International Journal of Grid and Distributed Computing
Vol. 9, No. 3 (2016), pp.135-144
http://dx.doi.org/10.14257/ijgdc.2016.9.3.16

# Proactive Fault Tolerance Algorithm for Job Scheduling in Computational Grid

Sarpreet Singh[1] and R.K. Bawa[2],

[1]*Assistant Professor, Department of Computer Science & Engineering, Sri Guru Granth Sahib World University, Fatehgarh Sahib*
[2]*Professor, Department of Computer Science, Punjabi University Patiala*
[1]*ersarpreetvirk@gmail.com*
[2]*rajesh_k_bawa@yahoo.com*

***Abstract***

*Grid use huge number of dynamic, distributed & heterogeneous resources under different organizations domain to executing user applications. Working with these resources, applications face problems due to occurrence of faults. There are number of different faults that occur due to many problems. Therefore grid computing needs an efficient fault tolerance mechanism. This fault tolerance makes the grid system capable to work correctly even in the presence of faults. In this paper we have study various kinds of faults and reasons for occurrence. With this, also study various kinds of existing fault tolerance techniques. We devise a strategy, based on proactive fault tolerance and take appropriate step, if any possibility for fault occurrence. Scheduling of task on available resources, which is provided by GIS consider previous history of these resources. This will decrease the probability of faults, execution time, and increase the execution rate. Proposed strategy also uses technique of check pointing to reduce execution cost. To implement proposed strategy GridSim toolkit is used for simulation*

***Keywords****: Grid computing, faults, fault tolerance techniques, Rescheduling, Proactive, check pointing*

## 1. Introduction

Grid is a type of parallel & distributed computing system & different from other computing system like cloud and cluster. In the case of grid computing, instead of using dedicated resources from only single organization for computational purpose, grid provides a platform that enables resources from different organizations to be shared so as to able to solve scientific problem which needs huge amount of resources.

A resource in a grid goes down or may be disconnected from the rest network due to a many reasons. It is essential for grid to handle such scenario for transparent manner to the user. The reason for this is that a user visualizes the grid as one virtual machine. User submits the application for execution and gets the result from it. They are not concerned with failures & techniques to recover from failure. So, failures should be handled by the grid without any user intervention. Grid resources/node loss their QoS due to occurrence of any one threat like fault, error, failure, crash. To comprehend fault tolerance mechanisms, it is important to point out the difference between faults, errors and failures

- Fault is a condition that causes the software to fail to perform its required function.
  - Error refers to difference between actual output & expected output.
  - Failure is inability of a system or component to perform required function according to its specification.
  - *Crash:* When nothing happens.

ISSN: 2005-4262 IJGDC

Reference [1] [2] [3] presents various types of faults that may occur in grids.

I. *Hardware Faults*: Hardware failures are due to occurrence of fault in components such as CPU, memory defect in RAM, ROM or cache and storage devices (due to bad sectors). Hardware faults can be classified into permanent, transient, intermittent, benign or malicious.
   - A *permanent fault* reflects the permanent going out of commission of a component.
   - *Transient Failure:* Staying for a short time period & causes malfunction to a component for some time. When fault is recovered functionality of the component is fully restored.
   - *An intermittent fault* oscillates between quiescent and active state. When the fault is quiescent, the component functions normally; when the fault is active, the component malfunctions.

II. *Application and Operating System Faults*: These faults include memory leaks when an application does not release allocated. Operating system faults such as deadlock or improper resource management, unavailability of requested resource, etc.

III. *Network Faults*: It happens as a result of physical damage to the networks. This may result in significant packet loss or packet corruption. Network faults also occur when individual nodes go down due to reasons like hardware faults.

IV. *Software Faults:* Such faults may occur due to poor implementation of logics such as division by zero or unexpected input like incorrect file supplied as the input to the program resulting in erroneous results.

