Designing and Implementation of Failure-Aware Based Approach for Task Scheduling in Grid Computing

Manjeet Singh[1], **Javalkar Dinesh Kumar**[2]

[1]Research Scholar, Department of Computer Science & Engineering, Lingaya's Vidyapeeth, Faridabad, India, 19phcs05w@lingayasvidyapeeth.edu.in
[2]Assistant Professor, Department of Electronics & Communication Engineering, Lingaya's Vidyapeeth, Faridabad, India, javalkardinesh@gmail.com

***Correspondence:** Manjeet Singh, Email: 19phcs05w@lingayasvidyapeeth.edu.in

ABSTRACT- Grid computing makes large-scale computations easier to handle. In heterogeneous systems like grid computing, failure is inevitable. Because of the volume and diversity of the resources, scheduling algorithm is among the most difficult challenges to overcome in grid computing. To reduce the make-span of the job to be executed a thorough understanding of scheduling in grid is important. Say there are two computing nodes that aren't being used right now. The scheduler may choose the node that has higher computing strength (for example, higher CPU speed, higher free memory), even though this node may also have high potential of failure. High potential of failure refers to the possibility of the failure occurring at execution time, resulting in the decrease of system performance. Therefore, awareness of failure is also very important in scheduling. This work proposes and implements a failure-aware scheduling method to schedule the tasks which uses both performance factors and failure factors of resources while making scheduling decision. The proposed algorithm is analyzed over various performance matrices and it shows considerably improved performance over existing algorithm.

General Terms: Grid Computing, Large Scale Computing, Scheduling Algorithms.
Keywords: Checkpoint, Failure, Fault Tolerance, QoS, Recovery, Resource, Reliability, Scheduling.

ARTICLE INFORMATION
Author(s): Manjeet Singh and Javalkar Dinesh Kumar
Received: 03/08/2022; **Accepted**: 14/09/2022**; Published**: 18/09/2022:
E- ISSN: 2347-470X
Paper Id: IJEER-220737
Citation: 10.37391/IJEER.100339
Webpage-link:
https://ijeer.forexjournal.co.in/archive/volume-10/ijeer-100339.html

Publisher's Note: FOREX Publication stays neutral with regard to jurisdictional claims in Published maps and institutional affiliations.

1. INTRODUCTION

The name Grid was used as a metaphor for a Power Grid, which provides continuous, ubiquitous, dependable, and fair access to electricity irrespective of starting point. Grids make it possible to share, select, as well as aggregate a wide range of resources, such as powerful computers, memory devices, sources of data, and equipment, that are geographically dispersed and owned by a variety of organizations. This allows grids to solve multiple problems in the fields of science, engineering, and commerce. In the beginning it link very high performance computers that were located in different parts of the world, but now it has grown to include much more than its initial remit [1]. Grid computing is a new computing model that does large-scale computations by connecting a network of connected processors or resources. Multiple resources are required since multiple jobs might execute at the same time or a grid system can solve multiple types of problems which may require different type of computing resources.

Fault tolerance becomes very important attribute in Grid computing since individual Grid resource reliability cannot always be guaranteed; additionally, as resources are used outside of organizational bounds, it is more difficult to ensure the behavior of resource. A system failure happens when the behavior of the system deviates from its specification, which is caused by system flaws [2].

Preventing system failures is one technique to increase system reliability. Fault prevention is the name for this method. Another approach is to provide the desired service notwithstanding any flaws. Fault tolerance is the term for this. Fault tolerance must be built into the system to guarantee uninterrupted service because no amount of fault avoidance can eliminate all possible failures [3].

The goal of fault tolerance is to ensure that expected services are delivered irrespective of the existence of fault-caused defects inside the system [7]. Failure, Faults and Errors are identified and resolved while the system continues to provide appropriate service [8]. Hence, the goal to deliver QoS (Quality of Service) is achieved with the help of fault tolerant mechanism.

When a work is submitted to the Grid for computation, the Resource Management System breaks it into roughly equal-sized subtasks. All of these subtasks are assumed to be independent of one another here. Grid information server maintains information of all resources into the system and schedule task as per their availability and various performance and failure related parameters. *Figure 1* shows an environment of Grid Computing System.

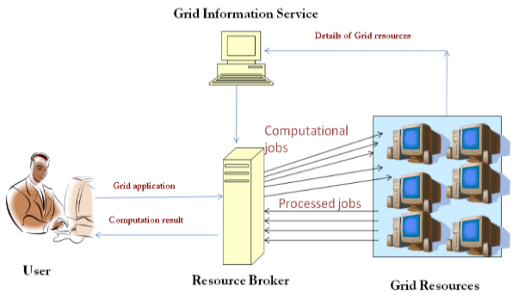

Figure 1: Grid Computing Environment [36]

Failure is inevitable in a heterogeneous system like the Grid. Failures of system resources have adverse effects on application's performance [9]. Failures can make a process run slower than normal or even stop it. A resource/node failure can be caused by failure of any of the component in Grid which may be a processor, memory, network connection and application/software [21]. Thus, failure information, in addition to other performance-related factors, should be taken into account when making scheduling decisions. Several algorithms have been proposed to maximize reliability and minimize makespan. However, these objectives cannot be achieved concurrently. There is a trade-off between reducing makespan and improving reliability. Generally, improving reliability of the system incurs some overheads, making applications take more time to finish. Therefore, a strategy is to minimize executing time of tasks under failure-prone condition.

