the slave processors can simultaneously perform task transmission and computation, and transmit the results back to the main processor after each scheduling. To reduce the waiting time for the processor, the scheduling model is optimized and improved, resulting in a

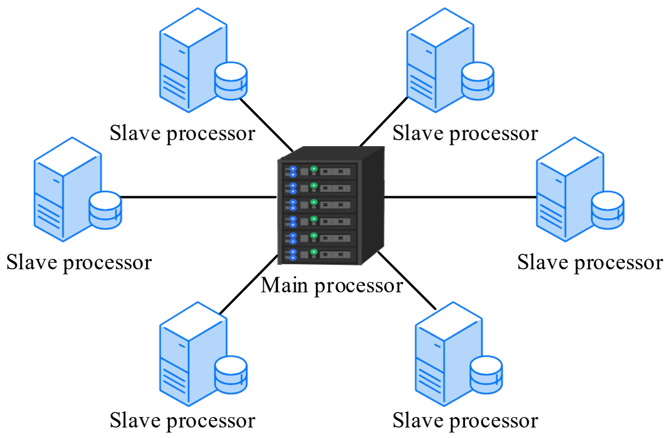


Figure 1 Distributed system architecture with star structure

fault-tolerant task scheduling model. The timing diagram is shown in Figure 2.

The improved model assigns tasks to multiple processors for processing, and performs transmission and retrieval operations after each scheduling is completed. This model includes a fault-tolerant mechanism that ensures it can continue to run in the event of an error. The scheduling process is divided into three stages, namely initial scheduling, internal scheduling, and final scheduling to ensure that tasks can be efficiently allocated and processed. In addition, the model ensures that the amount of tasks assigned by the main processor to each processor remains consistent in each round of scheduling, to ensure workload balancing among the processors. When a processor fails, the main processor immediately reschedules the uncalculated tasks and optimizes the task allocation ratio to complete task calculations in the shortest possible time (Zangana & Zeebaree, 2024). Meanwhile, the optimization target of the model is to improve the system's fault tolerance performance, shorten the task completion time (TCT), and the utilization rate. In the internal scheduling process, the processing time for each scheduling process is the same, as shown in equation (1):

$$s_1 + w_1\beta_1 V = s_2 + w_2\beta_2 V = \ldots = s_n + w_n\beta_n V \quad (1)$$

In equation (1), s_n is the cost of starting the n-th processor, β_n denotes the task partitioning weight, V represents the number of tasks scheduled per trip, and the task processing time per processing unit is denoted by w_n. In the internal scheduling process, the allocation ratio for processor tasks is calculated as shown in equation (2):

$$\beta_i = \frac{w_1}{w_i}\beta_1 + \frac{s_1 - s_i}{w_i}\frac{1}{V} \quad (2)$$

In equation (2), the task allocation ratio for the i-th processor is β_i. In the scheduling model, the TCT is expressed in equation (3):

$$T(m) = ms_1 + w_1 V(\alpha_1 + \gamma_1 + m\beta_1 - 2\beta_1) + \sum_{i=1}^{n}(o_i + g_1\gamma_1\theta V) \quad (3)$$

In equation (3), the total number of scheduling trips is denoted by m, the task allocation ratio for the first trip is denoted by α_1, and the task allocation ratio for the last trip is expressed by γ_1. The transmission starting time of processor i is expressed by o_i, g_1 represents the task processing time per processing unit for the communication link, and the proportion of task resolution is denoted by θ. The model aims to minimize TCT and takes the number of scheduling trips m as the decision variable. The first constraint condition a requires that in each scheduling process, the task will be divided into multiple subtasks and allocated to all processors participating in the scheduling. The second constraint condition b requires that the number of scheduling trips m be greater than or equal to 3. In the process of task scheduling, a Dynamic Fault Tolerant Scheduling Algorithm (DFTSA) is proposed for addressing processor failures. The process is shown in Figure 3. In the process of task scheduling for every processor, if a failure occurs, the main processor will reschedule the unprocessed tasks and recalculate the optimal task allocation ratio.