V. *Unconditional termination:* Due to some hardware or software interrupt.

VI. *Lifecycle faults:* Legacy or versioning faults.

VII. *Service expiry fault:* The service time of a resource may expire while application still using the resources.

VIII. *Interaction faults:* Such faults occur due to timing overhead, protocol incompatibilities, security incompatibilities, policy problems.

IX. *Omission:* Error occurs when one or more responses fail.

This paper is organized as follow. Section 2 discusses brief fault tolerance techniques & fault monitoring agent. In section 3, we propose an efficient fault tolerance algorithm i.e. Proactive Fault Tolerant algorithm for Job Scheduling (PFTAJS) & terminology used in algorithm. Section 4 discusses brief experimental details. Section 5 compare results of PFTAJS algorithm and compares results with post active fault tolerance heuristics.

## 2. Related Work

To achieve the promising potentials of dynamic & heterogeneous resources & for smooth working of grid, the fault tolerance is fundamentally requirement for grid computing. Reference [4] describes fault tolerance *"to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service.*" The probability of a failure is more when we are using geographically distributed resources comparatively to traditional parallel & cluster computing. In grid computing many parameters like makespan time, execution cost, QoS, user agreement etc. effected, if the resources were crashed during execution of jobs. Hence efficiency of resources will be optimized by identify the faulty resources & take corrective measure either before or after fault occur.

Fault tolerance is reflected by two parameters i.e. reliability & availability [5]. Reliability states that a system can continue to work without any interruption. It is closely defined by Mean Time to Failure (MTTF) and Mean Time between Failures (MTBF) & Mean Time to Repair (MTTR). MTTF is the average time the system operates until a

 

failure occurs, whereas the MTBF is the average time between two consecutive failures & MTTR, we obtain ***MTBF=MTTF+MTTR***. *Availability* is defined as the probability that resources are immediately available for smooth working of grid & restore normal working after some failure. ***Availability=MTTF/MTBF=MTTF/ (MTTF+MTTR).***

Fault tolerance techniques are often used to increase the availability and reliability [3].The fault tolerance service is essential to satisfy QoS requirements in grid computing and it deals with various types of resource failures, which include process failure, processor failure and network failures, application failure, violating timing deadlines and Service Level Agreement (SLA). When system fail due to occurrence of any kind of fault, fault tolerance technique take corrective action to recover the system from fail state. This can be done in three phases. The *first* step is identifying the kind of fault. In *second* step call the grid information service to explore other available resources. The *third* step is mapping the task to the best possible available resource which maintains QoS.

**Fault Tolerance Techniques**

This section, discuss various fault tolerance techniques for avoiding occurrence of faults in grid & improve the performance of grid nodes. Working of Grid fault tolerance techniques are divided into two mechanisms i.e. pro-active and post-active

I. **Post-active Fault Tolerance:** Whereas, post –active strategy handles the task failures after it has occurred and no failure consideration is made before the execution of tasks. However, post- active mechanism is applicable only in dynamic environment [5]

a. **Rescheduling:** When a node fails to execute assigned task, the task must be rescheduled to a different node. Rescheduling which changes previous schedule decisions based on a fresh resource status, can be taken as a remedy. There is a contract between user and the resource provider. A contract states that provider provides certain resources to user to achieve a specified satisfaction level of performance. When a contract violation is detected, a rescheduler will be activated & determine whether a rescheduling is profitable, based on completed work, estimates of the remaining work, and cost of moving to new resources. If the rescheduling appears profitable, the rescheduler will compute a new schedule. Rescheduling using two mechanisms rescheduling by stop and restart (RSR) and rescheduling by processor swapping (RPS) [6].

i. In the RSR approach, an application is suspended and migrated only when better resources are found. When a running application is signaled to migrate, executing process are suspending & rescheduling module is launch by restarting the application on the new set of resources after reading checkpoints and continue the execution of tasks. Restarting approach is flexible but, it could be very expensive because each migration event can involve large data transfers & incur expensive start-up costs.