2. LITERATURE REVIEW

The Queen-bee algorithm is used by Zahra Pooranian et al. [11] to solve the scheduling problem, as well as the comparison is made to different meta-heuristic algorithms. Furthermore, as compared to previous methods, the suggested algorithm minimizes both computation time and makespan. A FI-based scheduling strategy was presented by Jairam Naik K et al. [12], which picks resources based on response time and fault indicator. H. B. Prajapati et al. [13] provided an outline of Grid computing and explain the numerous subsystems that make it possible. In addition, the study looks at resource and application scheduling concepts, as well as scheduling method classification. In grid computing, Ritu Garg et al. [16] suggested a scheduling technique for dependent tasks. P. Keerthika and P. Suresh [19] presented a load-balancing-based scheduling technique that improves resource efficiency. Rakesh Kumar et al. [20] discussed Virtualization, Cloud, Grid, and Cluster Computing along with their characteristics, advantages, shortcomings, benefits, and downsides along with a comparison of cloud, cluster, and grid computing, as well as a comparison of grid and cluster computing.

A fault-tolerant scheduling technique with QoS constraints was proposed by S. Haider et al. [25]. A job scheduling system for Hadoop was developed by Mbarka Soualhia et al. [27]. Mark Baker et al. [28] discussed the web based techniques for grid environment. Muhanad et al. [31] have proposed two metahuristic scheduler for job scheduling. P. Kathalkar and A. V. Deorankar [33] discusses the checkpoint restore approach and the many mechanisms offered by different authors to increase the efficiency and performance of system. The main reason behind this study is to learn more about the process of check pointing and the classification of check pointing methods. Ankita et al. [35] uses genetic algorithm to solve and optimize a multi-objective GSP (Grid Scheduling Problem). Pranit Sinha et al. [37] study proposes methods for boosting the grid system's efficacy by combing two scheduling techniques. This hybrid scheduler has the potential to speed up the grid system's execution time. P. Kumari et al. [38] surveyed the fault tolerant techniques H. Eluri et al. [39] develop an energy saving scheme in Micro-Grid environment using fuzzy logic controller. L. Jenila et al. [40] proposed a scheduling algorithm for wireless multimedia sensor. S. Kulkarni et al. [41] have done the performance analysis for fault tolerant operation of PMSM to increase system reliability. N. Thapliyal et al. [42] developed a load balance min-min scheduling algorithm for load balancing in cloud computing inspired by the foraging activity of honey bees.

Table 1: Comparison of Various Existing Scheduling Approaches

Reference	Description of Scheduling Approach
H. Sajedi et al. [14]	offers the CUckoo-Genetic System (CUGA), a task scheduling system for grids that reduces machine completion times and is based on the cuckoo optimization algorithm (COA) as well as the genetic algorithm (GA)
J. Shanthini et al. [17]	developed a hybrid scheduling model for independent task based on best gap search and apparent tardiness cost indexing method.
M. K. Bhatia [22]	Briefed about various scheduling algorithms like, OLB (opportunistic load balancing), Min-Min algorithm, Max-Min algorithm Surffrage algorithm, GA, SA, GSA and Tabu search etc for grid computing environment.
M.T. Younis et al. [23]	Devised a scheduling approach for scheduling of independent jobs in grid computing, which is based on the genetic algorithm.
H. Idris et al. [26]	suggested an algorithm for grid fault tolerance scheduling which is based on ant colony optimization technique
Sophiya Sheikh et al. [29]	Propose a method for resource load balancing that commits advanced resource reservations to tasks in order to reduce load imbalance on nodes with the smallest makespan
J. Natarajan [30]	Proposed effective novel Backfilling technique to solve the Task Scheduling problem. Tasks are split into numerous threads for processing depending on their duration. In the core concept of "gang scheduling," several thread jobs are processed
T. V. Long et al. [32]	Described CAPE, is a checkpoint-based method for seamlessly interpreting and performing OpenMP programs on distributed applications
B. Anitha et al. [34]	Proposed a new scheduling technique based on heuristic approach for independent task.

The above comparison describes (see *Table 1*) various scheduling approaches. The strategy proposed in this research is based on failure factors and performance parameters both, which is different from the above methods. Hence the method devised is novel in nature.

3. METHODOLOGY

Due to the occurrence of failures in the Grid computing environment, a technique to determine faults and resolving the consequences of fault is required. In grid computing settings, fault tolerance mechanisms include check pointing, replication, and so on. For fault tolerance, we use a comprehensive checkpoint/restart mechanism. The Grid computing system used is based on the following assumptions:

- Nodes in Grid System have varying performance and failure related parameters.
- Jobs assigned are self-contained, meaning they can be completed in any order.
- The Weibull distribution governs the time between Grid node breakdowns.
- During the lifetime of the system, the failure rate of can grow, drop, or remain same.

Grid System is made up of different resources with varying failure rates and performance parameters, and the jobs to be carried out are assumed to be free from one another. The Weibull distribution is used to calculate the duration between failures. A node's failure rate may grow or decrease during execution, and can remain constant [6]. Several research [9-10, 18, 24] have looked at the period between failures and analyze the failure distribution and get Weibull distribution fit where failure rate is not constant i.e. increase or decrease with time. Some discover falling hazard rates, while others find flat or increasing hazard rates. There are two parameters (α and ß) in the Weibull failure distribution: scale and shape parameter respectively [10]. Symbol and notation used are given below in *table 1*.

