ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645
Indian Journal of Science and Technology, *Vol 8(S8), 507–515, April 2015* DOI: 10.17485/ijst/2015/v8iS8/70912

Dynamic Scheduling in Grid Environent with the Improvement of Fault Tolerant Level

K. Nirmala devi* and A. Tamilarasi

Department of Computer Application, Kongu Engineering College, Perundurai, Erode, Tamilnadu, India; nirmalak365@gmail.com, drtamil@kongu.ac.in

Abstract

Objective: The main intent of this research is to improve the fault tolerance capability in the geographically distributed resources by predicting the execution time of every job's earlier in the grid environment. **Methods:** In this manuscript Fault-Tolerant Based Dynamic scheduling algorithm (FTDS) is introduced for improving the fault tolerance mechanism. The execution time of every user submitted jobs are predicted to improve the fault tolerance capability. The check pointing system is maintained to keep track of all the jobs that are currently running in order to reschedule it in the new resources at the time of resource failure from the last execution step. The resource allocation at the time of failure is decided by the FTDS method and job processing state is decided by the check pointing system. **Results:** The Fault tolerance based dynamic scheduling algorithm shows better performance than the existing adaptive task check pointing and replication mechanism. In FTDS method, the processing time related parameters are computed to compare it with the existing approach. If the number of task is 500, the average response time in FTDS is 451 ms and the average co-ordination delay is 250ms and the makespan consumed is 450ms. Based on the comparison and the results from the experiment, it proves that the proposed approach works better than the other existing works with better performance. **Conclusion:** The findings demonstrate that the Fault-Tolerant Based Dynamic Scheduling algorithm is presented and suggested that this method has high fault tolerance rate to prevent the task execution from the resource failure.

Keywords: Check Pointing, Dynamic Scheduling, Fault Tolerance, Grid, Resource Failure

1. Introduction

Grid combines and coordinates users and resources that continue within various control domains. A grid is constructing from multifunction protocols and interfaces that address such problems like authorization, authentication and resource discovery. Grid computing is most important research area that combines geographically distributed computing resources into a single powerful system. Many applications can benefit from such integration. The problems like fault tolerance, security, heterogeneity and resource allocation are addressed by Grid software. Parallel computing on geographically distributed resources, generally known as distributed supercomputing, is one significant class for applications in grid computing. The grid computing provides the different kind of requirements such that sharing, selection, and aggregation of geographically distributed autonomous resources, at runtime, as a function of availability, capability, performance, cost, and user's quality-of-service and also its like as parallel computing.

In this work we integrate our execution time prediction module into a dynamic scheduling algorithm to observe to what extent the predictions made improve the algorithm performance. The algorithm operates in dynamic grid environments where tasks as input and unknown execution times (for which, however, periodic progress information can be collected) are run. The algorithm makes use of information services to collect dynamic system updates and applies these updates to (re)schedule tasks based on last saved checkpoint.

**Author for correspondence*

2. Previous Researches on Fault Tolerance Mechanisms

In this section, some of the previous research about the credit card fraud detection methods is suggested

In [9] introduced a Dynamic Fault Tolerant Dependency Scheduling (DFTDS) algorithm for to schedule the queries based on their dependency and it automatically allocates the resources by checking the status of the virtual machine based on its acknowledgment. This algorithm is used to solve the issues in the distributed data warehouse system. The proposed system integrates our previously proposed Dynamic Task Dependency Scheduling (DTDS) and Virtual Machine Fault Tolerant Scheduling (VMFTRS) algorithms[10–12] for scheduling queries based on their dependency as well as recycling resources without human intervention in case of any fault occurred in the virtual machine. This proposed DFTDS algorithm is an online, non preemptive type. Meaning that, the user given queries' estimated execution time does not know in advance and we do not suspend any queries when they are in the process of being executed. Initially user queries are stored in query table. Physical machines (PM) and virtual machines (VM) with their bandwidth details are saved in the resource table. In allocation table, details of allocated machines are stored and resource daemon reads this table to update the resource table. Whenever queries are submitted to PM or VM an entry is made in submit table and query daemon reads the entries from this table to update the query table. All these four tables are maintained in resource distributing monitor. In this model the online and preemptive scheduling algorithm for task prediction, resource classification is not addressed.

