# Checkpointing Based Fault Tolerant Job Scheduling System for Computational Grid

**Mangesh Balpande** [1] **and Urmila Shrawankar** [1]

[1] Computer Science and Engineering, G.H. Raisoni College of Engineering, Nagpur, Maharashtra, India

*Corresponding Author Email: mangesh.balpande111@gmail.com*

## Abstract

*A computational grid environment, due to its heterogeneous, autonomous and dynamic nature is prone to different kinds of faults which may lead to delay in completion of job or even execution of job from starting point. Checkpointing mechanism plays a vital role for making grid more reliable, cost effective and efficient. In this paper, we have proposed schemes based on system checkpointing and application checkpointing. Their performance comparison is done based on the empirical study. The ABSC scheme is suitable for the applications where computations are not intense. But for computationally intense applications where reliability is more important ABAC scheme is more suitable. But this scheme may produce slight overheads in fault free situations and very reliable in faulty situations.*

***Keywords:*** *Fault Tolerance, Computational Grid, Checkpointing, Genetic algorithm, performance evaluation.*

## 1. Introduction

A computational grid environment consists of decentralized, heterogeneous and autonomously managed subsystems, where computations between these subsystems are independent with no sharing of intermediate results. Designing of a fault tolerance system in a grid environment with optimized resource utilization and execution time is a critical and challenging task. A good fault tolerant job scheduling approach should be able to handle not only the complexity of the resources also various faults occurring during the job execution.

For long running computationally intensive scientific applications specified in [1] [2] [3] like Simulations, decoding, scientific experiments and industrial areas like bioinformatics may require hours, days, weeks or even months to carry out the execution due to which they are prone to various types of faults. Thus the checkpointing techniques can be used to resolve this by recovering from any type of fault without disturbing the normal operation.

Grid checkpointing service should meet the following requirements [4]: 1) Interoperability among of computing resources: There should be a provision of interoperability among variety of computing resources and the checkpointing operation must be compatible with different platforms. Checkpointing service should provide standard API to allow users to code in different grid resources independently. 2) Availability of checkpointing data: The checkpointing data should be stored in separate storage devices. It should be capable of applying its flexibility to different computing resources. Depending on the condition that interface is not modified, it can adapt to the change in the foundational architecture. 3) Interoperability between grid middleware and infrastructure: Besides of interactive interfaces checkpoint service should provide security, monitoring, performance evaluation etc. while operating on checkpoint.

Checkpointing is a technique which allows a process to preserve its state during arbitrary time interval and resuming its normal operation to reduce faults during recovery process after failure [5]. The proposed work addresses empirical analysis and performance comparison in faulty and faulty free situations of system checkpointing and application checkpointing also their reliability and computational speed have been observed.

In this paper we have proposed ABSC scheme and ABAC scheme capable recovering from node failure using checkpointing mechanism. Remaining part of the paper is organized as follows: Various issues related to checkpointing are discussed in section II, related work is elaborated in section III, section IV discusses the proposed schemes, section V gives the result and analysis and section VI contains the conclusion and future work.

## 2. Checkpointing

In Grid computing to improve the reliability and system availability a commonly used fault tolerance technique is checkpointing. The efficiency of checkpointing mechanism is based on three parameters [3]: 1) Checkpointing overhead in terms of time and resources consumed to generate and transfer checkpoint among other nodes. 2) Time to resume the execution in case of failures: Checkpoint size plays a major role in this context. The size and frequency of checkpoint may vary depending on the QoS requirement [6] [7]. 3) Compatibility and Portability of checkpoints: Due to heterogeneous nature of grid, machines at which the checkpoints are generated may have different platform or operating system from which it is going to be used for resuming the execution.

Checkpointing algorithm can be categorized into system and application level [8]: 1) In application checkpointing, the application itself is coded in a way to generate and store its own checkpoints. Due to which checkpointing can be platform independent. Application can also be coded to reduce the checkpoint size and overhead. 2) System checkpointing is used to store the kernel level information. It requires much effort from the programmer. It can generate checkpoint state that can be obtained from RAM using multithreading, but it is less flexible and creates more checkpoint overhead.