Table 2: Various Symbol/Notation

Symbol	Details
α	Scale Parameter
ß	Shape Parameter
f(t)	Probability function of Density
F(t)	Function of Cumulative Distribution
O_F	Checkpoint Overhead
T_{Re}	Re-computation Cost
R_F	Time required to recover
k	Coefficient of Time for Recomputing
n(t)	Frequency function
t_i	*i*th checkpoint placement
E[W]	Expected wasted time
R	Total number of failures

During execution if a resource crashes all applications that are in execution need to restart from scratch. In this work, the Full Checkpoint Scheme (a fault-tolerant method) is used to provide application reliability. Full checkpointing is the method that saves the entire application state. The checkpoint/restart system has two critical states: first one is checkpointing and second is recovery. In first state a snapshot of running application is taken that is utilised in second state in case of a failure for recovery from the last execution point. Checkpoint overhead is the time it takes to save a state (O_F). The task execution is resumed when we reload the saved snapshot. The recovery state has two costs: the time spent loading the most recent snapshot (R_F) and the time spent in re-computing (T_{Re}). As a result of the checkpoint system, time can be saved by not re-computing task from scratch after a failure. *Figure 2* and *Figure* 3 show the behavior of the whole checkpoint/restart paradigm [6].

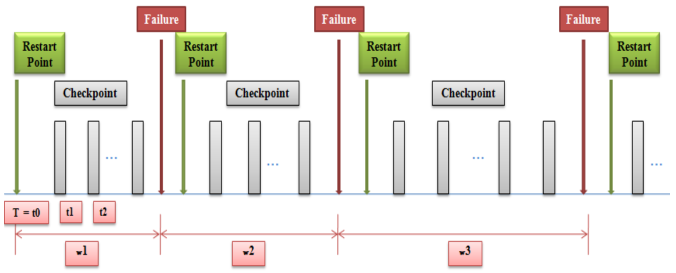

Figure 2: Mechanism of Checkpoint/Restart as stochastic renewal reward process

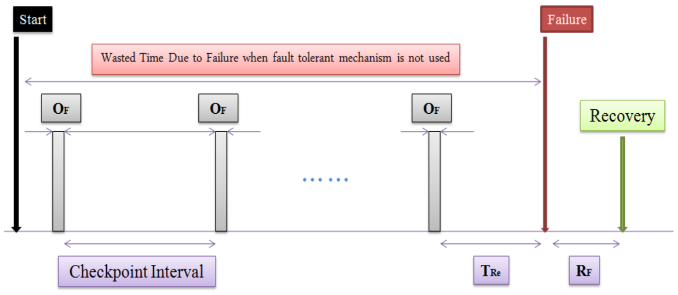
Start
Failure
Wasted Time Due to Failure when fault tolerant mechanism is not used
OF
OF
OF
Recovery
Checkpoint Interval
TRe
RF

Figure 3: Behavior of Full Checkpoint/Restart Model

The fundamental concept is to determine the anticipated amount of time lost as a result of errors, and then make use of this information to recalculate the amount of processing power actually available for resources. Later this new capacity/ processing power is used for making the scheduling decision. The wasted time is calculated as follows:

$$Wasted\ Time\ (W(T)) = (Checkpointing\ Overhead + Recomputing\ Time + Recovery\ Cost\ of\ Checkpoint) \quad (1)$$

The checkpointing frequency function is shown in *Eq. (2)* and hence number of checkpoints to be taken during the execution of an application can be computed with the help of *Eq. (3)* as given below:

$$\int_{t_{i-1}}^{t_i} n(t).dt = 1 \quad (2)$$

Where t_i (i=1, 2 …) and t_0 =0.

$$\int_0^T n(t).dt \quad (3)$$

Where, n(t) can be calculated by *Eq. (4)*.

F(t) is Cumulative Distribution Function (CDF) given by *Eq. (5)* and f(t) is Probability Density Function (PDF) given by *Eq. (6)* for Weibull Distribution [6].

$$n(t) = \sqrt{\frac{k}{O_F} \cdot \frac{f(t)}{1-F(t)}} \quad (4)$$

$$F(t) = 1 - e^{-(t/\alpha)^{\beta}} \tag{5}$$

$$f(t) = \left(\frac{\beta}{\alpha}\right) . \left(\frac{t}{\alpha}\right)^{\beta - 1} . e^{-(t/\alpha)^{\beta}} \tag{6}$$

Using *Eq. (5)* and *(6)*, *Eq. (4)* can be written as *Eq. (7)* given below:

$$n(t) = \sqrt{\frac{k}{O_F}} . \left(\frac{t}{\alpha}\right)^{\frac{\beta-1}{2}} . \sqrt{\frac{\beta}{\alpha}} \tag{7}$$

Using number of checkpoint during execution of the application (calculated by *Eq. (3)*) and overhead of one checkpoint O$_F$, we can calculate the total checkpoint overhead on the system, as shown in *Eq. (8)* below:

$$Checkpointing\ Overhead = O_F \int_0^T n(t).dt \tag{8}$$

The Re-computation time is calculated using Eq. (9), where k is the coefficient for re-computing time and its, value varies from 0 to 1.

$$T_{Re} \approx \frac{k}{n(T)} \tag{9}$$

Using *Eq. (8)* and *Eq. (9)* wasted time can be calculated as given in *Eq. (10)*. So, rewrite the *Eq. (1)* as *Eq. (10)* by putting these values.

$$W(T) = O_F \int_0^T n(\tau).d\tau + \frac{k}{n(T)} + R_F \tag{10}$$

Hence, the expected wasted time during execution of an application can be written as in *Eq. (11)* below:

$$E[W] = \int_0^{\infty} [W(T)].f(t).dt \tag{11}$$

Putting the value of *Eq. (10)*, the *Eq. (11)* can be rewritten as *Eq. (12)*:

$$E[W] = \int_0^{\infty} \left[O_F \int_0^t n(\tau).d\tau + \frac{k}{n(t)} + R_F\right].f(t).dt \tag{12}$$

Hence, *Eq. (12)* can be used for calculating the total expected wasted time in case of failure and full checkpoint fault tolerant mechanism.