In [13] presented an Algorithm-based fault-tolerance (ABFT) solution to PDE's that can both reduce the curse of dimensionality in high dimensional problems and can be made robust in the presence of both hard and soft faults. It is based on the sparse grid combination technique[14], where the problem is solved on a number of anisotropic component grids that have high resolution in some—but not all—dimensions, which are then combined to form a sparse grid approximation to the full high-dimensional grid. The technique can also be formulated using a MapReduce paradigm, where, in the context of implementation on a multicore cluster, the Map tasks corresponds to the distribution of component grids for solution across subsets of cores in the system, and the Reduce tasks being their subsequent combination over the sparse grid. While it thus offers the same scope for fault-tolerance as in general MapReduce formulations, the sparse grid combination technique also offers fault-tolerance through the redundancy of the multiple component grids. Thus, if any part or all of the solution on a component grid is missing, instead of restarting the corresponding task, we can instead simply use information from the other grids to either reconstruct the missing component grid, or simply use a modified combination of the remaining component grids to produce the overall solution, at the possible cost of some loss in accuracy. This leads us to a modified Map Reduce programming pattern under which the Reduce task is dynamically formulated upon the degree of success (after a given time) of the spawned Map tasks.

In [15] proposed a fault tolerant decentralized scheduling algorithm for P2P grid, which reschedules jobs of grid resource when node failure happens. This mechanism handles node failure in fully decentralized manner. Key feature of this fault tolerant approach is that, it reschedules jobs depending upon communication and computation cost associated with jobs. In this paper, this is based on our earlier work[16]. It takes into consideration both communication and computation cost related to task. This algorithm schedules work inspired from gravitational field in physics. Object under the influence of gravity is always trying to shift to lower position that is why things fall from height. Similarly, job is scheduled from node having high magnitude workload to node having smaller magnitude. Now we know in grid system all resources are allowed to join or leave grid at any moment. We add a fault tolerant mechanism to [16]. This fault tolerant mechanism is inspired from fault tolerant techniques used in distributed systems. Fault Tolerance technique proposed in this paper is simple and highly effective. Our fault tolerant technique is appealing because it is fully decentralized and hence can be applied on many existing decentralized scheduling algorithms. Our fault tolerant mechanism becomes activated upon detection of grid node failure. When the grid resource finds out that, its neighbor is not responding it unfolds process to cope with failure. First, it discards all tasks assigned by failed neighbor node. Secondly, it rearranges its jobs, which were assigned to failed grid resource one by one. This way it handles node failure in P2P grid. This technique doesn't handle to detect malicious behaviour of nodes in fault tolerance mechanism. Moreover, it doesn't consider the task duplication techniques to get results fast if fault happens.

In [17] proposed a new bi-criteria scheduling algorithm (BSA) that considers user satisfaction along with fault tolerance. The main contribution of this paper includes achieving user satisfaction along with fault tolerance and minimizing the makespan of jobs. It also ensures that the tasks are completed within the user expected deadline. While scheduling the jobs, failure rate of the resources is also considered. The scheduler gets the list of tasks from the user along with the deadline (UD). This algorithm focuses on pro-active fault tolerant mechanism where failure consideration for the grid is made before the scheduling of a job and is dispatched with hopes that the job does not fail. While scheduling the jobs, failure rate of the resources and the load of each resource are also considered. The proposed algorithm improves system performance and user satisfaction and reduces number of failures of the jobs by considering fault rate of the resources. This algorithm doesn't consider the additional constraints like load factor of resources and resource availability.