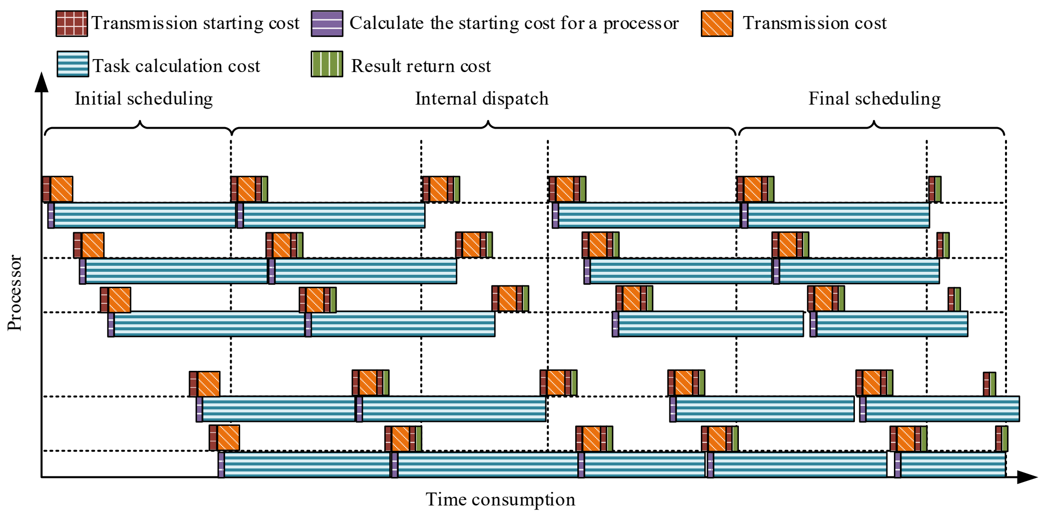


Figure 2. Time sequence diagram for the fault-tolerant task scheduling model

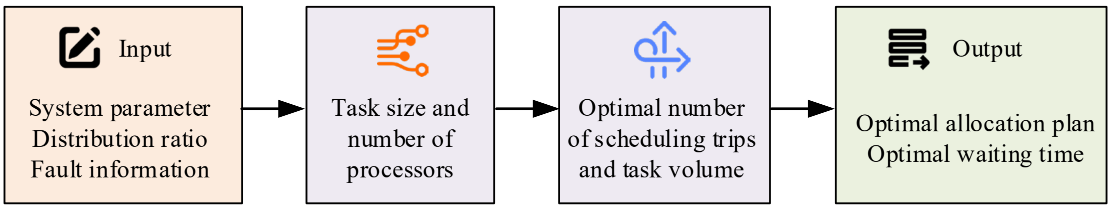


Figure 3. The process of DFTSA

In the last two scheduling processes, if a processor fails, the remaining tasks after task scheduling will be allocated to the processor with the strongest performance to save distributed system resources (Jeyaraman et al., 2024; Malti et al., 2024). Otherwise, the main processor will reallocate internal scheduling tasks through the DFTSA. After the resources of the first normal processor are released, the main processor will immediately reschedule. The resource release time for the processor is expressed in equation (4):

$$T_f^i = T_s + o_i + \sum_{j=1}^{i-1}(o_i + (\theta+1)g_i\beta_i V), i \in Q \tag{4}$$

In equation (4), T_f^i represents the resource release time for the i-th processor, T_s is the time for the main processor to restart scheduling, and the set of normal processors before the failure is denoted by Q. The time when the processor starts receiving tasks is expressed in equation (5):

$$T_s^i = T_s + o_i + \sum_{j=1}^{i-1}(o_i + g_i\alpha_i^{'}V'), i \in R \tag{5}$$

In equation (5), the time when the i-th processor starts receiving tasks is denoted by T_s^i. After recalculation, the task allocation ratio for the first scheduling is represented by $\alpha_i^{'}$, the number of tasks per trip is denoted by V', and the set of normal processors after failure is expressed by R. After rescheduling, the time conflict of the processor is calculated as shown in equation (6):

$$T_c^i = T_f^i - T_s^i \tag{6}$$

In equation (6), the time conflict for the i-th processor is denoted by T_c^i. The optimal waiting time for the processor is calculated as shown in equation (7):

$$T_w = T_f^k - T_s^k \tag{7}$$

In equation (7), for the context after a fault occurrence, the last normal processor is denoted by k. Before the processor executes a task, it needs to wait for a period of time to ensure that no other tasks are using the same computing resource, thereby avoiding time conflicts (Long et al., 2023).

3.2 GA-based Scheduling Sequence Optimization Method

To further raise the fault tolerance capability of distributed systems in the event of faults, a GA-based scheduling sequence optimization method is introduced based on the fault-tolerant task scheduling model. The GA is an optimization algorithm that simulates the progression of natural evolution, employing techniques like picking, mixing, and altering to discover the best solution (Lian et al., 2024). The GA process is shown in Figure 4:

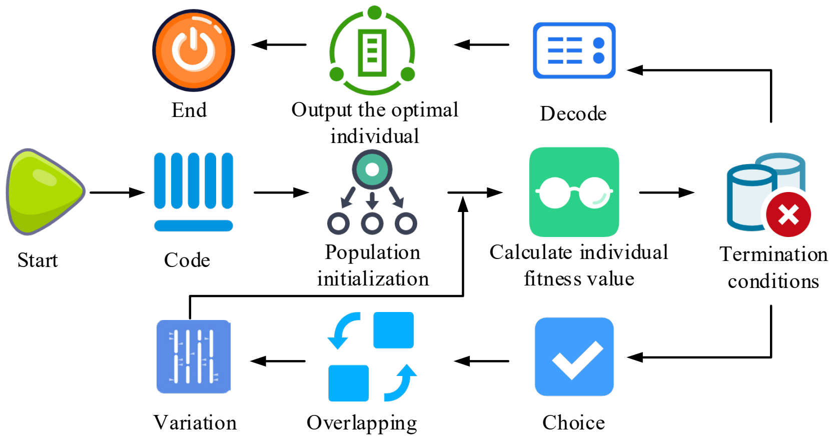

End
Output the optimal individual
Decode
Start
Code
Population initialization
Calculate individual fitness value
Termination conditions
Variation
Overlapping
Choice

