International Journal of Computers & Technology
Volume 4 No. 3, March-April, 2013, ISSN 2277-3061

Fault Tolerant Heterogeneous Limited Duplication Scheduling algorithm for Decentralized Grid

Neha Agarwal, Piyush Chauhan and Nitin, *Senior Member, IEEE*
Jaypee University of Information Technology,
P.O. Waknaghat, Solan-173234,
Himachal Pradesh, India.
{lko.neha, shbichauhan, delnitin}@gmail.com

ABSTRACT

Fault tolerance is one of the most desirable property in decentralized grid computing systems, where computational resources are geographically distributed. These resources collaborate in order to execute workflow applications as fast as possible. In workflow applications, tasks are dependent on each other, so it becomes extremely vital that scheduling techniques should also have some decentralized fault tolerant mechanism. In this paper, we have proposed a decentralized fault tolerant mechanism which utilize the checkpoint concept; for Heterogeneous Limited Duplication (HLD) algorithm. HLD is based on task duplication scheduling in heterogeneous environment. There are two fold benefits firstly; if node failure occurs then rest of grid nodes sustain the execution of application. Secondly, less makespan of application is obtained using checkpoint concept. Therefore, application scheduled over decentralized grid systems (which are known for their unreliable behavior) will yield results fast utilizing algorithm proposed in this paper.

General Terms

Grid Computing, Fault Tolerance

Indexing terms

Grid computing; dependent task scheduling; task duplication strategy; decentralized fault tolerance.

Academic Discipline And Sub-Disciplines

Grid Computing

SUBJECT CLASSIFICATION

Computer Science and Engineering

COVERAGE

Task Scheduling Algorithm

TYPE (METHOD/APPROACH)

Experimental

INTRODUCTION

In current scenario, grid computing has gained a lot of attention for executing application in parallel fashion on geographically distributed systems. Computational intensive applications consist of a lot of tasks. These tasks are dependent on one another. Hence weighted directed acyclic graph (DAG) can be used to represent a Computational intensive application which is shown in figure 1. [1]. In DAG the edges represent dependency among tasks and nodes represent tasks of application. We schedule these tasks upon grid using various Task scheduling strategies. Task scheduling strategies are classified into three categories for DAG; applications-list scheduling [3,4,5],cluster based scheduling [7] and duplication based scheduling[1,2,6,8-11]. In list based scheduling, a task sequence is generated on the basis of priority and according to that tasks are scheduled on grid nodes. In clustering scheduling, tasks which communicate more with each other, are grouped and assigned to same cluster so that communication delay is reduced. In duplication scheduling, the parent task can be duplicated on other grid node in its idle time slot if communication cost is high. By doing that the makespan (total time of execution) of application is reduced. Although it is finest technique for dependent task scheduling, it has a shortcoming that resources can be over consumed if heavy duplication is needed. Beside that task duplication scheduling is best one for DAG application in which communication latencies among task is high because by duplicating parent tasks makespan of application can be easily trimmed down.

Savina et al. in 2005 recommended the heterogeneous limited duplication (HLD) algorithm [2] that is based on the concept of the SD algorithm [15] in heterogeneous environment and then estimated the effectiveness of limited duplication approach while dealing with the strains of heterogeneity in a system. Dogan et al. in 2002 put forwarded a level sorting (LDBS) algorithm [12] in which tasks in DAG are categorized into various precedence levels. The tasks which are at the same level having no data dependencies can execute in parallel. Tasks are scheduled level wise in LDBS starting from the top level task. In 2010 Amit Agarwal et al. proposed EDS-G algorithm which is also based on task duplication and in this algorithm the unnecessary duplicated tasks are removed if they are not affecting the makespan of application.

In heterogeneous environment resources may fail frequently and it will affect the execution of the application. So it is desired that scheduling techniques should have some fault tolerance mechanism.Without any fault tolerance mechanism if fault occurs in any grid node then we have to reschedule all tasks again and it will increase the schedule length (completion time) of the application. A fault tolerant approach will be beneficial in order to potentially prevent a malicious resource affecting the overall performance of the application. In this paper we are proposing a fault tolerance mechanism on HLD algorithm (HLD-FT) by using the checkpoint strategy. According to that proposed algorithm if any fault occurs then there is a server who will detect the failed grid node and it will handle the execution of application by resuming the execution from the maintained checkpoints.

The paper is organized as follows: Section 2 presents a review of the related work in this field. In Section 3 task scheduling problem is described in brief by using the DAG. Section 4 is presenting our proposed algorithm that is HLD-FT. The simulation results and analysis part is described in section 5.Finally, Section 6 concludes the paper and presents future work.