In [18] proposed a precedence level based genetic algorithm (PLBGSA), which yields schedules for workflows in a decentralized fashion. Our approach says that we have to apply genetic algorithms to obtain the schedule for subtasks of one precedence level at one time. Also, if there is a single subtask at any precedence level, then we schedule subtask on P2P grid resource which gives results quickly. In this way, subtasks of DAG based task is scheduled over P2P grid resources from one precedence level to another. The probability of finding a nearly optimal schedule is higher with the approach adopted in this paper. This algorithm doesn't support the task duplication technique.

In [19] presented a new approach on fault tolerance mechanisms for the resource scheduling on grid by using Rough Set analysis in a local fashion and Case-Based Reasoning technique on scheduler machine. This approach applies a specific structure in order to prepare fault tolerance between executer nodes to maintain system in a safe state with minimum data transferring. Certainly, this algorithm increases fault tolerant confidence therefore, performance of grid will be high. This approach can select the best fault tolerance nodes and also detect a failure node and simply manage it by using one of the provided strategies such as multi versioning, Reservation Queue and Replacement, and transferring job to the nearest neighbor. The obtained results by our simulation indicate that the new approach may be very effective for adaptive grid scheduling due to reliability, fault tolerance, and then decrease of job completion time. This approach is not dynamic and intelligent component is missing in scheduling phase.

In [20] proposed a new evaluation (distributed) algorithm inspired by the effect of leaders in social groups, the group leaders' optimization algorithm (GLOA), to solve the problem of scheduling independent tasks in a grid computing system. GLOA is an evolutionary algorithm that is inspired by the effect of leaders in social groups. The problem space is divided into different groups, and each group has its own leader. The members of each group don't necessarily have similar characteristics, and they have quite random values. The best member of each group is selected as the leader. The members of each group try to become similar to their leader in each iteration. In this way, the algorithm is able to search a solution space between a leader and its group members. It is obvious that after some iteration, members of a group may become similar to their leader. In order to introduce diversity within a group, one of its members is selected randomly and some of its variables are interchanged with a member of another group. In addition, a crossover operator helps a group come out of local minima, and the solution space can be searched again so to produce diversity. This algorithm doesn't schedule the dependent task in grid environment.

In [21] proposed system focused on Fault Identification and Dynamic Scheduling. Since there is a probability of resource overloading due to dynamic scheduling of multiple jobs, and there may be chances of fault in the resources. The monitoring component gives the information about the resource overloading. In this, a novel algorithm is proposed to address this issue through implementing a strategy for dynamic rescheduling of jobs in the grid environment. We have proposed dynamic scheduling and rescheduling when fault is identified. The type of fault occurred here is overloaded fault. This fault occurs when the processor gets overloaded. To identify this type of fault the novel algorithm named Dual Space Search Algorithm (DSSA) is used proposed a dynamic, adaptive, and decentralized performance driven fault-tolerant load-balancing algorithm for computational grid environment.

In [22], introduced an new methodology for managing the resources which are present in the grid environment. Providing the grid resources to the user with the agreement of QoS requirement is the challenging process which is discussed in this work. In addition to that semantics

are considered while selecting the most suitable resources which lead to a reduced time complexity and the cost. The QoS parameters considered in this work are delay and bandwidth which need to provide sufficiently to the users to make the agreement with them. This work proves that the efficient resource allocation can achieved in the grid environment.

3. Fault-Tolerant based Dynamic Scheduling Algorithm

In the proposed work scheduling algorithm is achieved by dynamically adjusting the execution time mainly to maximize the throughput and minimize the execution time, and in case of resource failure, the proposed FTDS Algorithm reschedules the job from failed resource to another available resource based on fault occurrence record, and then the job is executed from the last stored checkpoint. The grid model considered in this paper consists of [8]: geographically distributed computational sites with several computational resources (r) at every site. The latter involve a user interface (UI) through which the jobs are submitted into the system; a Resource Broker (RB) which is used to identify all the available resources, a scheduler(S) to schedule the job to the available resources. A checkpoint server (CS) defined as where Check pointing data is made persistent. An Information server (IS) which collects the job and resource status information required by the scheduler and checkpoint server. It maintains the history of information about each and every resource. Assume the scheduler, Information server, checkpoint server is protected against failure and only the computational resources are unstable.