The fault tolerance could be carried by approaches based on the job replication, checkpointing and adaptive approach [18] [9]. In checkpointing approach, the status of the running job before occurrence of the fault is stored into the stable storage and when fault occurs the roll backing of the state of the job up to the failure point is done. The rollback mechanism plays a vital role because data from the previous node may be faulty or the data recovery from the faulty state of a particular node may lead to malfunctioning [10].

The checkpointing mechanism along with rollback and recovery is as shown in figure1. 1) The grid user’s job is submitted to Resource Information Service (RIS) through scheduler manager for optimized resource utilization. The Job scheduler offers the computing resources from resource pool. 2) After receiving jobs from the user scheduler manager select optimal resources by submitting jobs to RIS. The mapping between jobs to resource is generated and a job dispatcher dispatches the job to central checkpoint unit. 3) As soon as a new checkpoint is created, rollback mechanism is initiated by checkpoint controller. It performs rollback operation for a faulty process and finds the possible states of recovery. The checkpoint replication services are initiated by checkpoint controller if the checkpoint rollback is not possible. Application replicas get called by replication services and if not available on demand application is carried out and critical services are replicated. The checkpoint controller contains the buffer inside to store the details such as time of initiation, identity of replicas.

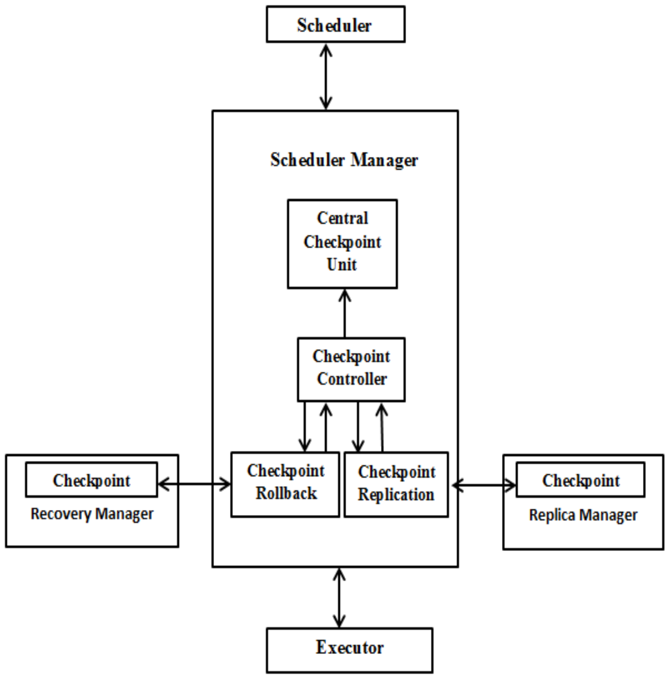

Fig 1: Checkpointing with rollback and recovery

## 3. Related Work

An Adaptive fault tolerant scheduling [2] utilizes an adaptive number of job replicas according to the grid failure history. This technique composed of Adaptive Job Replication (AJR) and Backup Resource Selection (BRS). AJR determines number of replicas according to selected resources. BRS evaluates the resources selected for execution by AJR. Due to adaptive nature, this technique reduces the excessive cost and response time, but may lead to overloading of Grid in extreme conditions.

The NSGA-II with fuzzy mutation works on the theory of diversity preservation [11]. In fuzzy mutation fuzzy if then rules are evaluated to get the probability of population in chromosome by using individual cost variance and gene variance as inputs. This technique requires less number of iterations but defuzzification cost is more.