RELATED WORK

Dependent Task Scheduling Techniques

The dependent task scheduling algorithms can be classified into a variety of classes, such as list scheduling algorithms, cluster algorithms, and duplication-based algorithms. The list scheduling algorithms afford a good quality of makespan and their performance is as good as the other classes at a lower time complexity. Some examples are: dynamic critical-path (DCP) [16], heterogeneous earliest finish time (HEFT) [17], critical path on a processor (CPOP) [17] and the longest dynamic critical path (LDCP) [18]. Cluster algorithms unite tasks in a graph to an unrestricted number of clusters and tasks in a cluster are scheduled on the same grid node. Some examples in this class are clustering for heterogeneous processors (CHP) [19], clustering and scheduling system (CASS) [20], objective- flexible clustering algorithm (OFCA) [21]. The proposal of duplication-based algorithms is to schedule a task graph by mapping some tasks redundantly, which lessens the interprocess communication overhead. There are various duplication-based algorithms, for example, selective duplication (SD) [15], heterogeneous limited duplication (HLD) [2], heterogeneous critical parents with fast duplicator (HCPFD) [14], and heterogeneous earliest finish with duplication (HEFD) [13], Economical Task Scheduling Algorithm for Grid Computing System (EDSG) [1]. This class of algorithms can trim down the makespan effectively, but it is traded with huge amount of energy consumption.

Grid Fault Management

A vast number of task scheduling algorithms for DAG applications have been put forwarded. But most of the existing algorithms presume that the processors of the system are completely in safe hands, so they do not tolerate any breakdown in the system components. In heterogeneous environments, workflow execution failures can transpire for various causes such as network failure, overloaded resource circumstances, or non-availability of required software components.

A variety of approaches are exploited for tolerating faults in grid, so grid fault management can be classified as :

Pro-active and Post-active management

In literature, the work of tolerating fault in grid nodes can be divided into pro-active and post-active strategy. In pro-active strategy, before executing the task the failure consideration for grid is made and tasks are assigned to the grid nodes in the hope that no fault will be occurred. Whereas, post-active strategy handles the task failures after it has occurred and no failure consideration is made before the execution of tasks. However, in the dynamic systems only post-active mechanism is applicable [22].

Push Model and. Pull Model

In order to identify occurrence of fault in any grid node two approaches can be used: the push or the pull model. In the push model, grid components sporadically send heartbeat messages to a failure detector server declaring that they are alive and no fault is occurred. In the absence of any heartbeat message from any grid node, the fault detector server identifies that failure has occurred at that grid node.

After the identification of failure it employs suitable measures dictated by the predefined fault tolerance mechanism. In contrast, in the pull model the failure detector server sends some request messages (“Are you alive?” messages) sporadically to grid nodes in order to check the fault.[23].If no reply message is received from any grid node by the failure detector server then it can easily identify that at which grid node failure has occurred.

The checkpointing is one of the most popular techniques to provide fault-tolerance on unpredictable systems. It is a trace of the snap of the entire system state in order to reschedule the application if some fault is occurred at any grid node. The checkpoint can be maintained on temporary as well as stable storage. However, the effectiveness of the mechanism is strongly reliant on the span of the checkpointing interval. Frequent checkpointing may boost up the overhead, while lazy checkpointing may go ahead to loss of significant computation.

TASK SCHEDULING PROBLEM

In a grid computing system when a dependent task scheduling model is considered then it comprises 1) a target grid computing system in which plentiful computing grid nodes are connected, and 2)A DAG which demonstrates the dependency among different tasks and communication cost in communicating desired data among different tasks.

Grid Resource Model

A grid computing system can be represented by GR (GN, CE) where GN is the set of k capricious connected computing grid nodes (1, 2, 3..... k) which are geographically distributed and together form grid system and CE is the set of communication edges which unite grid nodes. As in grid computing system, grid nodes are heterogeneous and geographically separated, so the execution time of tasks on different grid nodes will be different. In this grid resource model some points are presumed which are as follows:

1) Execution of tasks on grid node are non preemptive.

2) When two tasks are scheduled on a similar grid node then communication overhead between these tasks will be negligible.

3) There is a co-processor attached with each grid node which will deal with communication among the grid nodes so that communication and computation both can be done in parallel fashion.

Table 1. Computation cost matrix [cc_{ij}] for dag in figure.1

Task	Computation Costs on different Grid nodes				Mean Costs
	gn_0	gn_1	gn_2	gn_3	$\overline{mp_i}$
t_0	1	1	2	1	1.25
t_1	3	2	4	2	2.75
t_2	5	6	3	4	4.5
t_3	2	4	4	2	3.0
t_4	4	8	7	8	6.75
t_5	3	3	1	2	2.25
t_6	5	5	5	5	5.0
t_7	1	2	2	2	1.75

Grid Application Model

A grid computing system can be represented by a quadruple GM=(T, PR, [cc_{ij}], [$m_{i,k}$]) where T = {t_0, t_1, t_2 -------t_{n-1}} is a set of n tasks, PR specifies the precedence relation t_i R t_j that is task t_i must finish before t_j can carry on its execution, [cc_{ij}] is a n×n matrix of communication cost that gives amount of dataflow between task t_i and t_j for 0≤ i, j <n, and [$m_{i,k}$] is a n×m matrix of task's execution time on different grid nodes for 0≤ i <n and 0≤k < m (Table 1). This grid computing system can better be demonstrated by weighted directed acyclic graph(DAG) (Figure 1) where tasks are signified by vertices and dependency among them is signified by edges, and weights along them represents mean computation cost $\overline{mp_i}$ and mean communication costs $\overline{cc_{ij}}$, respectively.