Our proposed algorithm can be subdivided into the following five steps (Figure 1):

Step 1 (dynamic collection of information): The information on resource load and availability, as well as the information on job status and progress of running jobs is collected. Important to mention is that the data collected can be outdated, depending on the length of the interval used by the IS to query the grid.

Step 2 (Execution time prediction): In this step, the execution time predictions of tasks within running set of tasks are (re)computed based on updates in task progress and resource status. By the term running task we understand a current task that exclusively contains finished

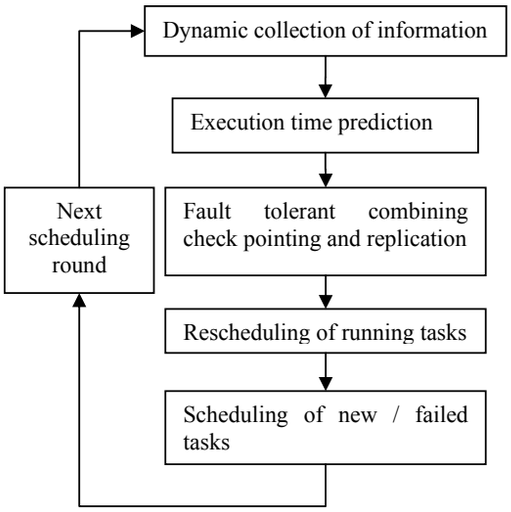

Figure 1. Process of Fault-Tolerant Based Dynamic Scheduling algorithm

tasks and tasks actually running on active grid resources, but not waiting tasks. The algorithm proposed is primarily designed for dynamic schedulers assigning jobs with *a priori* unknown execution times within dynamic distributed environments. The idea is that the scheduler periodically consults the prediction algorithm to determine a new job execution time estimate, based on the earlier collected job progress history. The functionality of the proposed algorithm can be formalized as follows:

A Novel algorithm for Fault tolerance with the consideration of check pointing system:

Input: Job id: J,
Set of (estimate, progress timestamp)-pairs: $H_{est} = \{(E_1,T_1), \ldots, (E_n,T_n)\}$,
Set of initial parameter values: Init = $\{i_1, \ldots, i_m\}$,
Set of prediction evolution functions: Func = $\{(F_1,\ldots,F_k)\}$

Output: Job execution time estimate: E_J^{est}

1: Pi= Size (H_{est})
2. **If** Pi ≡ GetPreviousEstimatesNo(J) **then**
2: $E_J^{est} \Leftarrow E_n$
3: **else**
4: SetPreviousEstimatesNo (J, Pi)
5: **if** Pi ≡ 1 **then**
6: $E_J^{est} \Leftarrow E_1$

7: **else**

8: **for all** $F_j \in Func$ **do**

9: $\left[X_{Fj}, Norm_{Fj}\right] \Leftarrow Match\left(F_j, Init, H_{est}\right)$