The job scheduling based on application checkpointing is proposed in [3]. This scheme generates threads of application as checkpoints. For a specific time period a message based communication between the manager and executor is carried out to check either fault is carried out or not. This scheme provides more flexibility but may leads to checkpointing overhead in fault free situations and whenever checkpoint frequency increases

The method proposed in [12] consists of predictive scheduling which utilizes rough set theory for predicting the number of supporter nodes with less probability of failure. This scheme requires excessive storage for storing every state in advance.

The scheme in [13] consists of cooperative checkpointing which utilizes the dynamic adjustment of checkpointing interval with in the presence of failure to complete the job with in its time. In this the remaining time, time left after deadline and remaining number of failure expected are evaluated.

The work in [17] addresses a Distributed Fault Tolerant Scheduling (SFTS) algorithm that combines job scheduling and job replication together. One scheduler manager acts as a scheduler manager for another. It uses fixed number replicas for each job and that are scheduled to various sites to be executed. This method doesn't provide flexibility in scheduling and can perform bottleneck.

## 4. Proposed Algorithms Based On Checkpointing Scheme

The proposed algorithms are specifically based on the checkpointing mechanism. The ABSC is designed for fault tolerant job scheduling which is based on the Genetic Algorithm (GA) which utilizes a system checkpointing. The ABAC scheme addresses the mechanism in which the application threads are generated in the form of checkpoints. In case of failure the corresponding application thread from its checkpoint is resuming from the point of failure.

### *4.1 Algorithm Based on System Checkpointing (ABSC) Scheme*

The proposed ACSC Scheme is based on Genetic Algorithm (GA) which works on the principal of "Natural Selection" [11] and is used to provide the optimized solution in faulty situations. The working of the proposed scheme is based on four operations [15]: 1) Initialization of population generation: This operation uses genetic operators for generation and representation of possible solution of mapping between task and nodes in the search space. 2) Chromosome representation and evaluation: In grid computing a chromosome can be a one dimensional array of the node id indexed by the corresponding job allocated to it and the evaluation can be carried out with the help of fitness function. The fitness function calculates the fitness value which is nothing but the makespan between task and node.3) Crossover and Mutation operation: In crossover, the random selection of task from the number of tasks is taken known as crossover point. The parts of two parent chromosomes are swapped over the crossover point for the generation of two new chromosomes. The random selection of a job and replacement of its value in its entry in the parent chromosome with randomly selected valid resource id which leads to mutation.

**ALGORITHM:**

Start

1. Assume a Fault is occurred due to resource failure, resource overloading or underloading.
2. Generate the initial population as a set of nodes.
3. Fitness value F(x) is evaluated as [14]:

$$F(x) = \frac{F_i}{\sum_j^n F_j} \qquad \text{.......(1)}$$

Where, Fi- Fitness value of ith node

n- Total number of nodes.

4. Cross over of parameters like resources, failure rate, memory, speed etc. which leads to new chromosome.
5. If the fitness value of newly arrived chromosome from crossover is better than the available one then add this to the existing population else got to step 5.
6. While new generated population becomes zero and of the generated fitness value is satisfied then stop and return the best node else go to step 3.

7. Repeat this process until the Grid scheduler receives the backup storage.

8. Stop.

## *4.2 Algorithm Based on Application Checkpointing (ABAC) Scheme*

In ABAC scheme, the application itself is designed to store its own checkpoints by inserting the checkpoint code. The replica manager replicates the checkpointing data on various nodes. The recovery manager recovers in case of faulty situation.

The database used for storing the checkpoint data is Mysql. The table named as Applcheck is created which provides fast access and flexibility of storing various types of data also the intermediate results can be stored and retrieved from the table without disturbing the normal flow of application.

**ALGORITHM:**

```
1. Start
2. User submit the job to the scheduler manager
3. While (all thread finished execution)
   If (the thread is present in the application threads) then
       a. Execute the thread by executor node.
       b. Continue the handshaking between executor and
          manager
4. If handshaking delay is greater than threshold then
           a. Restart the failed thread on another available
              node.
5. Return the computed final result to the user.
6. Stop
```

The figure 2 shows different phases of thread during its execution. After the application has been started, threads of an application do some of its work and store the corresponding thread's checkpoint in a stable storage. After the completion the thread restarts its work.