4. ALGORITHM

The algorithm will arrange T number of jobs among P nodes/resources in order to achieve the shortest possible makespan. The proposed algorithm is divided into two parts. Part A of algorithm is used to calculate the new node capacity of resources with respect to their failure and performance factors. Part B of algorithm schedule task according to new node capacity to minimize the job execution time. *Figure 4* below demonstrates the flowchart of proposed algorithm.

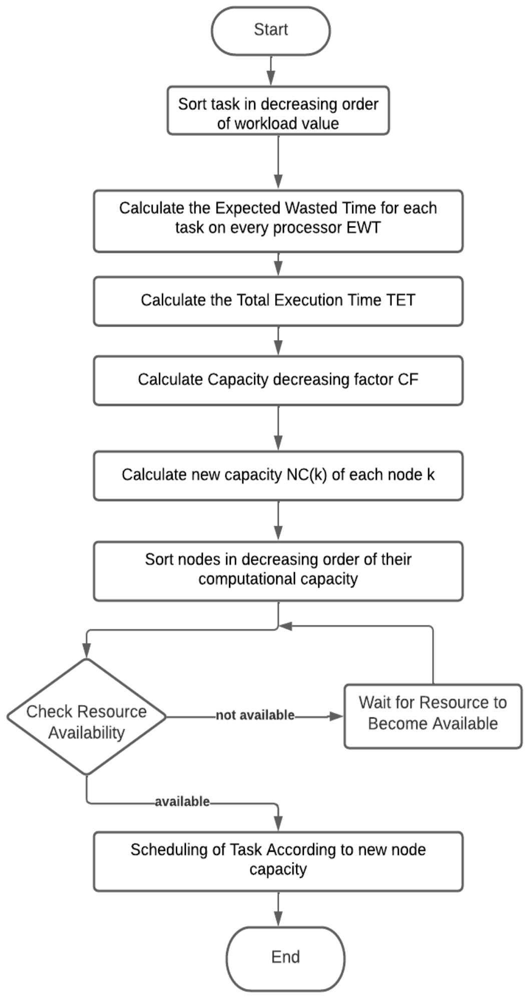
Start
Sort task in decreasing order of workload value
Calculate the Expected Wasted Time for each task on every processor EWT
Calculate the Total Execution Time TET
Calculate Capacity decreasing factor CF
Calculate new capacity NC(k) of each node k
Sort nodes in decreasing order of their computational capacity
Check Resource Availability
not available
Wait for Resource to Become Available
available
Scheduling of Task According to new node capacity
End

Figure 4: Flowchart for Proposed Scheduling Algorithm

Various parameters used in algorithm are given as under:

- Assume that t[T] is an array, and that t(i) represents the task's size.
- Let p[P] be an array, and p(i) be the node i's initial computing capacity.
- Let EXT[T][P] be the execution time, and EXT[i][j] denote the processing time for task i on resource j in the absence of failures and fault tolerance.
- Let T*P order matrix EWT[T][P], with EWT[i][j] denoting the Expected Wasted Time (additional time necessary for job i on processor j) in the event of failure.
- Let TXT[T][P] denote running time with faults and a fault tolerant method.
- The T*P order matrix CF [T][P] specifies the Capacity recomputing factor.
- Assume NC[P] is an array, and NC(k) is the node k's recomputed capacity.

Steps for Scheduling Algorithm:

Part A: Recalculating the Node Capacity-

1. Sort task/job in descending order of job size.
2. Calculate the EWT for all the tasks over all nodes/processor.
   EWT=Check pointing Overhead + Re-computation Time + Recovery Time
3. Now we can calculate the TET by adding EWT to EXT (execution time of job without failure and without fault tolerant system).

   TET=EXT + EWT
4. Calculate the CF (Capacity Decrement Factor):

$$CF[i][j] = \frac{TET[i][j]}{EXT[i][j]} \quad (13)$$

5. The first row will properly portrays the behavior of the system and hence used for calculating new capacity.
6. Calculate new capacity NC(k) of each node k:

$$NC[k] = \frac{p[k]}{CF[1][k]} \quad (14)$$

Part B: Scheduling of Task According to new node capacity-
Arrange resources/nodes in descending order of new computing capacity

```
j=1
While (j <= T)
    For i =1 to P
        If resource/node i is not available or is faulty, then
            i++
            Continue
        Else
            Assign job t[j] to the resource/ node with capacity NC[i]
                j++
                If (j > T)
                    Break
                End if
        End if
    End for
    Wait delta time
End while
```

5. RESULTS AND ANALYSIS

The Simulation is done with the help of Matrix Laboratory Tool MATLAB. By randomizing node characteristics in simulation, an 8-node grid computing system is developed. The number of tasks ranges from four to twenty. The scale parameter (α) has a value of 15 while the shape parameter (ß) has a value between 1 and 5. The job size ranges from [50, 1000]. The number of minutes it takes to finish a job/task on a standard machine is the size of the job/task. The coefficient k is considered to have a value of 0.5. The snapshot time for each full snapshot is 2 minutes, and the full checkpoint recovery time is 0.5 minutes. The data used for evaluating the performance of proposed FABS method is system generated.