10: **end for**

11: $F_j \Leftarrow MinNormGet\left(Norm\right)$

12: $E_j^{est} \Leftarrow CalculateLimit\left(F_j, X_{Fj}\right)$

13: **end if**

14: **end if**

The input to the algorithm consists of the following parameters: the id of a job (*J*), for which a new estimate of the execution time is required; the prediction history H_{est}, containing pairs of execution time predictions *E*, computed by estimate of consecutive progress measurements, and progress collection timestamps *T* ; and a set, *Init*, of initial parameter values (*i*). The initial parameters serve to initialize the optimization performed in the scope of fitting. The parameters within *Init* are provided by end users, which are presumed to possess sufficient application knowledge to provide for the appropriate initial values. A good choice of the latter is highly important for the accuracy of the prediction mechanism, since it avoids the optimization ending in a local optimum. Also the set, *Func*, of functions, is provided to the algorithm. Each function describes a possible evolution over time of job execution time estimates. The historical input-data (*E*) is matched against the available functions to provide a new estimate, which is the output of the algorithm. Before proceeding with calculating, the algorithm first checks whether new information on job progress has become available since the last algorithm run. If the latter is not the case, the previous value of still applies (see lines 1–2). On the other hand, when the number of extrapolated estimated values increases ($Size(H_{est})$), this number is saved for the next run and the algorithm proceeds with computing the new (see lines 3–14).

Obviously, it is assumed that the prediction algorithm is called only after at least one progress indication is collected. However, since no fitting can be performed for a single point, the algorithm simply returns the initial estimate value E_1 (see lines 5–6). When the set H_{est} contains multiple estimates, for each predefined function the optimization procedure *Match* is called (see lines 7–9). The *Match* method takes as arguments a function, the initial parameter values and the execution time estimate data, together with the estimate timestamps. The outputs of the method are two vectors: *X* contains the parameter values that best fit function *(X, T)* to the data H_{est}; and *Norm*, which used by the prediction algorithm to determine the function that best fits the provided input data. *Norm* is calculated as the (see Formula 1), which means the parameter depicts the squared difference between the optimized function and the input data. Clearly, the smaller the difference as the best fit.

$$Norm_{Fj} = \sum_{k} \left(F_j\left(Init, T_k\right) - E_k\right)^2$$

Finally, the limit of the function with the minimum *Norm* is calculated, which provides us with a new execution time prediction (see lines 11–12). As was mentioned previously, we assume that the longer a job is running, the closer its execution time prediction gets to the real execution time value. This means that the function representing the execution time evolution converges over time to a certain limit-value. Since there are always only a limited number of functions provided, the limit expression can easily be determined analytically by an end user and provided to the algorithm together with the *prediction evolution* functions (*Func*). Afterwards, the predictions can be calculated by substituting the *X*-parameters into the limit expression of the -function.

Step 3 (**Fault-tolerant algorithm**). This approach mainly concentrates on achieving fault tolerance by reducing unnecessary checkpoint overhead which will reduce the job throughput. Hence to reduce the unnecessary checkpoints the two algorithms differentiates the check pointing interval based on the history of failure frequency of the resource and current status of a particular job. Here we consider two parameters: the last failure time of a resource, and the mean failure time of resource. These parameters suggest the stability of the resource based on which the check pointing interval is omitted or modified.

Dynamic Adaptation of Checkpoints

By dynamically changing the checkpoint frequency[8], we will, on one hand, eliminate unnecessary checkpoints and, on the other hand, introduce extra job state savings, where the danger of failure is considered to be severe.

Last Failure Time Based Checkpoint Adaptation

The main aim of this Last Failure time based Checkpoint Adaptation (LFCA) algorithm[8] is to omit unnecessary checkpoint in-order to reduce the checkpoint overhead on

a relatively stable resource. These unnecessary checkpoints are omitted mainly to reduce the overhead and to increase the job throughput. This algorithm considers the last failure time (Lf_r) of the resource, which is one of the parameter that suggests the stability of a resource.

Mean Failure Time Based Checkpoint Adaptation

The Mean Failure based Checkpoint Adaptation (MFCA) algorithm[8] dynamically modifies the check pointing frequency and deal with inappropriate check pointing intervals. The check pointing frequency is modified based on the Remaining job execution time (RE_r^j) and mean failure interval of the resource (Mf_r) where r is the resource and j is the job assigned to that resource. The use of mean failure time instead of last failure time reduces the effect of individual failure event.