ii. RPS is based on duplication heuristics. Same application is launched to more than one computation machines from which some machines are become part of computation (active set) while other do nothing initially (inactive set). During execution, contract monitor periodically performance of machines in active set and if it finds that machines does not perform according to contract it will swaps slower machines from active set with faster machines in the inactive set.

b. **Checkpoint/restart** [7]: Checkpoint/restart is most typical approach in parallel & distributed system & widely adopted by many fault tolerance methods. Checkpoint periodically capture snapshots (or add checkpoints) during normal execution of application & saved to a stable storage. These checkpoints are then used to restore the application to a previous execution state after the occurrence of fault. Restart technique is helpful to reduce the computation lost by restoring the computation &

execution will be start by resuming from recently added check point rather than resuming the execution from the beginning. Check pointing is primarily used to avoid losing all the useful processing done before a fault has occurred

c. **Retrying**: Another simplest technique known as Retrying [8]. After occurrence of fault , it will retries the same task regardless the cause of failure on the same grid resource, up to some threshold value and hopes that failure will not come in successive retries.

**II. Proactive Fault Tolerance:** Unlike post active, which takes action when a failure is detected, a proactive tolerance monitors the grid for possibilities of failures and track of memory consumption of an executing process, hardware conditions, network resources and component mean time to failure (MTTF). Proactive mechanisms consider the status of grid before the actually scheduling decision is made. Propose number of different agents to take corrective action before actual fault occur [9].

**a. Hardware Fault Tolerance:** Hardware Fault Tolerance using two agent i.e. *hardware monitoring agent (HMA)* and *hardware rebooting agent (HRA)*. HMA maintains history of every node which includes historical information about node availability, previous history, pings, MTTF and life. Based on this information agent assigns score to each node. Scheduler will consider node with the highest score. HMA also score the node which is in running mode. If the score of such a node falls below a threshold, scheduler shifts the task to another machine with a better score.

**b. Application and Operating System Fault Tolerance:** A *memory usage monitoring agent (MMA)* deals with the memory leaks in applications. It keeps track of total memory utilization during a specific time interval. If the memory utilization for an application is high, the state of the application is saved and it is rescheduled to another node. A *micro rebooting agent (MRA)* monitors the processes running on a node. If process is consuming exceptional amount of memory and has an unhealthy growth, it is killed and restarted. The *resource tracker agent (RTA)* keeps a record of resources available on the system and whether they meet application needs or not. It also monitors the resources to consider possibility of deadlock.

**c. Network Fault Tolerance:** Two agents have been designed to counter network errors: *network monitoring agent (NMA)* and *node failure tracking agent (NFTA)*. NMA monitors the send-receive status of network packets to check whether the packets have been received properly or not. If packet loss or corruption is above a certain threshold, the job is migrated to another system. The NFTA keeps information about maximum network load, current network load, MTTF, and MTBF of individual nodes and decides about network state also and checks for malicious network attacks. Based on this information, NFTA may decide to migrate an application to another node.

- Reference [5] gives another method to detect occurrence of fault in any grid by using two models i.e**. push or the pull model**. In the push model, Grid nodes send heartbeat messages to a failure detector time to time & declare that they are alive. But in pull model, failure detector sends message periodically to Grid components to check nodes are alive or not.
- **User defined exception handling** – In user defined exception handling technique [10][11] user specifies the particular treatment to workflow of a task on failure.
- **Rescue workflow** – The rescue workflow technique [10] allows the workflow to continue even if the task fails until and unless it becomes impossible to move forward without catering the failed task.
- Hwang, S. and Kesselman, C. [12] present a **failure detection service (FDS)** and a flexible failure handling framework (Grid-WFS) that provide generic failure detection service to detect failures without requiring any modification to Grid protocol and the local policy of each node.

 

❖ *Phoenix* [13] is a transparent middleware-level fault-tolerance layer that transparently makes data-intensive grid applications fault tolerant. It detects failures early, classifies failures into transient and permanent, and appropriately handles the transient failures.