$$\overline{mp_i} = \frac{\sum_{k=1}^{m} mp_{i,k}}{m} \qquad 1\le i \le n \text{ and } 1\le k\le m \tag{1}$$

$$\overline{cc_{ij}} = \frac{cc_{ij}}{\text{average data transfer rate over the links of grid}} \tag{2}$$

International Journal of Computers & Technology
Volume 4 No. 3, March-April, 2013, ISSN 2277-3061

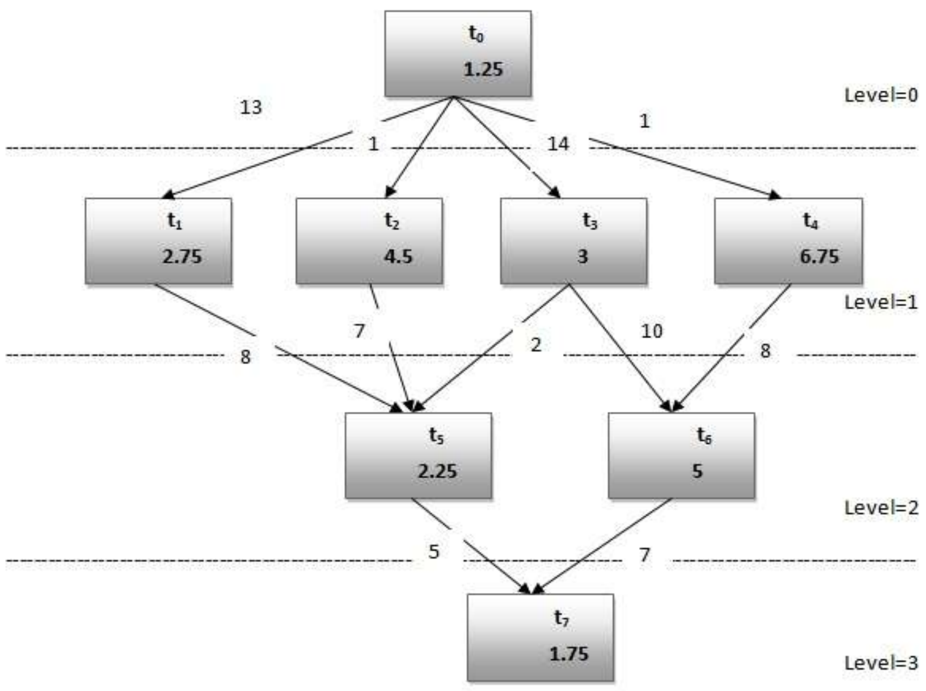

Fig 1: Weighted Directed Acyclic Graph (DAG) with Precedence Constraints

PROPOSED ALGORITHM

As we have discussed above that in heterogeneous environment fault tolerant mechanism is necessary, so here we are proposing a fault tolerance mechanism along with HLD algorithm that is HLD-FT to handle the execution of any workflow application if fault occurs. In dependent task scheduling HLD algorithm is the finest one so we are incorporating fault tolerance mechanism in that scheduling. According to our proposed algorithm if no failure occurs at any grid node then the normal execution of HLD algorithm will be going on but in presence of fault FT(GN, ø) algorithm will be called to run fault tolerance mechanism.

Pseudo Code for HLD-FT Algorithm is shown in figure 2.Firstly the application will be originated at one grid node that will be origin node and it will distribute the tasks on other grid nodes. In this we are assuming that our origin grid node is gn_0 and it will distribute tasks on all grid nodes GN. In the normal execution of HLD algorithm firstly priority sequence of the tasks ø is to be constructed which is based on their levels. We will use following equation to generate it:

$$ps_i = \overline{mp_i} + \max\{ps_j + \overline{cc_{ij}}\} \; \forall t_j \in \text{successor}(t_i) \qquad (3)$$

Here in equation 3 successor (t_i) is the set of immediate child nodes of task t_i in the DAG. After constructing the task sequence based on priority (i.e. ø), first unscheduled task from the sequence will be picked up. Now we will find out the finishing time FNT_{ik} of task t_i on different grid nodes GN. At the grid node gn_k on which finishing time is minimum we will schedule that task t_i. A task on a grid node can only execute when that grid node have all required data of all its immediate parents. The parent of task t_i whose data received last of all at the grid node is termed as the most important immediate parent (MIIP).

$$DA(t_i, gn_k) = \max_{t_j \in ipr\,(t_i)} \{ \min\{FNT_{jk}, FNT_{jk'} \quad + cc_{ji}\} \qquad (4)$$