Figure 4. The process of GA

The GA-based scheduling sequence optimization method mainly optimizes the task scheduling sequence to reduce the time conflict of faulty processors and improve the system's fault tolerance performance. The study uses integer encoding to represent the number of processors as an integer sequence. In the process of population initialization, first a reference individual is created with the same number of processors, and then the sequence is randomly arranged multiple times to generate the initial population. The fitness function is expressed in equation (8):

$$f_1(x) = \frac{1}{T_c + T} \tag{8}$$

In equation (8), the shorter the time conflict T_c and the TCT T, the better the system performance (Liu, 2022). The individual penalty term is expressed in equation (9):

$$f_2(x) = \vartheta(6 - x) \tag{9}$$

In equation (9), the hyperparameter is ϑ. The penalty term for individual fitness values is expressed in equation (10):

$$f_3(x) = O(x) + \upsilon \tag{10}$$

In equation (10), the objective function value is denoted by $O(x)$ and the penalty coefficient is expressed by υ. During the crossover operation, the algorithm adopts a sequential crossover strategy. During the crossover process, two parental individuals first randomly select two crossover points, then they exchange genes between these two points, and finally fill in the remaining genes in order (Shiva Rama Krishna & Mangalampalli, 2023). The mutation operation consists in shifting an individual's gene to the first position of the gene string. In addition, local search strategies have also been applied in the GA. Local search is performed on each individual, and the initial one is substituted with a new one when the fitness value of the new searched individual is better. Finally, the GA-based scheduling sequence optimization process is shown in Figure 5.

Firstly, the evolutionary parameter is initialized at 0 and population initialization is performed based on the given basic parameters and task volume. Next, in each generation, crossover, mutation, and local search operations are performed sequentially. Then, the individual fitness values are calculated based on the fitness function, and a strategy for preserving the best individuals is employed to ensure their direct inclusion in the subsequent generation. For the remaining individuals, the roulette wheel selection method is employed for selecting the remaining individuals until the required population size is reached in the next generation (Pham, Nguyen Dang & Nguyen, 2024). Then, iterations will be conducted for the above steps until the termination iteration is reached. Ultimately, the optimal scheduling sequence for the output processor is determined.

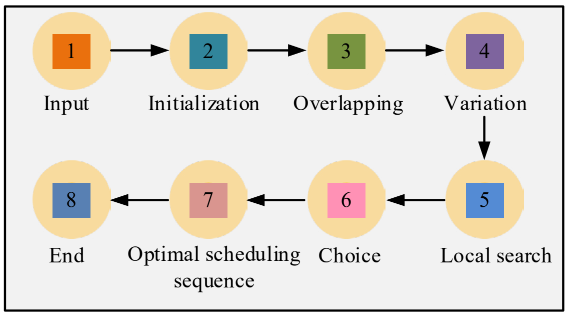
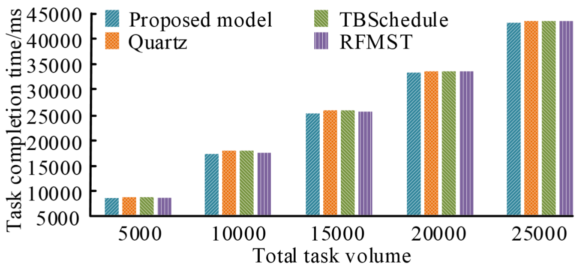


Figure 5. GA-based scheduling sequence optimization process

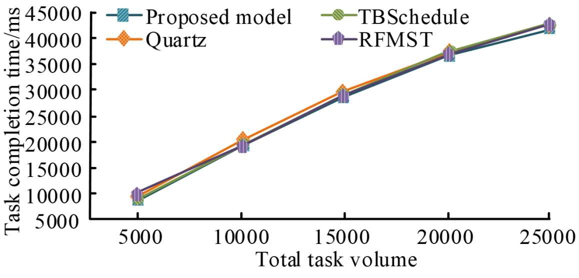
4. Results and Discussion

4.1 Performance Study of the Fault Tolerant Task Scheduling Model for Distributed Systems

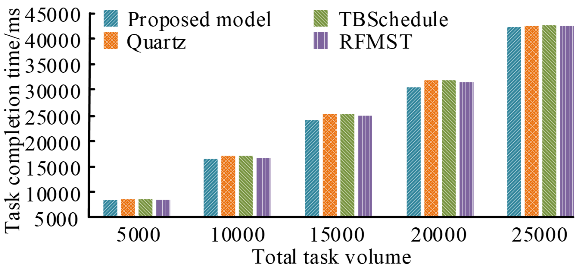
In terms of experimental hardware configuration, the server used Dell PowerEdge R740, equipped with an Intel Xeon Gold 6248R processor, a 128GB DDR4 memory, and a 4TB SAS hard drive. In terms of software configuration, the operating system adopted Windows Server 2019, Hadoop distributed system software and related task scheduling tools were installed, and the JavaSE 11 programming language was selected. To validate the performance of the proposed fault-tolerant task scheduling model, in the experiments carried out the Quartz, TBSchedule, and Result Feedback for Multiple Scheduling Trips (RFMST) models were compared. The comparison of TCT for the four models under different task volumes is illustrated in Figure 6. In Figure 6(a), the number of scheduling trips was 5. As the total number of tasks rose, the TCT value gradually increased. When the total task volume was 25000, the proposed model had a TCT of 43215 ms, the Quartz model had a TCT of 43651 ms, the TBSchedule model had a TCT of 43597 ms, and the RFMST model had a TCT of 43495 ms. As it can be seen in Figure 6(b), when the number of scheduling trips was 10 and the total number of tasks was 25000, the TCT of the proposed model was 42765 ms, the TCT of the Quartz model was 42974 ms, the TCT of the TBSchedule model was 42981 ms, and the TCT of the RFMST model was 42954 ms. As shown in Figure 6(c), when the number of scheduling trips was 15 and the total number of tasks was 25000, the TCT of the proposed model was 42471 ms, the TCT of the Quartz model was 42761 ms, the TCT of the TBSchedule model was 42794 ms, and the TCT of the RFMST model was 42671 ms. The outcomes prove that the proposed fault-tolerant task scheduling model outperformed the Quartz, TBSchedule, and RFMST models with regard to TCT under different task volumes and scheduling times.