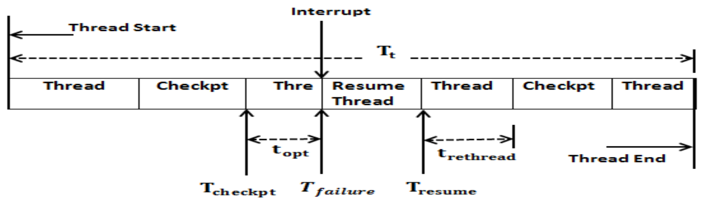

Fig.2. Thread execution with checkpoints

The application initiates checkpointing at time t then the time require to complete the execution of the thread is $T_t$. At fault free situation, the time requires to compute from last checkpoint up to the point of failure is $T_{failure}$. The time at which the latest checkpoint generated is $T_{checkpt}$. The time requires to restart the execution of the thread by scheduler manager is $T_{resume}$ and $T_{overhead}$ is the time consumed to generate and store checkpoints on scheduler manager in fault free situations. $t_{opt}$ indicates the checkpoint latency. If the job fails, its corresponding threads force the job to restart from previous checkpoint, it requires $T_{overhead}$ time.

The efficiency of checkpointing can be evaluated depending on the value of $T_t$. The minimum value of $T_t$ indicates efficient checkpointing scheme. Thus the total completion time of thread in a faulty situation is given by:

$$T_{tfailure} = (T_t - T_{checkpt}) + T_{resume} + T_{overhead} \quad \text{........ (2)}$$

## 5. Analysis and Discussion

Building a grid environment on a real scale to evaluate the fault tolerant scheduling techniques is very difficult. Gridsim a widely used simulator based on java. It supports the heterogeneous nature of resources and platforms. Since this work is concentrated on the resource failure and the latest version of the Gridsim toolkit 5.2 contains the package gridsim.ResFailure for detection of resource failure, thus it is used in our simulation.

The simulation parameters are as shown in table I. Based on different MIPS, nodes are registered to GIS (Grid Information Server). The handshaking delay threshold for resource failure in proposed application checkpointing is 150 seconds.

Table I. Parameter setting

| **Parameters** | **Values** |
|---|---|
| No. of Computing Nodes | 300 |
| No. of Tasks | 1200 |
| Speed of Resource Nodes (MIPS) | 1000-3500 |
| Task Length in Million Instructions (MI) | 300000-450000 |

The performance evaluations of the proposed checkpointing schemes have been done in two situations fault free and faulty and are also compared with the system checkpointing algorithm. Because application checkpointing generates the overhead in fault free situations, the system checkpointing algorithm performs better than application checkpointing algorithm. The results are as shown in table II with generated overhead and time difference.

Table II. Execution time along with the overhead in fault free situation

| Sr. No. | No. of Threads | ABSC Scheme | ABAC Scheme | Overhead Caused |
|---|---|---|---|---|
| 1 | 1 | 170.290 | 172.752 | 2.462 |
| 2 | 3 | 222.231 | 226.894 | 4.663 |
| 3 | 6 | 270.338 | 276.860 | 5.522 |
| 4 | 8 | 318.780 | 325.191 | 6.414 |
| 5 | 9 | 332.405 | 325.513 | 6.892 |

The experiment is carried out for evaluating the performance of the ABAC scheme in faulty situation. In this the execution is interrupted by either by switching off or disconnecting the executor node during the execution of the thread. The results are as shown in table III.