In the above equation ipr (t_i) is a set of all immediate parent nodes of task t_i in the DAG. To calculate the earliest finishing time (FNT) of the task firstly we should know that at which time the task can start its execution on that grid node. So starting time of the task is totally dependent on data arrival from its MIIPs. There are two methods to find out the data of the set of immediate parents (MIP_i) which is needed for that execution of task: 1) Duplicating the immediate parent MIPi in the suitable idle time slot IIS_k on that grid node gn_k, 2) By communicating to the grid node where that parent MIP_i is scheduled for required data. If communication cost is high between the tasks then we adopt first method which is duplicating task otherwise second one. So starting time STT_{ik} of t_i on grid node gn_k can be figured out by following equation

$$STT_{ik} = \max\{DAT(MIP_i, gn_k), \min\{r_k^R, IIS_k\}\} \qquad (5)$$

In this equation IIS_k is the start time of first idle slot at which MIP_i can be executed on the grid node gnk. After having the start time of task ti on the grid node gnk , finishing time of task FNT_{ik} can be figured out by using following equation:

$$FNT_{ik} = STT_{ik} + mp_{ik} \tag{6}$$

Here mp_{ik} is the execution time of task t_i on grid node gn_k. After having the finishing time of task t_i on all grid nodes GN, determine the minimum finishing time. The grid node gn_k at which the finishing time FNT_{ik} of task t_i is minimum the task will be scheduled to that grid node. In the presence of any fault on grid node proposed algorithm will call FT(GN,ø) fault tolerance mechanism. In this mechanism there is a failure detector sever (FD) which works on push method to detect the failure in grid node. A heartbeat message (h) is periodically sent by all the grid nodes GN to FD. If from any grid node gnf reply message is not arrived at FD then it comes to know that at grid node gn_f fault is occurred. In this we are assuming that FD will be that grid node on which application will be originated. As in this the origin grid node is gn_0 so it will work as fault detector server as shown in figure 3. If this origin grid node gn_0 fails then we don't need to go further as origin grid node has failed.

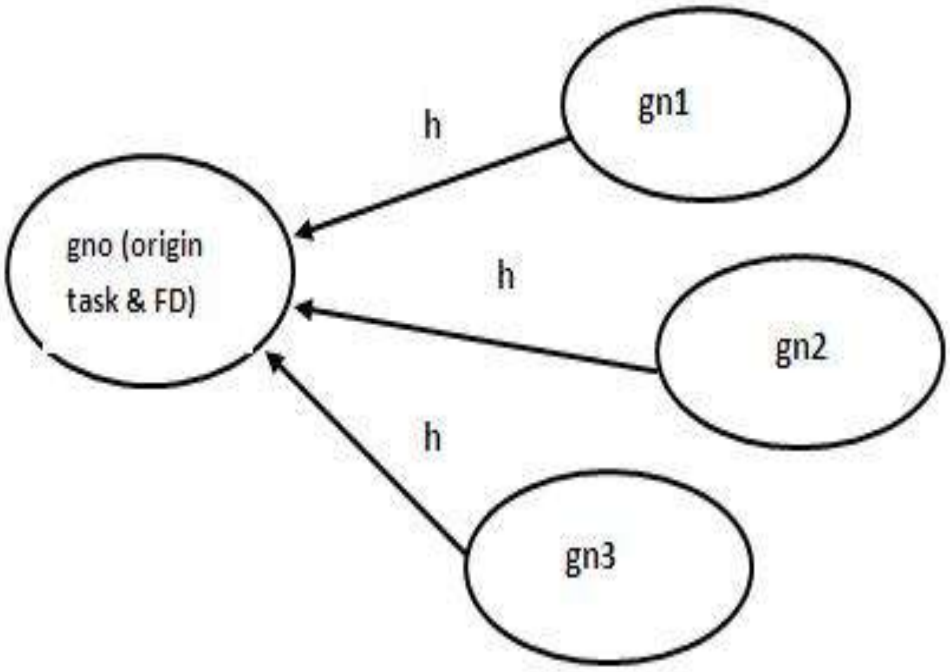

Fig 3: Origin task working as fault detector server

In addition to sending heartbeat message(h),all grid nodes GN will also tell the FD that which task is scheduled on that and by using this information FD will maintain checkpoints. The main motive of maintaining checkpoints is that if any fault is occurred at any grid node then there is no need to reschedule all the tasks.