## 3. Proposed PFTJS Algorithm

Proposed algorithm is based on the concept of Proactive Fault Tolerance which follows the idea, "Prevention is better than cure". If the faulty nodes are consider before actual scheduling or during the execution & take corrective action, this step will minimize execution time, increase execution rate, reduce faults occurrence & execution cost. As discussed in survey there are number of proactive agents who monitor health of nodes, application & network. Proposed algorithm use advantage of these agents to detect faults & its nature & take action before occurrence of any fault. Proposed algorithm using following terminology to define proposed technique:

- **Performance index** of resource, based on past historical information & other parameters like workload, availability, previous success rate, pings, MTTF and MTBF of node. There parameter have some weight which is increment or decrement whenever any event is occurring. Based on this information performance index give score to each node. If index value is between from 1 to 20, that is choose as satisfactory level.
- **Resource life table (RLT):** RLT is maintained by GIS (Gird information service) & stores performance index value for every node.
- **Shifting cost:** { ( Execution cost at new suitable node + migration cost ) – (Investment cost for completed work before latest checkpoint on faulty node) }
- **Completion time:** Execution time + Performance index value
- **Execution time:** it is a time used by node $n_j$ to execute job $j_i$.

**PFTJS Algorithm:**

1. User submits task with QoS parameters like deadline, execution cost etc.
2. GIS have complete list of available nodes with complete life history maintained through Resource life table**.**
3. Scheduler will select suitable number of nodes R1, R2, R3…..$R_i$ which fulfill task's QoS & goto step 4.
4. Task assignment to best node is based on one of following decision:
   a. If Performance index value of node $R_i$ is 0, assign task & update its corresponding index value according to various performance parameter like workload, availability, previous success rate, pings, MTTF and MTBF & go to step no 5
   b. If Performance index value of node $R_i$ is 1 to 20, calculate completion time of task at every suitable resource. Assign resource which gives minimum completion time & go to step no 5
   c. If index value of any resource $R_i$ is above 20 GOTO step 2 & search fresh list of resources
   5. Add checkpoint after the 30% completion of every task.
6. During execution, monitoring agent predict node performance index of every $R_1$, $R_2$, $R_3$…..$R_i$
   ***a.*** If performance index value is in satisfactory level then execution goes continue and wait for results.
   ***b.*** If performance goes down due to overloading of work or high performance index value then task may be shifted from $R_i$ to another suitable $R_j$ after calculating shifting cost according to following steps.

***i.*** If shifting cost is below than threshold (i.e. profitable) then shifting of job will be done.

***ii.*** If shifting cost is more than threshold (i.e. non profitable), set index value of that node to 0 & GOTO step 2 for rescheduling of job.

***iii.*** Else execution is going on same node and waits for result.

7. Go to step 2 ,until job queue is not empty

## 4. Experimental Detail

GridSim Toolkit–4.0 used for the simulation of proposed algorithm & to analyze the performance of PFTJS algorithm with post active fault heuristics. The performance of PFTJS algorithm analyzes in term of total execution time of jobs, number of faults occurs & execution cost. Results were analyzed by executing 1000 jobs with same size (i.e. equal MI) on varying number of grid nodes. Node's power defined in term of MIPS.

## 5. Result & Conclusion

First experiment is done with PFTJS algorithm to evaluate execution time and compare with the post active fault heuristics. First column of table 1 indicate number of nodes used to test proposed algorithm, second & third column gives execution time used by PFTJS algorithm & post active heuristics respectively to complete the execution of all jobs.

**Table 1. Execution Time**

| No of nodes | Execution time by PFTJS algorithm | Execution time by post active fault heuristics |
|---|---|---|
| 10 | 147 | 160 |
| 50 | 98 | 120 |
| 100 | 68 | 92 |
| 200 | 38 | 52 |

If analyze above results shown in figure 1, total execution time to complete the tasks by PFTJS algorithm will be less as comparatively to post-active fault heuristics even in the presence of faults. Because during execution if some nodes goes down then post-active fault heuristics, choses one of corrective measure may be restarting the job on same node, rescheduling on some different node or wait to normal functioning of node. While proposed algorithm take corrective measure before actual fault occur & also it is use check pointing technique, which will save time to restart the job staring on some second safe node.