Table III. Execution time along with the saved time in faulty situation

| Sr. No. | No. of Threads | ABSC Scheme | ABAC Scheme | Saved Time |
|---|---|---|---|---|
| 1 | 1 | 187.465 | 172.344 | 15.121 |
| 2 | 3 | 248.728 | 232.458 | 16.270 |
| 3 | 6 | 360.606 | 327.164 | 33.442 |
| 4 | 8 | 409.826 | 367.453 | 42.373 |
| 5 | 9 | 418.917 | 373.707 | 45.214 |

By studying the experimental results, we can say that, in fault free situations the ABSC scheme performs well. But the dynamic nature of grid makes it more vulnerable to various faults. For such situation this scheme is not well suited, instead ABAC scheme provides very good results.

In both faulty and fault free situation the checkpoint frequency may have a considerable effect on total execution time. It may lead to increase in total execution time in fault free situation. It depends on the failure rate and thread failure time [3]. On the other hand the increased frequency of checkpoint may improve recovering capability of job after the node failure [16]. Based on the results shown in table III, the execution time comparison can be observed graphically as shown in figure 3.

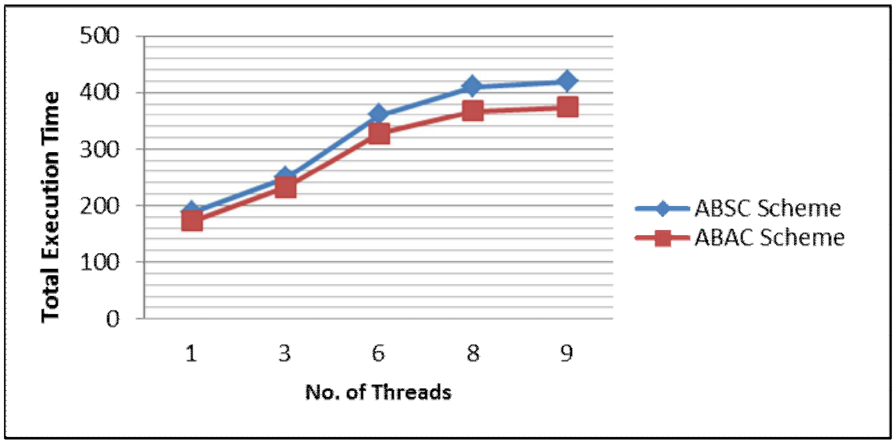

Fig.3. comparison of Total execution time

## 5. Conclusion and Future

The dynamic and heterogeneous nature of the grid makes it more vulnerable to different types of faults. In this paper, we proposed two fault tolerant job scheduling algorithms based on system checkpointing and application checkpointing. The performance of both the algorithms for total execution time and generated overhead under faulty and fault free situation is evaluated. Thus it can be concluded that the ABAP scheme provides better source sharing, improved resource utilization and computational speed which is best suited for the computationally intensive applications in grid environment.

Thus After the performance comparison we can say that, in future the proposed ABAC scheme could be modified to deal with scheduler manager failure and for improving the reliability of tasks, resource utilization and computational speed by increasing the frequency of checkpoints with in an application itself.

## References

[1] Chtepen, M.; Claeys, F.H.A.; Dhoedt, B.; De Turck, F.; Demeester, P.; Vanrolleghem, P.A., "Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids," *Parallel and Distributed Systems, IEEE Transactions on* , vol.20, no.2, pp.180,190, Feb. 2009.

[2] Amoon, M., "Design of a Fault-Tolerant Scheduling System for Grid Computing," *Networking and Distributed Computing (ICNDC), 2011 Second International Conference on* , vol., no., pp.104,108, 21-24 Sept. 2011.

[3] Bawa, R.K.; Singh, R., "Application checkpointing in grid environment with improved checkpoint reliability through replication," *Computing Communication & Networking Technologies (ICCCNT), 2012 Third International Conference on* , vol., no., pp.1,6, 26-28 July 2012.

[4] Wenxing Wang; Zhen Li, "Research of Process Migration Mechanism Based on Checkpoint in Computational Grid," *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual* , vol., no., pp.245,248, 16-18 July 2010.