Checkpoint will tell that from which task tf the execution of the application can be resumed. So now from the task t_f in sequence ø the task will be rescheduled and earliest finishing time of task is calculated on grid nodes GN except on gn_f(GN-gn_f). The grid node gnk at which the finishing time FNT_{ik} of task t_i is minimum the task will be scheduled to that grid node.

```
HLD-FT

Begin
Step 1:   Generate task sequence øbased on priority;
Step 2:   if (no fault occurs in grid nodes)
          {
            Do
                    Step 2a:    $t_i$ ← first unscheduled task in ø ;
                    Step 2b:    for (all $gn_k$ in GN)
                                  Call Cal_EFT ($t_i$, $gn_k$) and record the $FNT_{i, k}$;
                    Step 2c:    Find out $gn_k$ on which $FNT_{i, k}$ is minimized
                    Step 2d:    Schedule $t_i$ on grid node $gn_k$ at which $FNT_{i,k}$ is minimized
                    while (there exists an unscheduled task in ø)
          }
else
Call FT(GN,ø) for running fault tolerance mechanism
End
```

```
Cal_EFT (t_i, gn_k)

Begin

Repeat
 {
Step s1:    $STT_{i,k}$ ← Find start time of $t_i$ on $gn_k$
Step s2:    $FNT_{i,k}$ ← $STT_{i,k}$ + $m_{i,k}$
Step s3:    $MIP_i$ ← Find MIIP of $t_i$ for $gn_k$
Step s4:    if ($MIP_i$ does not exist or is already scheduled on $gn_k$)
              Return $FNT_{i, k}$;
else if (suitable slot exists for $MIP_i$ on $gn_k$)
                {    $FNT_{i,k}$ ← Find the finish time of $MIP_i$ (= $t_i$ say) on $gn_k$
if ($FNT_{j, k}$ is less than $STT_{i, k}$)
                     Duplicate $MIP_i$ on $gn_k$; // duplication successful, repeat the loop for next MIIP.
else Return $FNT_{i, k}$
                 }
elseReturn $FNT_{i, k}$;
}
End
```

```
FT(GN,ø)

Begin

Step 1: Failure detector server will detect the grid node $gn_f$ on which fault occurs.
Step 2: Server will find out the task $t_f$ from the maintained checkpoints from which the task will be rescheduled now.
*Step 3:*Do

          Step 3a:    $ta_i$ ← Start unscheduled task in φ from
                                          the task $t_f$ which we get from step 2;
          Step 3b:    for (all $gn_k$ in (GN-$gn_f$) )
                        Call Cal_EFT ($t_i$, $gn_k$) and record the $FNT_{i, k}$;
          Step 3c:    Find out $gn_k$ on which $FNT_{i, k}$ is minimized
          Step 3d:    Schedule $t_i$ on grid node $gn_k$ at which $FTT_{i,k}$ is minimized
while(there exists an unscheduled task in $\varphi$)
End
```

Fig 2: Pseudo code for HLD-FT

RESULTS AND ANALYSIS

We applied HLD-FT algorithm for the DAG as shown in figure 1 on four grid nodes. We are referring table 1 for communication cost matrix. It is visible in figure 4 that, if in HLD there is no fault tolerance mechanism, then after the grid node failure we have to reschedule all the tasks on non faulty grid nodes. We have assumed that grid node 2 fails after executing task t_1. Here scheduling is non preemptive so before the execution of other tasks we can't do rescheduling. Note that t_i(D) is representing the duplication of that task. Figure 5 is showing what happens when we apply our proposed algorithm HLD-FT for the same DAG (as shown in figure 1).

In HLD-FT fault detector server will come to know that grid node 2 has failed and it will resume the execution of application on other grid nodes. It will reschedule the tasks from checkpoint (t_1) in task sequence øon grid nodes which are not faulty. Waiting time for individual tasks of application also decreases when we apply HLD-FT as shown in figure 6. Figure 7 is showing the comparison between the makespan if we apply HLD and our proposed algorithm HLD-FT. In HLD-FT makespan is less as compared to HLD when grid node failure occurs.