The suggested failure-aware based scheduling algorithm (FABS) is compared to the Speed-only approach (SOSA) to see how well it performs. While scheduling tasks, SOSA algorithm considers only resource performance parameters. The below given performance metrics are used to evaluate the suggested algorithm's performance [4-5, 15].

Performance Ratio (PR): It's the ratio of the speed-only method's make-span to the make-span of the suggested algorithm's. The proposed algorithm should have a shorter make-span than the speed-only scheduling algorithm for better performance i.e. the value of PR should come out to be greater than 1.

Failure Ratio (FR): It's the proportion of total crashes in the proposed method to total crashes in the existing technique. If the value of FR is less than 1, the proposed failure-aware scheduling approach will perform better.

Average Time to Respond (ATR): Let T be the number of task in a system and TETi be the time system takes to complete work ti, and Arrivali be the time taken for job *i* to arrive in system. The system's average response time is defined as follows:

$$ATR = [\textstyle\sum_{1 \le i \le T} (TET_i - Arrival_i)] \,/\, T \quad (15)$$

The better performance of the system the ATR of Failure-aware scheduling method should be lower than ATR of speed-only scheduling approach.

Throughput: It's the number of task completed in a given time span. Proposed method should have a higher throughput than the speed-only scheduling approach. Throughput is a direct indicator of improved and better system performance.

Performance Improvement Rate (PIR): It details the percentage amount by which the suggested approach (FABS) outperforms the other available algorithm (SOSA).

$$PIR(\%) = \left(\frac{Makespan\ (SOSA) - Makespan\ (FABS)}{Makespan\ (FABS)}\right) \times 100 \quad (16)$$

The various performance metrics parameter values such as makespan, performance ratio, number of failures, failure ratio, average response time, throughput and PIR are given below in *Table 3* to *Table 6* and *Figure 5* to *Figure 11* shows the performance comparison of FABS and SOSA.

Table 3: Makespan and Performance Ratio

Number of Task	Make-span (SOSA)	Make-span (FABS)	PR (FABS)	PIR (FABS)
4	107.1152	80.4794	1.331	33.0964
8	113.9768	84.3085	1.3519	35.1902
12	174.0754	172.8185	1.0073	0.7273
16	263.9208	224.6135	1.175	17.5000
20	314.0572	295.6922	1.0621	6.2109

Table 4: Number of Failure and Failure Ratio

Number of Task	No. of Failure (SOSA)	No. of Failure (FABS)	FR (FABS)
4	39.7355	32.7863	0.8251
8	68.028	61.0682	0.8977
12	132.2089	116.0461	0.8777
16	184.5777	172.1199	0.9325
20	264.9271	249.8278	0.943

Table 5: Average Response Time

Number of Task	ART (SOSA)	ART (FABS)
4	88.5548	76.3694
8	79.7817	78.1088
12	94.2157	92.9473
16	100.5566	99.3666
20	109.6399	108.8908

Table 6: Throughput

Number of Task	Throughput (SOSA)	Throughput (FABS)
4	0.1129	0.1309
8	0.1253	0.128
12	0.1061	0.1076
16	0.0994	0.1006
20	0.0912	0.0918