Step 4 (Rescheduling of Running task). Running tasks are reassigned to balance their predicted execution times: the longest task is assigned to the fastest available resource (minimizing maximum execution time), while other tasks are assigned such that their execution times are as close as possible to the execution time of the longest task within the running task, without actually exceeding it (minimizing processing time difference). This means that the shortest tasks are migrated to slowest resources, leaving the fastest resources to the tasks requiring fast execution.

Step 5 (Scheduling of idle tasks). The running tasks containing idle tasks are assigned to the resources remaining after the rescheduling step. The tasks are processed in the order of their arrival into the Grid Scheduler-queue. For some applications we may possess an initial estimate of the total tasks execution time. In this case the scheduler proceeds as described in the previous step. Otherwise, tasks are assigned randomly.

4. Performance Evaluation

In this section, the performance is validated to compare the existing Adaptive task check pointing and replication algorithm and the proposed Fault tolerant based dynamic scheduling algorithm. The following are the graphical results of our implemented systems namely FTDS considered for the comparison of these methods are namely

- Response time
- Co-ordination delay and
- Makespan

Based on the comparison and the results from the experiment shows the proposed work works better than the other existing systems with higher fault tolerance rate

4.1 Response Time

Response time for a task is the delay between the submission time and the arrival time of execution output which is shown in Figure 2 Effectively, the response time includes the latencies for coordination and the CPU time.

Response time is calculated as follows:

Response Time = Entering time + Schedule start time

In Figure 2, Number of tasks ranging from 50 to 500 is taken along x-axis and average response time per task (in seconds) is taken along y-axis ranging from 0 to 600. It can be inferred from the graph that response time of Fault-Tolerant Based Dynamic Scheduling is lesser than Adaptive Task Check pointing and Replication.

Table 1 Shows the average response time values for the existing and the proposed system for the number of tasks. If the number of tasks is increased, the response time is reduced in the proposed system when compared to the existing system.

4.2 Average Co-Ordination Delay

The metric coordination delay sums up the latencies for: (i) resource claim to reach the index cell, (ii) waiting time till a resource ticket matches with the claim, and (iii) notification delay from coordination service which is shown in Figure 3.

The Figure 3 on which number of tasks ranging from 50 to 500 is taken along x-axis and average coordination

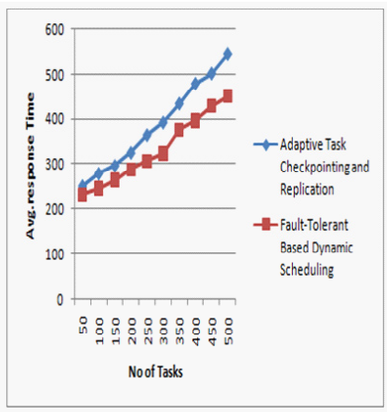

Figure 2. Average response time graph.

Table 1. Average Response time Vs Number of Tasks

SNO	Number of Task	ATCA	FTDS
1	50	250	232
2	100	278	246
3	150	296	264
4	200	325	289
5	250	364	307
6	300	39	32
7	350	43	37
8	400	47	39
9	450	50	42
10	500	54	45

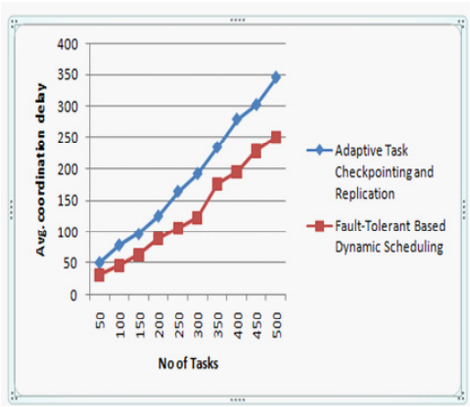

Figure 3. Average coordination delay graph.

delay time per task (in seconds) is taken along y-axis ranging from 0 to 350. It can be inferred from the graph that coordination delay time of Fault-Tolerant Based Dynamic Scheduling is lesser than Adaptive Task Check pointing and Replication. The values plotted in the graph are listed in the following Table 2.