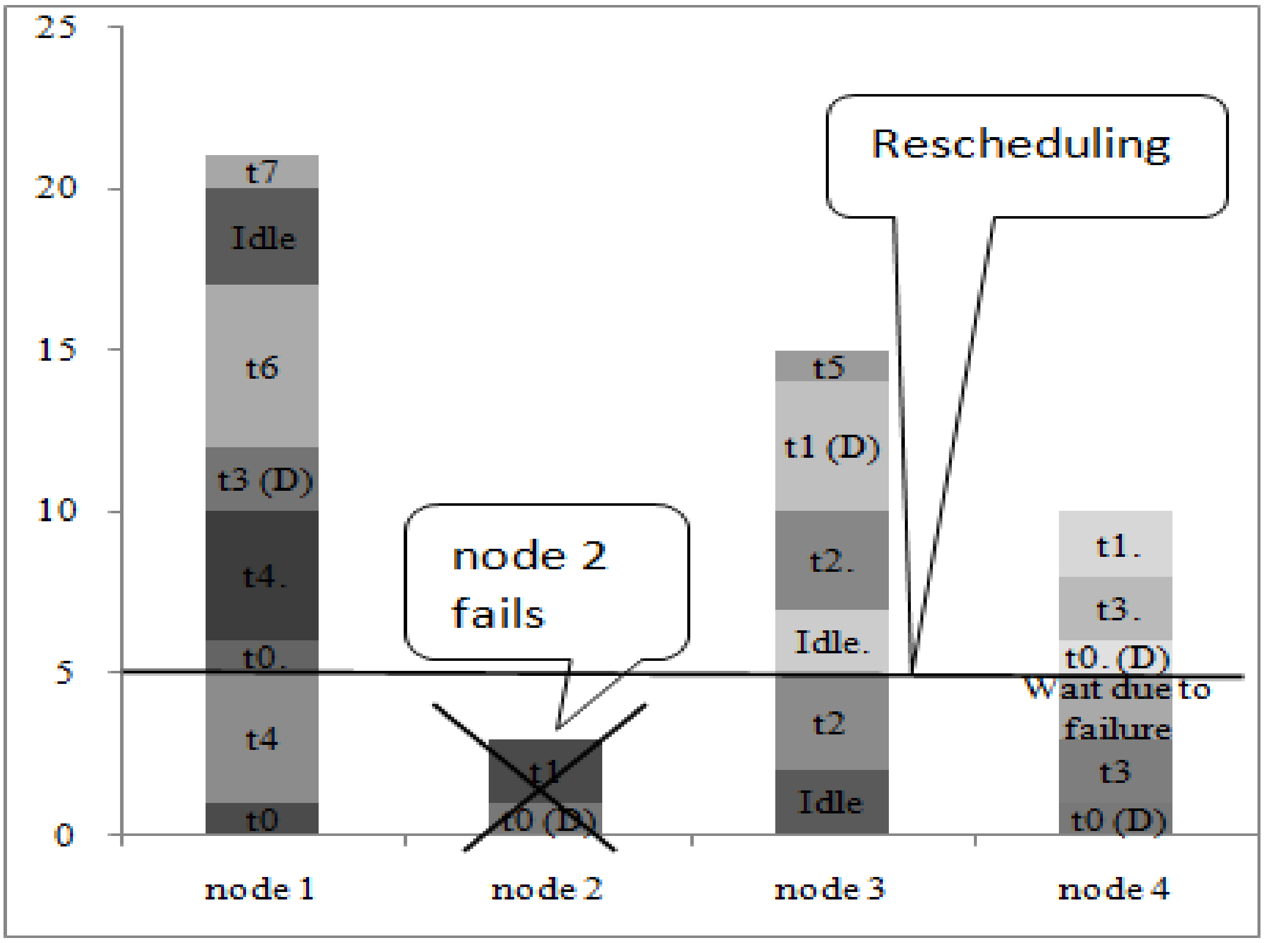

Fig 4: Detailed time wise (in seconds) schedule for all tasks of application using HLD algorithm.