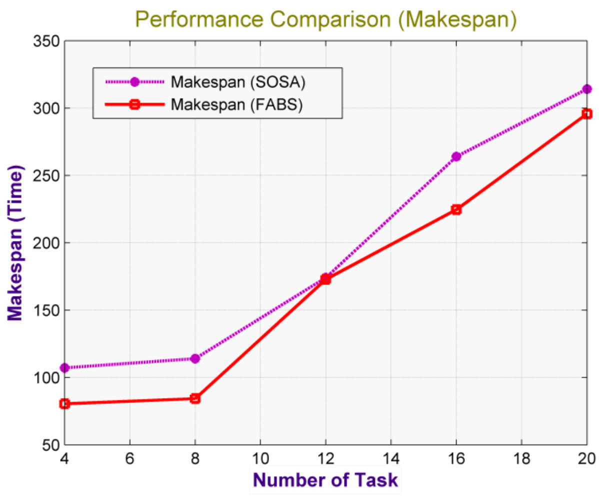

Figure 5: Makespan Comparison

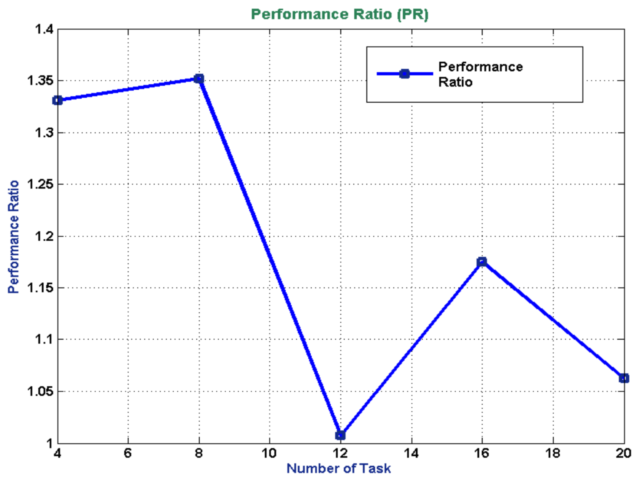

Figure 6: Performance Ratio

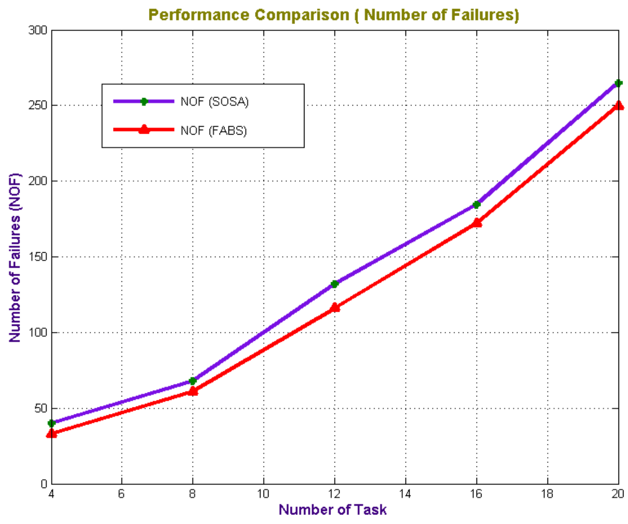

Figure 7: Number of Failures Comparison

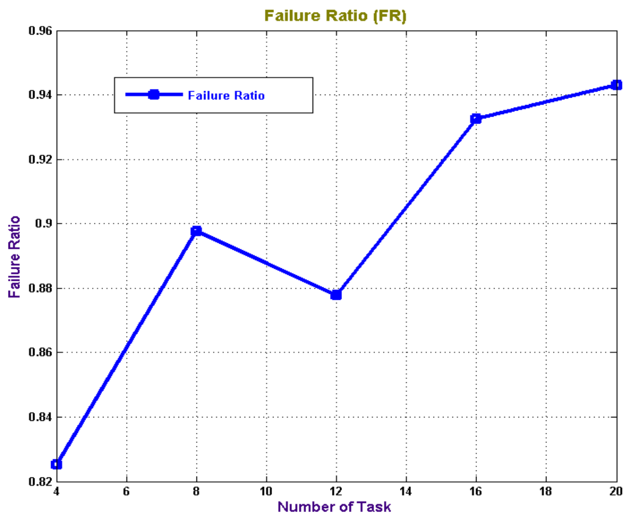

Figure 8: Failure Ratio

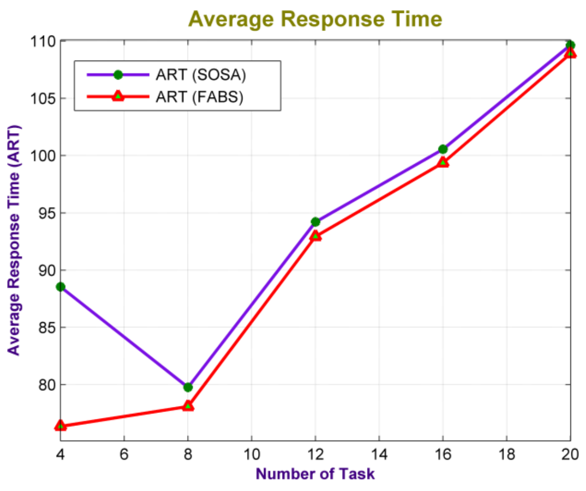

Figure 9: Average Response Time

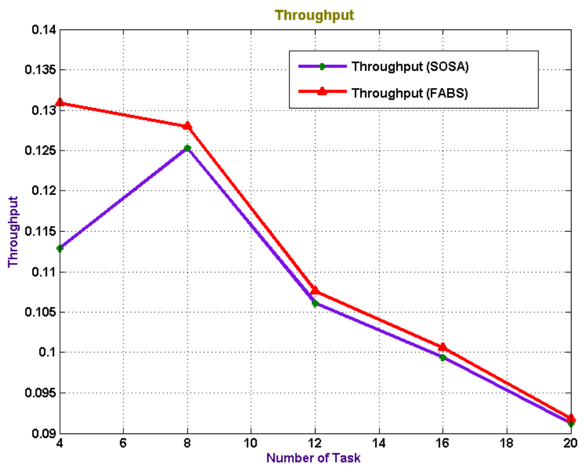

Figure 10: Throughput

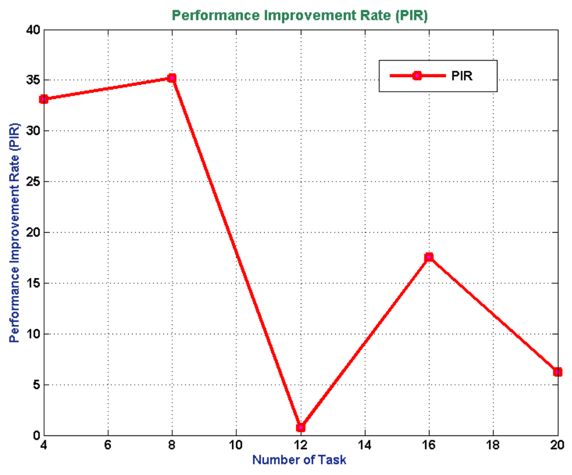

Figure 11: Performance Improvement Rate (PIR)

Figure 5 demonstrates that the FABS makespan is less than those of SOSA, which means that average time taken by the proposed algorithm is less than average time taken by the existing scheduling algorithm, and hence the efficiency of system is increased. *Figure 6* shows that the new algorithm's performance ratio (PR) is more than 1, means that FABS outperforms SOSA method. It signifies that the proposed algorithm takes less time to complete a task than the existing SOSA approach. *Figure 7* illustrates that the suggested approach has less failures than the existing algorithm and hence failure ratio (FR) of proposed algorithm is less than 1, as can be seen from *Figure 8*. Less number of failures means that less recovery and re-computing is required, hence saves the execution time. It means FABS algorithm increases the reliability of system by reducing the number of faults. *Figure 9* shows the ATR and it always remains less than that for Speed-only algorithm. It depicts that the process or task have to wait for less time to start its execution in case of proposed algorithm. *Figure 10* show the throughput and it always remains higher for FABS than that for SOSA i.e. FABS completes more number of tasks in the same time frame. *Figure 11* represents the percentage improvement in performance (PIR) in term of makespan. For instance when number of task is 8, FABS is performing best and giving 35% better performance than SOSA.