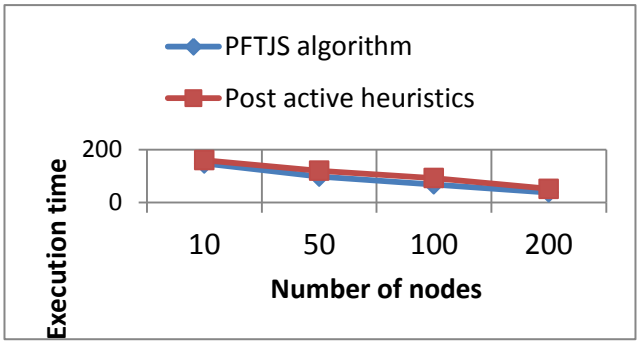


**Figure 1. Execution Time**

Occurrence of faults was also analyzed through PFTJS algorithm & post-active heuristics, shown in table 2.

**Table 2. Fault Occur**

| No of nodes | Number of faults in PFTJS algorithm | Number of faults in post-active heuristics |
|---|---|---|
| 10 | 50 | 95 |
| 50 | 31 | 58 |
| 100 | 19 | 42 |
| 200 | 10 | 22 |

Figure 2 shows, number of faults occur using PFTJS are very much less as faults occurs in post-active heuristics. Since PFTJS algorithm monitors status of nodes by considering workload, availability, pings, MTTF and MTBF & past historical history & set performance index value. If this value crosses from defined threshold value, there may be chance of node crash. To prevent grid system from faults, PFTJS algorithm takes corrective measure before occurrence of such condition in grid & reduces the probability of faults. Also as number of nodes are increased probability of faults occurrence will be also reduce in both heuristics

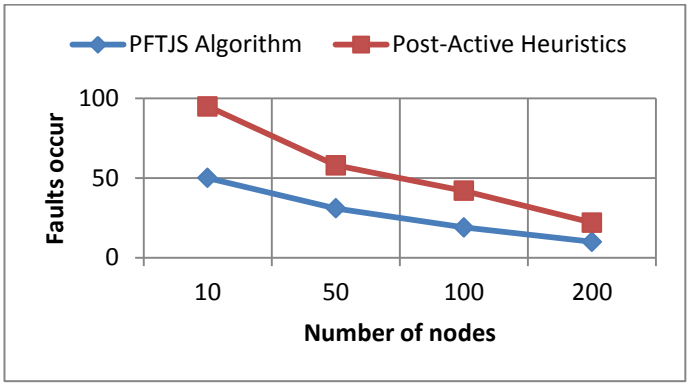


**Figure 2. Fault Occur**

Third objective of proposing PFTJS algorithm is to reducing execution cost & compare with post-active heuristics, shown in Table 3.

**Table 3. Execution Cost**

| No of nodes | Execution cost in PFTJS algorithm | Execution cost in Post-active heuristics |
|---|---|---|
| 10 | 76 | 96 |
| 50 | 48 | 68 |
| 100 | 30 | 46 |
| 200 | 17 | 33 |

In figure 3 execution cost of job was analyzed between PFTJS algorithm & post-active heuristics**.** If faults are occur in grid system, post-active heuristics use either rescheduling or wait to recover node from fault. This will increase cost of execution as well as degrading QoS by violating deadline for completion of task. But in case of PA if any node goes down or performance index value is high, PFTJS algorithm calculate shifting cost & if cost is below threshold i.e. profitable then resume the pending task from latest checkpoint to other health node.