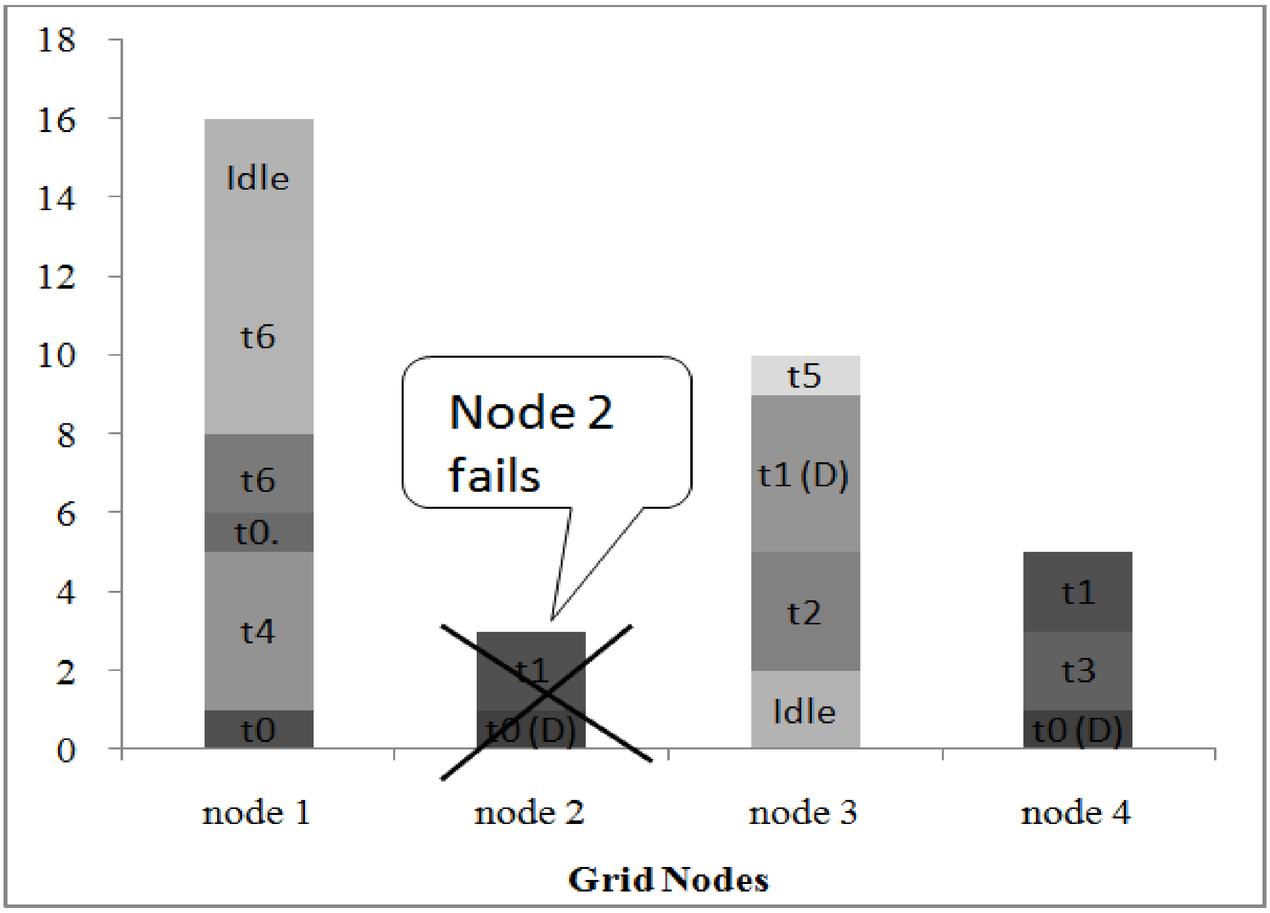

Fig 5: Detailed time wise (in seconds) schedule for all tasks of application using HLD-FT algorithm.

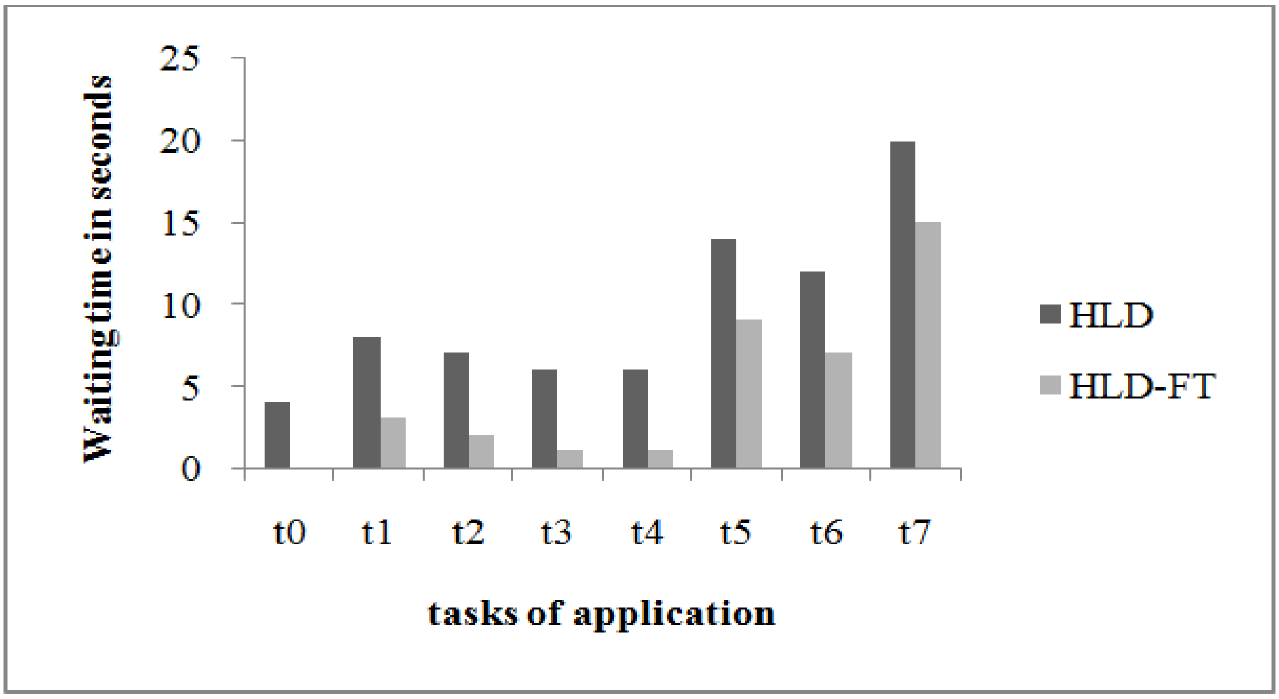

Fig 6: Waiting time for individual tasks of application when we schedule using HLD and HLD-FT algorithms