(a) The number of scheduling trips is 5



(b) The number of scheduling trips is 10



(c) The number of scheduling trips is 15

Figure 6. Comparison of TCT under different task volumes

After rescheduling, the change in processor time conflict is represented in Figure 7. As shown in Figure 7, processor 5 and processor 10 malfunctioned. Except for the first processor, all the other processors had varying degrees of time conflicts. In the event of a fixed total number of tasks, the time conflict of processors would rise with the rise of the number of processors and reach its peak on the last processor. The obtained results proved that the fault tolerance theory is correct.

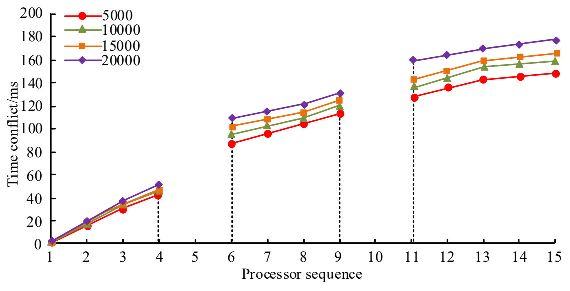
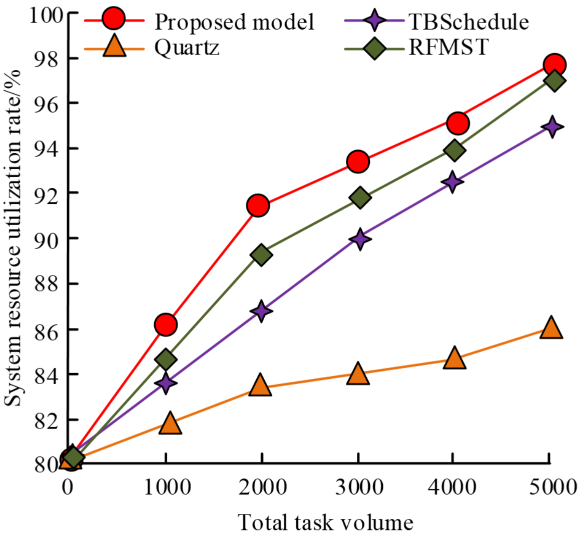
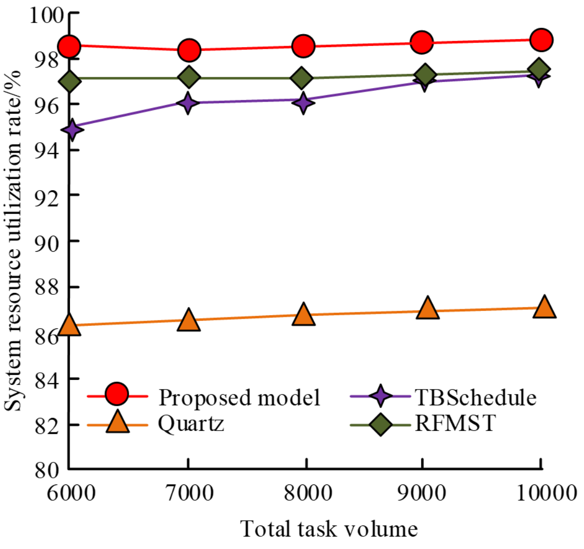


Figure 7. Changes in processor time conflict

The comparison of system resource utilization under different task volumes is shown in Figure 8:



(a) The total task quantity is [0,5000]