4.3 MakeSpan

Makespan is measured as the response time of a whole workflow, which equals the difference between the submission time of the entry task in the workflow and the output arrival time of the exit task in that workflow which is shown in Figure 4.

Makespan is calculated by using the following equation:

Makespan = $\max\{CT_{ij} | i \in T, i = 1, 2, \ldots n \text{ and } j \in VM, j = 1, 2, \ldots m\}$

Note that, these measurements (except makespan) are collected by averaging the values obtained for each task

Table 2. Average Co-ordination delay Vs number of tasks

SNO	Number of Task	ATCA	FTDS
1	50	50	32
2	100	78	46
3	150	96	64
4	200	125	89
5	250	164	107
6	300	192	124
7	350	234	177
8	400	279	196
9	450	302	229
10	500	345	251

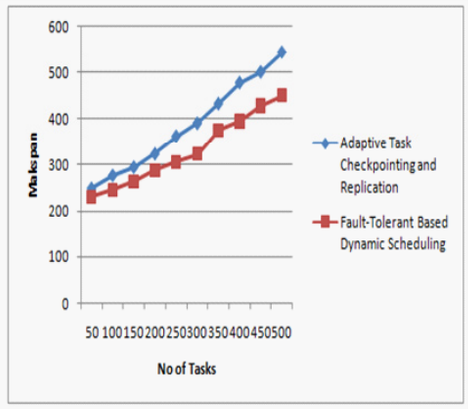

Figure 4. Makespan graph.

Table 3. Avg Makespan Vs number of tasks

SNO	Number of Task	ATCA	FTDS
1	50	250	232
2	100	278	246
3	150	296	264
4	200	325	289
5	250	364	307
6	300	392	324
7	350	434	377
8	400	479	396
9	450	502	429
10	500	545	451

in the system. The measurement of makespan is taken by averaging over all the workflows in the system. In Figure 4, Number of tasks ranging from 50 to 500 is taken along x-axis and average makespan workflow is taken along y-axis ranging from 0 to 500. It can be inferred from the graph that makespan of Fault-Tolerant Based Dynamic Scheduling is lesser than Adaptive Task Check pointing and Replication.

5. Conclusion and Future Work

Fault tolerance forms a major issue in all kind of distributed platforms. Here we concentrate on the issues of fault tolerance in terms of resource failure. Thus the proposed work achieves fault tolerance by dynamically adjusting the checkpoint frequency based on history of failure information and job execution time, which minimizes checkpoint overhead, and maximizes the throughput and prediction approach is proposed that uses the progress history to dynamically predict the execution time. In case of resource failure, the proposed FTDS algorithm effectively reschedules the job from failed resource to another available resource by minimum execution time based on the history of fault occurrence information which is available in the information server. Therefore the fault-tolerant based dynamic scheduling algorithm dynamically adjusts the checkpoint frequency and job execution time and thus makes to improve the system performance.

In future work we extend our work to considers fault-tolerance mechanisms based on Job retry, Job migration without check pointing (JMG) mechanism, Job migration with check pointing (JCP) mechanism and Job Replication (JRP) and apply optimization techniques to minimize the execution time.