6. CONCLUSION

Grid computing has considered system failure a significant factor. In this paper, a solution for improving Grid computing performance is provided, when a node/resource fails. The approach is to employ a fault-tolerant environment and propose a job scheduling approach that is aware of failures. Weibull distribution is used to describe the duration between failures. Full checkpointing/restart mechanism is used to provide a fault tolerant environment. A task scheduling approach is designed and implemented which consider both performance matrices and failure matrices. Proposed algorithm recalculated the node capacity and schedule tasks with that new capacity of nodes. The performance analysis of purposed algorithm (FABS) is done making comparison with existing approach (SOSA) which considers only performance factors. Experimental data is analyzed based on five performance metric's which are PR, FR, ART, Throughput and PIR. The result graph clearly shows that the system performance is improved significantly where failure rate of nodes is considerable. This research present a failure-ware based scheduling method, in which failure information is considered to make more effective scheduling decisions and the objective to improve the system reliability while minimizing the job execution time is achieved.

REFERENCES

[1] M. Baker, R. Buyya, and D. Laforenza, "Grids and Grid technologies for wide-area distributed computing", Software – Practice and Experience. Vol. 32, No. 15, 2002.
[2] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauve, "Faults in grids: why are they so bad and what can be done about it?" In: Proc. of First Latin American Web Congres, pp. 18-24, 2003.
[3] J. H. Abawajy, "Fault Detection Service Architecture for Grid Computing Systems", In: Proc. of ICCSA 2004, LNCS 3044, Springer-Verlag Berlin Heidelberg, pp. 107–115, 2004.
[4] B. Nazir and T. Khan, "Fault Tolerant Job Scheduling in Computational Grid", In: Proc. of IEEE 2nd International Conference on Emerging Technologies, Peshawar, pp. 708-713, 2006.
[5] T. Do, T. Nguyen, D. T. Nguyen, and H. C. Nguyen, "Failure-aware scheduling in Grid computing environments", In: Proc. of the International Conference on Grid Computing and Application, 2009.
[6] M. Paun, N. Naksinehaboon, and R. Nassar, "Incremental Checkpoint Scheme for Weibull Distribution", International Journal of Foundations of Computer Science, Oct. 2009.