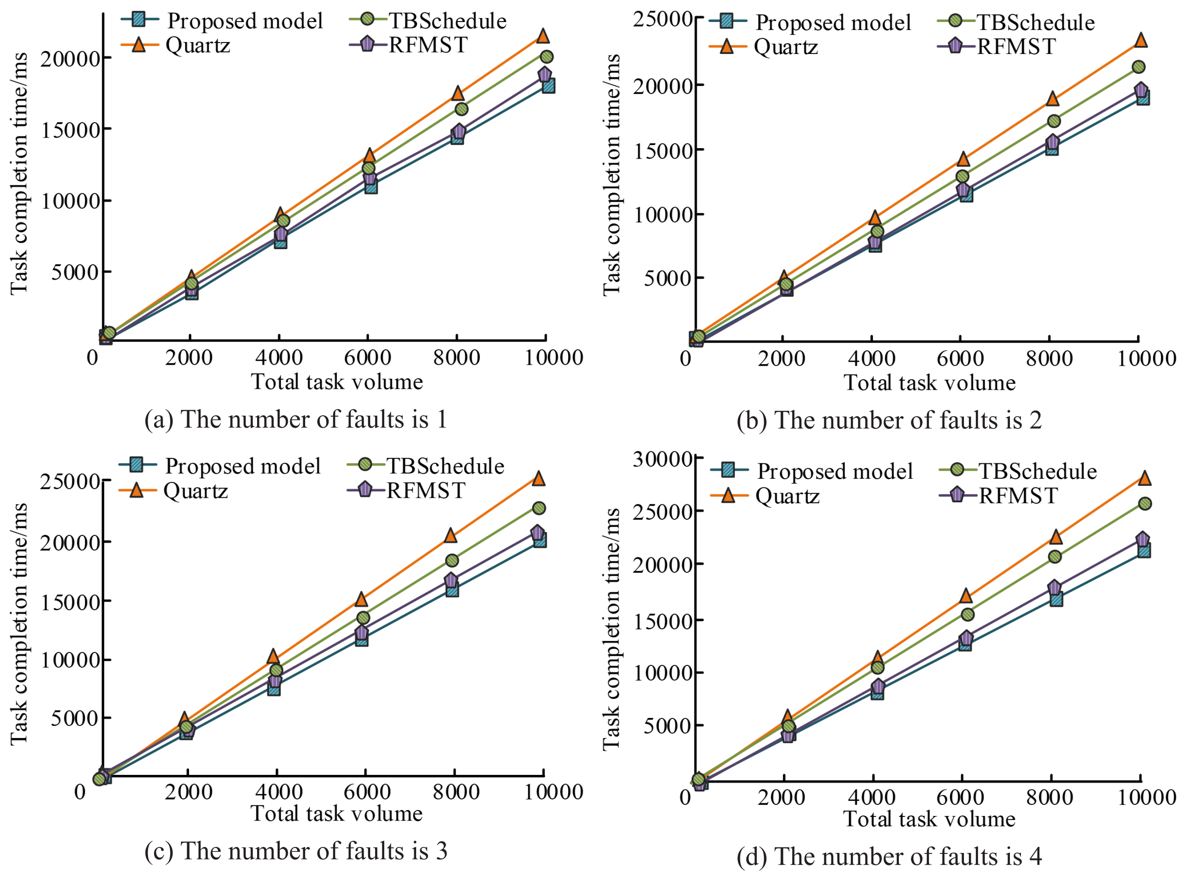


(b) The total task quantity is [6000,10000]

Figure 8. Comparison of the system resource utilization rate under different task volumes

In Figure 8(a), the total task volume was [0, 5000]. As the total number of tasks in the system increased, the system could more effectively utilize its resources, that is, the resource utilization rate increased. The system resource utilization rate for the proposed model reached a maximum of 97.9%, higher than the value of 86.1% obtained by the Quartz model, the value of 95.3% achieved by the TBSchedule model, and the value of 97.1% for the RFMST model. In Figure 8(b), the total task volume was [6000, 10000], and as the total task volume increased, the resource utilization rate for the proposed model fluctuated around 98.5%. When the total task volume was 10000, the system resource utilization rates of the proposed model, Quartz model, TBSchedule model, and RFMST model were 99.1%, 85.3%, 97.6%, and 98.1%, respectively. The outcomes show that the proposed model exhibited significant advantages with regard to system resource utilization under high task volumes, outperforming other comparative models.

The comparison of the task completion duration for the employed models under different numbers of faulty machines is shown in Figure 9. In brief, the Quartz model obtained the longest TCT, followed by the TBSchedule model and RFMST model, while the research model achieved the shortest TCT. As shown in Figure 9(a), when the number of faulty processors was 1 and the total number of tasks was 10000, the proposed model only needed 17954 ms, which was 361 ms shorter than the value obtained by the RFMST model. As it can be seen in Figure 9(b), when the total number of tasks was 10000 and the number of faulty processors was 2, the proposed model's task completion time was only 18413 ms, which was lower than the value of 18880 ms obtained by the RFMST model. As shown in Figure 9(c), when the total number of tasks was 10000 and the number of faulty processors was 3, the RFMST model needed 19964 ms, and the proposed model needed 19352 ms. In Figure 9(d), the number of faulty processors was 4. When the total number of tasks was 10000, the proposed model needed 21571 ms, which was lower than the value of 22492 ms obtained by the RFMST model. The results indicated that, under different numbers of faulty machines, the proposed model obtained a shorter TCT and featured a higher efficiency in comparison with the other three models.



(a) The number of faults is 1

(b) The number of faults is 2

(c) The number of faults is 3

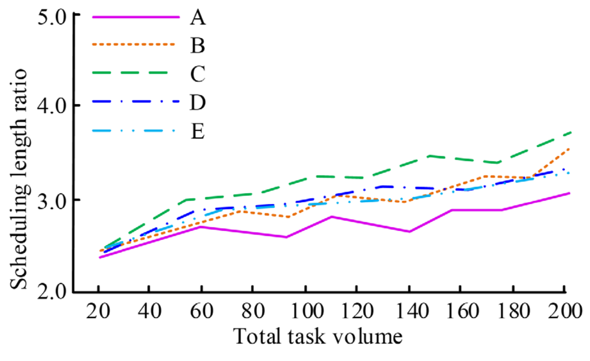
(d) The number of faults is 4

Figure 9. Comparison of task completion time under different numbers of faulty machines

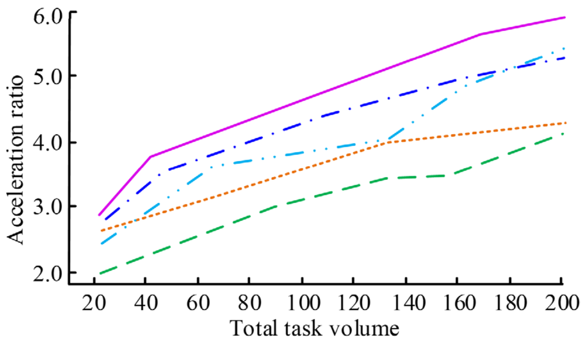
4.2 Analysis of the Optimization Effect of the Scheduling Sequence