6. References

1. Liang H, Xi-Long C, Si-Qing Z. Online System for Grid Resource Monitoring and Machine Learning-Based Prediction. IEEE transactions on parallel and distributed systems. 2012 Jan; 23(1).
2. Poonguzhali M. Dwelling-Time Based Resource Scheduling Algorithm Using Fuzzy Logic in Grid Computing. International Conference on Computer Communication and Informatics (ICCCI -2012); 2012 Jan 10–12.
3. Zikos S, Karatza HD. Resource allocation strategies in a 2-level hierarchical grid system. Proceedings of the 41 st Annual Simulation Symposium (ANSS); 2008 Apr 13-16. IEEE Computer Society Press; SCS. p. 157–64.
4. Reddy KHK, Roy DS. A Hierarchical Load Balancing Algorithm for Efficient Job Scheduling in a Computational Grid Testbed. Recent Advances in Information Technology. 2012.
5. Lorpunmanee S, Sap MN, Abdullah AH, Chompoo-inwai C. An Ant Colony Optimization for Dynamic Job Scheduling in Grid Environment. International Journal of Computer and Information Science and Engineering. 2007; 207–14.
6. Kumar M, Mohanty MP, Mund GB. A Time-minimization Dynamic Job Grouping-based Scheduling in Grid Computing. 2012.
7. Song S, Kwok Y-K, Hwang K. Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling. IEEE Transactions on Computers. 2006; 55(6):703–19.
8. Chtepen M, Claeys FHA, Dhoedt B, De Turck F, Demeester P, Vanrolleghem PA. Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids Parallel and Distributed Systems. IEEE Transactions. 2009 Feb; 20(2): 180–90.
9. Krishnaveni S, Hemalatha M. Evaluation of DFTDS algorithm for distributed data warehouse. Egyptian Informatics Journal. 2014; 15:51–8.
10. Krishnaveni S, Hemalatha M. Query processing in distributed data warehouse using proposed dynamic task dependency scheduling algorithm. Int J Comput Appl. 2012; 55(8):17–22.
11. Krishnaveni S, Hemalatha M. Query scheduling in distributed data warehouse using DTDS and VMFTRS algorithms. Eur J Sci Res. 2012; 89(4):612–25.
12. Krishnaveni S, Hemalatha M. Query management in data warehouse using virtual machine fault tolerant resource scheduling algorithm. Int J Theor Appl Inform Technol. 2013; 47(3):1331–7.
13. Larson JW, Hegland M, Harding B, Roberts S, Stals L, Rendell AP, Strazdins P, Ali MM, Kowitz C, Nobes R, Southern J, Wilson N, Li M, Oishi Y. Fault-Tolerant Grid-Based Solvers: Combining Concepts from Sparse Grids and MapReduce. ICCS Procedia Computer Science. 2013; 130–9.
14. Bungartz H-J, Griebel M. Sparse grids. Acta Numerica. 2004; 13:147–269.
15. Chauhan P, Nitin. Fault Tolerant Decentralized Scheduling Algorithm for P2P Grid. 2nd International Conference on Communication, Computing & Security [ICCCS]; 2012. p. 698–707.
16. Chauhan P, Nitin. Decentralized Computation and Communication Intensive Task Scheduling Algorithm for P2P Grid. 14th International Conference on Computer Modelling and Simulation (UKSim); 2012. P. 516.

17. Keerthika P, Kasthuri N. An Efficient Grid Scheduling Algorithm with Fault Tolerance and User Satisfaction. Mathematical Problems in Engineering: Hindawi Publishing Corporation; 2013.
18. Chauhan P, Nitin. Fault Tolerant PLBGSA: Precedence Level Based Genetic Scheduling Algorithm for P2P Grid. Journal of Engineering. 2013.
19. Bouyer A, Mohd Noor MD SAP. A Prediction-based Fault Tolerance on Grid Resources scheduling by using Optimized Case-based Reasoning. 2013. comp. utm.my.
20. Pooranian Z, Shojafar M, Abawajy JH, Singhal M. GLOA: A New Job Scheduling Algorithm for Grid Computing. International Journal of Artificial Intelligence and Interactive Multimedia. 2013; 2(1).
21. Kiruthika P, Dev SG. Dynamic Scheduling and Rescheduling With Fault Tolerance Strategy in Grid Computing. IJAICT. 2014; 1(2).
22. Barzegar B, Esmaeelzadeh H, Shirgahi H. A new method on resource management in grid computing systems based on QoS and semantics. Indian Journal of Science and Technology. 2011 Nov; 4(11):1416–9.