[7] R. Garg and A. K. Singh, “Fault Tolerance in Grid Computing: State of the Art and Open Issues”, International Journal of Computer Science & Engineering Survey (IJCSES), Vol. 2, No. 1, pp. 88-97, 2011.
[8] P. Latchoumy and P. S. A. Khader, “Survey on Fault Tolerance in Grid Computing”, International Journal of Computer Science & Engineering Survey (IJCSES), Vol. 2, No. 4, pp. 97-110, 2011.
[9] M. Tiryakioglu and D. Hudak, “Guidelines for 2-Parameter Weibull Analysis for Flaw-Containing Materials”, Metallurgical & Materials Transactions, Vol. 41, pp. 1130-1147, 2011.
[10] Nielsen and A. Mark, “Parameter Estimation for the Two-Parameter Weibull Distribution” https://scholarsarchive.byu.edu/etd/2509, Theses and Dissertations, pp. 1-99, 2011.
[11] Z. Pooranian, M. Shojafar, and B. Javadi, “Independent Task Scheduling in Grid Computing Based on Queen-Bee Algorithm”, IAES International Journal of Artificial Intelligence (IJ-AI), Vol. 1, No. 4, pp. 171-181, 2012.
[12] J. K. Naik and N. Satyanarayana, “A Novel Fault-tolerant Task Scheduling Algorithm for Computational Grids”, In: Proc. IEEE Conference, ISBN 978-1-4673-2818-0/13, 2013.
[13] H. B. Prajapati, and V. A. Shah, “Scheduling in Grid Computing Environment”. In: Proc. 2014 Fourth International Conference on Advanced Computing & Communication Technologies, ISBN: 978-1-4799-4910-6, DOI: 10.1109/ACCT.2014.32, 2014.
[14] H. Sajedi and M. Rabiee, “A Metaheuristic Algorithm for Job Scheduling in Grid Computing”, I.J. Modern Education and Computer Science, Vol. 5, pp. 52-59, 2014.
[15] R. Garg and A. K. Singh, “Fault Tolerant Task Scheduling on Computational Grid Using Checkpointing Under Transient Faults”, Springer, Arab J Sci Eng, Vol. 39, pp. 8775–8791, 2014.
[16] R. Garg and A. K. Singh. “Adaptive workflow scheduling in grid computing based on dynamic resource availability”, Engineering Science and Technology, an International Journal, Vol. 18, pp. 256-269, 2015.
[17] J. Shanthini, T. Kalaikumaran, and S. Karthik, “Hybrid Scheduling Model for Independent Grid Tasks”, Hindawi Publishing Corporation The Scientific World Journal, pp. 1-9, 2015.
[18] P. Jiang, Y. Xing, X. Jia, and B. Guo, “Weibull Failure Probability Estimation Based on Zero-Failure Data”, Hindawi Publishing Corporation, Mathematical Problems in Engineering Volume , pp. 1-8, 2015.
[19] P. Keerthika and P. Suresh, “A Multiconstrained Grid Scheduling Algorithm with Load Balancing and Fault Tolerance”, Hindawi Publishing Corporation, The Scientific World Journal, pp. 1-10, 2015.
[20] R. Kumar and S. Charu, “Comparison between Cloud Computing, Grid Computing, Cluster Computing and Virtualization”, IJMCSA, Vol. 3, No. 1, pp. 42-47, 2015.
[21] S. Haider and B. Nazir, “Fault tolerance in computational grids: perspectives, challenges, and issues”, Springer Plus, Vol. 5, pp. 1-20, 2016.
[22] M. K. Bhatia, “Task Scheduling in Grid Computing: A Review”, Advances in Computational Sciences and Technology, Vol. 10, No. 6, pp.1707-1714, 2017.
[23] M. T. Younis and Shengxiang, “A Genetic Algorithm for Independent Job Scheduling In Grid Computing” MENDEL- Soft Computing Journal, Vol. 23, No. 1, pp. 65-72, 2017.
[24] J. Liu, Z. Wu, J. Wu, J. Dong, Y. Zhao, and D. Wen, “A Weibull distribution accrual failure detector for cloud computing”, PLoS ONE, Vol. 12, No. 3, pp. 1-16, 2017.
[25] S. Haider and B. Nazir, “Dynamic and Adaptive Fault Tolerant Scheduling, With QoS Consideration in Computational Grid”, IEEE Access, Special Section On Emerging Trends, Issues, And Challenges In Energy-Efficient Cloud Computing, Vol. 5, pp. 7853-7873, 2017.
[26] H. Idris, A. E. Ezugwu, S. B. Junaidu, and A. O. Adewumi, “An improved ant colony optimization algorithm with fault tolerance for job scheduling in grid computing systems”, PLOS ONE , pp. 1-24, 2017. https://doi.org/10.1371/journal.pone.0177567.
[27] M. Soualhia, F. Khomh, and S. Tahar, “A Dynamic and Failure-aware Task Scheduling Framework for Hadoop”, IEEE Transactions on Cloud Computing, pp. 2168-7161, 2018.
[28] R. Buyya, and M. Baker, “Grids and Grid technologies for wide-area distributed computing”, SP&E., 2018.
[29] S. Sheikh, A. Nagaraju, and M. Shahid, “Dynamic load balancing with advanced reservation of resources for computational grid”, In: Proc. International Conference in Computing, Analytics and Networking, Springer, pp. 501–510, 2018.
[30] J. Natarajan, “Parallel queue scheduling in dynamic cloud environment using backfilling algorithm”, Int. J. Intell. Eng. Syst., Vol. 11, No. 2, pp. 39–48, 2018.
[31] M. T. Younis and S. Yang, “Hybrid meta-heuristic algorithms for independent job scheduling in grid computing”, Appl. Soft Comput., Vol. 72, pp. 498–517, 2018.
[32] V. L. Tran, E. Renault, V. H. Ha, and X. H. Do, “Time-stamp Incremental Checkpointing and its Application for an Optimization of Execution Model to Improve Performance of CAPE”, Informatica, Vol. 42, pp. 301–311, 2018.
[33] P. Kathalkar and A. V. Deorankar, “Study of Checkpoint Restore mechanism for Fault Tolerance in Cloud computing”. IJARSE, Vol. 7, No. 4, pp. 237-243, 2018.
[34] B. Anitha and G. K. Kamalam, "Heuristic Algorithm for Independent Task Scheduling In Grid Computing", IJRTE, Vol. 8, No. 4, pp. 12861-12866, 2019.
[35] Ankita and S. K. Sahana, “Evolutionary based hybrid GA for solving multi-objective grid scheduling problem”, Microsystem Technologies, Springer Nature, 2019.
[36] M. Singh, “An Overview of Grid Computing”, In: Proc. IEEE ICCCIS-2019, pp. 194-198, 2019.
[37] P. Sinha, G. Aeishel, and N. Jayapandian, "Computational Model for Hybrid Job Scheduling in Grid Computing", In: Proc. ICICV 2019, Lecture Notes on Data Engineering and Communications Technologies, ISBN: 978-3-030-28364-3, LNDECT 33, pp. 387–394, 2020.
[38] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing", Journal of King Saud University – Computer and Information Sciences, Vol. 33, pp. 1159–1176, 2021.
[39] H. Eluri and M. Gopichand, “Energy Management System and Enhancement of Power Quality with Grid Integrated Micro-Grid using Fuzzy Logic Controller” International Journal of Electrical and Electronics Research (IJEER), Volume 10, Issue 2, Pages 256-263, e-ISSN: 2347-470X, 2022.
[40] L. Jenila and R. Aroul Canessane, “Cross Layer Based Dynamic Traffic Scheduling Algorithm for Wireless Multimedia Sensor”, International Journal of Electrical and Electronics Research (IJEER), Volume 10, Issue 2, Pages 399-404, e-ISSN: 2347-470X, 2022.
[41] S. Kulkarni and A. Thosar, “Performance Analysis of Fault Tolerant Operation of PMSM using Direct Torque Control and Fuzzy Logic Control”, International Journal of Electrical and Electronics Research (IJEER), Volume 10, Issue 2, Pages 297-307, e-ISSN: 2347-470X, 2022.
[42] N. Thapliyal and P. Dimri, “Load Balancing in Cloud Computing Based on Honey Bee Foraging Behavior and Load Balance Min-Min Scheduling Algorithm”, International Journal of Electrical and Electronics Research (IJEER), Volume 10, Issue 1, Pages 1-6, e-ISSN: 2347-470X, 2022.