[5] Rakesh, V.K.; Kar, C.; Samanta, T.; Banerjee, S., "A resource selection strategy and check pointing to minimize computational time in case of grid resource failure," *Advanced Communication Control and Computing Technologies (ICACCCT), 2012 IEEE International Conference on* , vol., no., pp.444,448, 23-25 Aug. 2012.

[6] El-Desoky, A.E.; Ali, H.A.; Azab, A.A., "Improving Fault Tolerance in Desktop Grids Based On Incremental Checkpointing," *Computer Engineering and Systems, The 2006 International Conference on* , vol., no., pp.386,392, 5-7 Nov. 2006.

[7] Janki Mehta; Chaudhary, S., "Checkpointing and Recovery Mechanism in Grid," *Advanced Computing and Communications, 2008. ADCOM 2008. 16th International Conference on* , vol., no., pp.131,140, 14-17 Dec. 2008.

[8] S.T.L. Anthony, G.Sumathi, S.Anthony Dalya, "*Dynamic Adaptation of Checkpoints and Rescheduling in Grid Computing*", *International. Journal of Computer Applications*, Vol. 2(3). pp. 95-99, 2010.

[9] Medeiros, R.; Cirne, W.; Brasileiro, F.; Sauve, J., "Faults in grids: why are they so bad and what can be done about it?," Grid Computing, 2003, Proceedings. Fourth International Workshop on , vol., no., pp.18,24, 17 Nov. 2003.

[10] Baghavathi Priya, S.; Subramaniam, C.; Ravichandran, T., "On demand check pointing for grid application reliability using communicating process model," *Advanced Communication Technology (ICACT), 2011 13th International Conference on* , vol., no., pp.393,398, 13-16 Feb. 2011.

[11] Salimi, R.; Motameni, H.; Omranpour, H., "Task scheduling with Load balancing for computational grid using NSGA II with fuzzy mutation," *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on* , vol., no., pp.79,84, 6-8 Dec. 2012.

[12] Bouyer, A.; Abdullah, A.H.; Ebrahimpour, H.; Nasrollahi, F., "Fault-Tolerance Scheduling by Using Rough Set Based Multicheckpointing on Economic Grids," *Computational Science and Engineering, 2009. CSE '09. International Conference on* , vol.1, no., pp.103,109, 29-31 Aug. 2009.

[13] Yang Xiang; Zhongwen Li; Hong Chen, "Optimizing Adaptive Checkpointing Schemes for Grid Workflow Systems," *Grid and Cooperative Computing Workshops, 2006. GCCW '06. Fifth International Conference on* , vol., no., pp.181,188, Oct. 2006.

[14] Priya, S.B.; Prakash, M.; Dhawan, K. K., "Fault Tolerance-Genetic Algorithm for Grid Task Scheduling using Check Point," *Grid and Cooperative Computing, 2007. GCC 2007. Sixth International Conference on* , vol., no., pp.676,680, 16-18 Aug. 2007.

[15] Upadhyay, N.; Misra, M., "Incorporating fault tolerance in GA-based scheduling in grid environment," *Information and Communication Technologies (WICT), 2011 World Congress on* , vol., no., pp.772,777, 11-14 Dec. 2011.

[16] Latip, R.; Lew Wai San; Chanchary, F.H., "Checkpointing in selected most fitted resource task scheduling in grid computing," *Computer Science & Education (ICCSE), 2012 7th International Conference on* , vol., no., pp.331,334, 14-17 July 2012.

[17] J. Díaz, S. Reyes, A. Niño, C. Muñoz-Caro, "Derivation of Self-Scheduling Algorithms for Heterogeneous Distributed Computer Systems: Application to Internet-based Grids of Computers," *Future Generation Computer Systems*, ElSevier, vol. 25, no. 6, pp. 617-626, 2009.

[18] R.k.bawa and Ramandeep Singh. Article: Comparative Analysis of Fault Tolerance Techniques in Grid Environment. International Journal of Computer Applications 41(1):21-25, March 2012