To confirm the effectiveness of the GA-based scheduling sequence optimization method proposed in this paper (marked as method A), the ascending-order sorting method (marked as method B), the descending-order sorting method (marked as method C), the algorithm proposed by Ge et al. (2024) (marked as method D), and the algorithm proposed by Singh & Chaturvedi (2024) (marked as method E) were studied, and the scheduling length ratio and acceleration ratio were used as evaluation indicators. The experimental parameter settings are as follows: there were 5 processors in the distributed system, with a total of 10 tasks, a task computation time of {1, 2, 3, 4, 5}, and a processor computing power of {1, 2, 4, 8, 16}. For the simulation experiment, the basic parameters of GA were set as follows: a population size of 80, a total number of 1000 iterations, a crossover probability of 0.7, and a mutation probability of 0.06. The comparison results for the employed methods under different task volumes are illustrated in Figure 10. In Figure 10(a), as the number of tasks increased, the scheduling length ratio gradually increased. When the number of tasks was 200, the scheduling length ratio for method A was 3.21, the scheduling length ratio for method B was 3.45, the scheduling length ratio for method C was 3.63, the scheduling length ratio for method D was 3.34, and the scheduling length ratio for method E was 3.32. In Figure 10(b), the acceleration ratio increased with the rise in the number of tasks. When the number of tasks was 200, the acceleration ratio for method A was 5.92, the acceleration ratio for method B was 4.21, the acceleration ratio for method C was 4.13, the acceleration ratio for method D was 5.28, and the acceleration ratio for method E was 5.42. The results indicated that the GA-based scheduling sequence optimization method featured a high efficiency and effectiveness in dealing with distributed system task scheduling problems.

The comparison results for the employed methods under different processor performance levels are depicted in Figure 11. In Figure 11(a), as the performance coefficient of the employed processor rose, the scheduling length ratio gradually increased, and method A obtained the smallest scheduling length ratio.

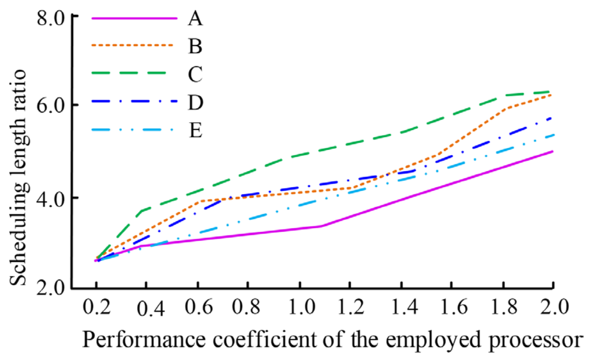


(a) Scheduling length ratio

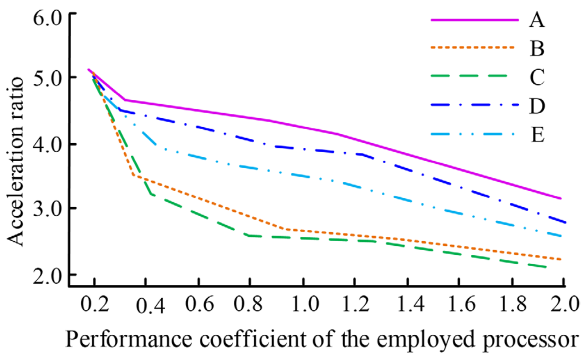


(b) Acceleration ratio

Figure 10. Comparison of the employed methods under different task volumes



(a) Scheduling length ratio



(b) Acceleration ratio

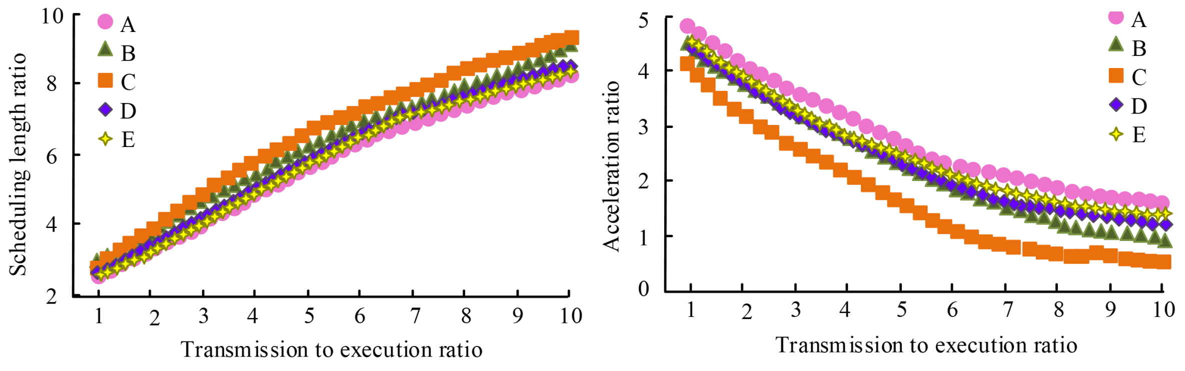
Figure 11. Comparison of the employed methods under various processor performance coefficients

When the performance coefficient of the employed processor was 2, the scheduling length ratio for method A was 4.92. In Figure 11(b), the acceleration ratio decreased with the increase of the processor performance difference coefficient, and method A obtained the highest acceleration ratio. When the performance coefficient of the processor was 2, the acceleration ratio for method A was 3.15.