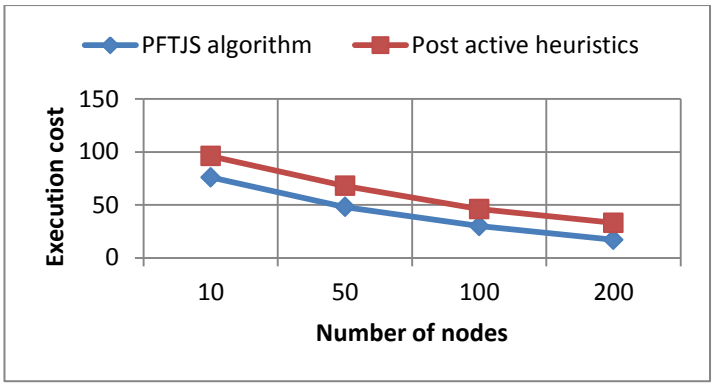


**Figure 3. Execution Cost**

## References

[1] Mohammad Tanvir Huda, Heinz W. Schmidt, and Ian D. Peake, " An agent oriented proactive fault-tolerant framework for grid computing", Proceedings of the 1st International Conference on e-Science and Grid Computing, Washington- USA (2005), pages 304–311
[2] Paul Townend and Jie Xu," Fault Tolerance within a Grid Environment", IEEE Second International Conference on Computer Engineering and Applications(2009)
[3] P. Latchoumy and P. Sheik Abdul Khader," Survey on fault tolerance in grid computing", International Journal of Computer Science & Engineering Survey, Vol.2, No.4(November 2011).
[4] A. Avizienis, "The N-version Approach to Fault- Tolerant Software" - IEEE Transactions on Software Engineering - vol. 11(1985) .
[5] Babar Nazir, Taimoor Khan," Fault Tolerant Job Scheduling in Computational Grid", 2nd International Conference on Emerging Technologies, IEEE-ICET, 13-14 Nov. (2006), pp. 708 - 713
[6] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z.Shi, A. YarKhan and J. Dongarra, "New Grid Scheduling and Rescheduling Methods in the GrADS Project", in Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, New Mexico USA(2004), April, pp. 199-206
[7] Kalim Qureshi, Fiaz Gul Khan, Paul Manuel, Babar Nazir," A hybrid fault tolerance technique in grid computing system", J Supercomputing, Vol. 56(2011), pp. 106–128.
[8] Gartner Felix C," Fundamentals of fault-tolerant distributed computing in asynchronous environments", Journal of ACM (1999), pp 1–26.
[9] Mohammad Tanvir Huda, Heinz W. Schmidt, and Ian D. Peake," An agent oriented proactive fault tolerant framework for grid computing", Proceedings of the 1st International Conference on e-Science and Grid Computing, Washington, DC, USA (2005), pp 304–311,
[10] Yu Jia, Buyya Rajkumar," Taxonomy of workflow management systems for grid computing", Journal of Grid Computing (2006), pp. 171–200.
[11] Gioiosa Roberto, Sancho Jose Carlo, Jiang Song, Petrini Fabrizio, Davis Kei," Transparent incremental check-pointing at kernel level: a foundation for fault tolerance for parallel computers", Proceedings of the ACM/IEEE conference, 12-18 Nov. 2005, pp-9
[12] S. Hwang and C. Kesselman. A flexible framework for fault tolerance in the grid. *Journal of Grid Computing*, pages 251–272, 2004.
[13] G. Kola, T. Kosar, and M. Livny. Phoenix: Making data-intensive grid applications fault-tolerant. In Proceedings of 5th IEEE/ACM International Workshop on Grid Computing, 2004.

## Authors

**Sarpreet Singh**, He has done Master in the area of Grid Computing from Department of Computer science, Punjabi University, Patiala, INDIA. Presently he is working as Assistant professor at Sri Guru Granth sahib World University, Fatehgarh sahib & doing his Ph.D from Punjabi University, Patiala, INDIA. His area of interest is distributed computing

**Rajesh K. Bawa**, He has done Master's and Ph.D degree in the area of Numerical Computing from IIT Kanpur INDIA. Presently he is working as Professor in Department of Computer science, Punjabi University, Patiala, INDIA. His present area of interest is Parallel and Scientific Computation. He has published many research papers in international & national journals.

 