The comparison of the employed methods under various transmission to execution ratios is illustrated in Figure 12. In Figure 12(a), as the transmission execution ratio rose, the scheduling length for each method gradually increased. When the transmission to execution ratio was 10, the scheduling length for method A was 8.15, for method B it was 9.03, for method C it was 9.11, for method D it was 8.25, and for method E it was 8.21. In Figure 12(b), as the transmission to execution ratio increased, the acceleration ratio for each method gradually decreased. When the transmission to execution ratio was 10, the acceleration ratio for method A was 1.47, which was about 0.58 higher than the value obtained by method B. The results indicated that the GA-based scheduling sequence optimization method proposed in this paper had a good performance in terms of processor performance coefficients and transmission to execution ratios, and the values of its scheduling length ratio and acceleration ratio were superior to those obtained by the other four methods.

5. Conclusion

A fault-tolerant task scheduling model and a GA-based scheduling sequence optimization method were proposed to address the reliability and stability issues faced by task scheduling in distributed systems during failures. By analyzing the star-shaped distributed system and combining the processes of task segmentation, allocation, and result processing, the optimization of task scheduling was achieved. The obtained results indicated that the proposed model performed faster than the Quartz, TBSchedule, and RFMST models. In terms of system resource utilization, the proposed model featured significant advantages under high task volumes. So, when the total task volume was 10000, the system resource utilization rates for the proposed model, the Quartz model, the TBSchedule model, and the RFMST model were 99.1%, 85.3%, 97.6%, and 98.1%, respectively. Further on, when the number of faulty processors was 4 and the total number of tasks was 10000, the proposed model`s time consumption was 21571 ms, which was lower than the value obtained by the RFMST model. When the number of tasks was 200, the GA-based scheduling sequence optimization method obtained a scheduling length ratio of 3.21 and an acceleration ratio of 5.92, values which were superior to those obtained by the other four methods. Research showed that the proposed fault-tolerant task scheduling model and the GA-based scheduling sequence optimization method played a significant role in improving the reliability, stability, and performance of distributed systems. However, there are some limitations to this study. The distributed system architecture based on the adopted star structure is relatively specific, and the integration mode for the cloud edge architecture is not considered, which limits the application of the research results to a wider range of scenarios to a certain extent. At the same time, the impact of more complex network environments and of the dynamic changes of distributed systems on task scheduling is not considered enough. Future research directions could focus on exploring the



(a) Scheduling length ratio (b) Acceleration ratio

Figure 12. Comparison results for the employed methods under different processing speed ratios

impact of implementing the cloud edge architecture in distributed system task scheduling, and on a in-depth analysis of its advantages in improving system performance and fault tolerance, so as to expand the research boundaries with regard to distributed system architecture and enhance the adaptability of the research results to complex practical scenarios. In addition, it is necessary to further explore how to better adapt the GA to complex and changeable practical application scenarios and optimize the algorithm to improve its adaptability and performance under different conditions.

Acknowledgements

This work was supported through the Research talent launch start-up project of Fuyang Normal University under grant no. 2019kyqd0018 and the Anhui Province University key research project under grant no. 2022AH052820.

REFERENCES

Abedpour, K., Hosseini Shirvani, M. & Abedpour, E. (2024) A genetic-based clustering algorithm for efficient resource allocating of IoT applications in layered fog heterogeneous platforms. *Cluster Computing*. 27(2), 1313-1331. https://doi.org/10.1007/s10586-023-04005-x.

Benamor, H., Yousfi, M., Chrifi Alaoui, L. et al. (2023) New Approach for Estimating the Input Fault Based on the Sliding Window Identification Technique of the SISO ARX-Laguerre Multimodel. *Studies in Informatics and Control*. 32(4), 115-127. https://doi.org/10.24846/v32i4y202311.

Çavdar, M. C., Korpeoglu, I. & Ulusoy, Ö. (2024) A Utilization Based Genetic Algorithm for virtual machine placement in cloud systems. *Computer Communications*. 214(1), 136-148. https://doi.org/10.1016/j.comcom.2023.11.028.

Ge, Y.-F., Wang, H., Cao, J. et al. (2024) Privacy-preserving data publishing: an information-driven distributed genetic algorithm. *World Wide Web*. 27, art. no. 1. https://doi.org/10.1007/s11280-024-01241-y.

George, N. Kadan, A. B. & Vijayan, V. P. (2023) Multi-objective load balancing in cloud infrastructure through fuzzy based decision making and genetic algorithm based optimization. *IAES International Journal of Artificial Intelligenc*e. 12(2), 678-685. https://doi.org/10.11591/ijai.v12.i2.pp678-685.

Indhumathi, R., Amuthabala., K, Kiruthiga, G. et al. (2023) Design of Task Scheduling and Fault Tolerance Mechanism Based on GWO Algorithm for Attaining Better QoS in Cloud System. *Wireless Personal Communications*. 128(4), 2811-2829. https://doi.org/10.1007/s11277-022-10072-x.

Ismael, H. R., Abdulrahman, L. M., Rashid, Z. N. et al. (2022) Web Technology Grounded Effects of Task Scheduling in Distributed and Cloud Systems. *Journal of Smart Internet of Things*. 1, 196-218. https://doi.org/10.2478/jsiot-2022-0013.

Jeyaraman, J., Bayani, S. V. & Malaiyappan, J N A. (2024) Optimizing Resource Allocation in Cloud Computing Using Machine Learning. *European Journal of Technology*. 8(3), 12-22. https://doi.org/10.47672/ejt.2007.

Krishnamurthy, O. (2023) Genetic Algorithms, Data Analytics and it's applications, Cybersecurity: verification systems. *International Transactions in Artificial Intelligence*. 7(7), 1-25.

Li, Z, Yu, H, Fan, G. et al. (2023) Cost-Efficient Fault-Tolerant Workflow Scheduling for Deadline-Constrained Microservice-Based Applications in Clouds. *IEEE Transactions on Network and Service Management*. 20(3), 3220-3232. https://doi.org/10.1109/TNSM.2023.3241450.

Lian, D., Mo, P., D' Ariano, A. et al. (2024) Energy-saving time allocation strategy with uncertain dwell times in urban rail transit: Two-stage stochastic model and nested dynamic programming framework. *European Journal of Operational Research*. 317(1), 219-242. https://doi.org/10.1016/j.ejor.2024.03.015.

Liu, Z. (2022) Optimization of Computer-aided Decision-making System for Railroad Traffic Dispatching Command. *Computer-Aided Design and Applications*. 19(S4), 123-134. https://doi.org/10.14733/cadaps.2022.S4.123-134.

Long, Y., Liu, S., Qiu, D. et al. (2023) Local Path Planning with Multiple Constraints for USV Based on Improved Bacterial Foraging Optimization Algorithm. *Journal of Marine Science and Engineering*. 11(3), art. no. 489. https://doi.org/10.3390/jmse11030489.

Lu, Q., Nguyen, K. & Huang, C. (2023) Distributed parallel algorithms for online virtual network embedding applications. *International Journal of Communication Systems*. 36(1), 4325-4367. https://doi.org/10.1002/dac.4325.

Malti, A. N., Hakem, M & Benmammar, B. (2024) A new hybrid multi-objective optimization algorithm for task scheduling in cloud systems. *Cluster Computing*. 27(3), 2525-2548. https://doi.org/10.1007/s10586-023-04099-3.

Naderi, E., Mirzaei, L., Pourakbari-Kasmaei, M. et al. (2024) Optimization of active power dispatch considering unified power flow controller: application of evolutionary algorithms in a fuzzy framework. *Evolutionary Intelligence*. 17(3), 1357-1387. https://doi.org/10.1007/s12065-023-00826-2.

Păun A.-M., Coandă, H.-G., Mincă, E et al. (2024) Improved Multi-objective Genetic Algorithm Used to Optimizing Power Consumption of an Integrated System for Flexible Manufacturing. *Studies in Informatics and Control*. 33(1), 27-36. https://doi.org/10.24846/v33i1y202403.

Palle, R R. (2023) Evolutionary Optimization Techniques in AI: Investigating Evolutionary Optimization Techniques and Their Application in Solving Optimization Problems in AI. *Journal of Artificial Intelligence Research*. 3(1), 1-13.

Pham, V. H. S., Nguyen Dang, N. T. & Nguyen, V N. (2024) Enhancing engineering optimization using hybrid sine cosine algorithm with Roulette wheel selection and opposition-based learning. *Scientific Reports*. 14(1), art. no. 694. https://doi.org/10.1038/s41598-024-51343-w.

Reghenzani F., Guo, Z., Fornaciari, W. (2023) Software Fault Tolerance in Real-Time Systems: Identifying the Future Research Questions. *ACM Computing Surveys*. 2023, 55(14s), Art. ID 306. https://doi.org/10.1145/3589950.

Shiva Rama Krishna, M. & Mangalampalli, S. (2023) A Novel Fault-Tolerant Aware Task Scheduler Using Deep Reinforcement Learning in Cloud Computing. *Applied Sciences*. 13(21), art. no. 12015. https://doi.org/10.3390/app132112015.

Singh, G. & Chaturvedi, A K. (2024) Hybrid modified particle swarm optimization with genetic algorithm (GA) based workflow scheduling in cloud-fog environment for multi-objective optimization. *Cluster Computing*. 27(2), 1947-1964. https://doi.org/10.1007/s10586-023-04071-1.

Siyadatzadeh, R., Mehrafrooz, F., Ansari, M. et al. (2023) ReLIEF: A Reinforcement-Learning-Based Real-Time Task Assignment Strategy in Emerging Fault-Tolerant Fog Computing. *IEEE Internet of Things Journal*. 10(12), 10752-10763. https://doi.org/10.1109/JIOT.2023.3240007.

Vargas, G. A. D., Mosquera, D. J. & Trujillo, E R. (2024) Optimization of Topological Reconfiguration in Electric Power Systems Using Genetic Algorithm and Nonlinear Programming with Discontinuous Derivatives. *Electronics*. 2024, 13(3), Art. ID 616. https://doi.org/10.3390/electronics13030616.

Xu, H., Xu, S., Wei, W. et al. (2023) Fault tolerance and quality of service aware virtual machine scheduling algorithm in cloud data centers. *The Journal of Supercomputing*. 79(3), 2603-2625. https://doi.org/10.1007/s11227-022-04760-5.

Zangana, H. M. & Zeebaree, S. R. M. (2024) Distributed Systems for Artificial Intelligence in Cloud Computing: A Review of AI-Powered Applications and Services. *International Journal of Informatics, Information System and Computer Engineering (INJIISCOM)*. 5(1), 11-30. https://doi.org/10.34010/injiiscom.v5i1.11883.