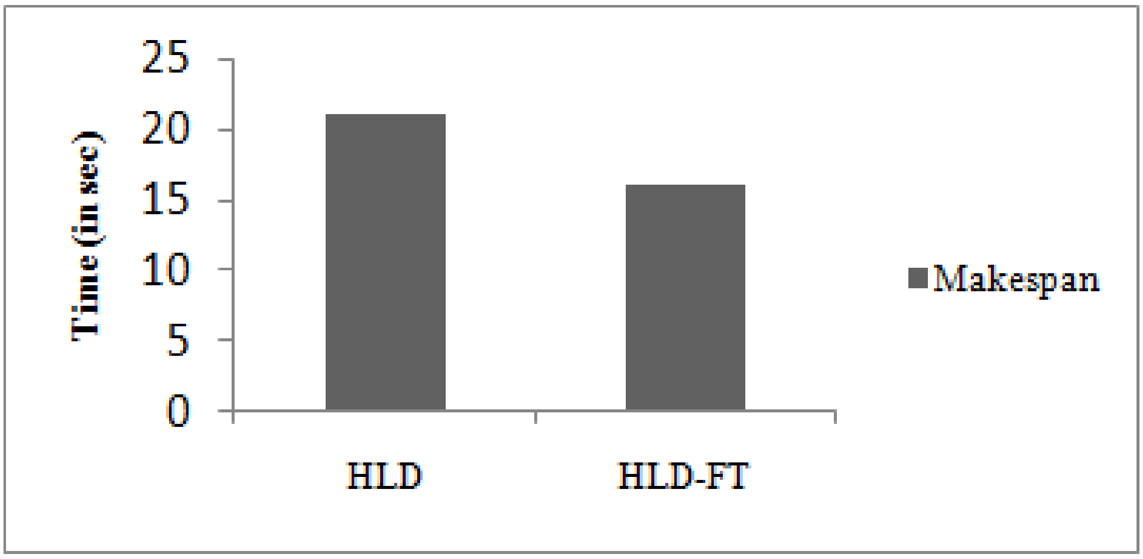

Fig 7: Makespan comparison between HLD and HLD-FT.

CONCLUSION AND FUTURE WORK

In all static dependent task scheduling strategies, task duplication is finest one to shrink the makespan of application. However, existing task duplication based scheduling algorithms are short of fault tolerant mechanism to overcome the unreliable nature of decentralized grid. Hence, we have proposed HLD-FT algorithm in which failure detector server detects the failure at grid node by utilizing push model and reschedule tasks through maintained checkpoint. Proposed fault tolerance mechanism is very effective in handling grid node failure. Results show that in case of node failure, HLD-FT gives minimum makespan as compared to HLD algorithm. In future, we want to add real time constraints in HLD-FT algorithm.

REFERENCES

[1] Agarwal,A., and Kumar,P. 2010. Economical Task Scheduling Algorithm for Grid Computing System. Global Journal of Computer Science and Technology, Vol. 10, 48-53.

[2] Bansal, S., Kumar, P., and Singh, K. 2005. Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. Journal of Parallel and Distributed Computing, Vol. 65, 479-491.

[3] Vincent B.B., and Yves, R. 2002. The Iso-level Scheduling Heuristic for Heterogeneous Processors. Euromicro workshop on parallel, distributed and network based processing, 335-342.

[4] Topcuoglu, H., Hariri, S., and Wu,M.Y. 2002. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Trans. on Parallel and Distributed System, Vol. 13(3), 260- 274.

[5] Hagras, T., and Janecek, J. 2003. A High Performance, Low Complexity Algorithm for Compile-Time Job Scheduling in Homogeneous Computing Environments. International Conference on Parallel Processing Workshops, 149-155.

[6] Sharma, R., and Nitin. 2012. Optimal Method for Migration of Tasks with Duplication. 14th International Conference on Computer Modelling and Simulation, 510-515.

[7] Sharma, R., and Nitin. 2011. Duplication with task assignment in mesh distributed system. World Congress on Information and Communication Technologies (WICT), 672-676.

[8] Lai, K.C., and Yang, C.T. 2008. A Dominant Predecessor Duplication Scheduling Algorithm for Heterogeneous Systems. Journal of Supercomputing,Vol. 44(2), 126-145.

[9] Ernemann, C., Hamscher, V., and Yahyapour,R.2002. Economic scheduling in grid computing. Workshop on Job Scheduling Strategies for Parallel Processing,Lecture Notes in Computer Science, Springer Vol. 2537, 128-152.

[10] Ahmed, I., and Kwok Y.K. 1998. On Exploiting Task Duplication in Parallel Program Scheduling", IEEE Trans. on Parallel and Distributed Systems, Vol. 9(9), 872-892.

[11] Hwang, S. and Kesselman, C.2003. Grid Workflow: A Flexible Failure Handling Framework for the Grid. In 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03), Seattle, Washington, USA.

[12] Dogan, A., and Ozguner, F. 2002. LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems. International Conference on Parallel Processing, 352-359.

[13] X. Tang, X., Li, K., Liao, G., and Li, R.2010. List scheduling with duplication for heterogeneous computing systems. Journal of Parallel and Distributed Computing,Vol. 70(4), 323-329.

[14] Hagras, T. and brevecek, J.J. 2005. A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. Parallel Computing,Vol. 31(7), 653-670.

[15] S. Bansal, S., Kumar, P., and Singh, K.2003. An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 14(6), 533-544.

[16] Kwok, Y.-K.and Ahmad, I. 1996. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. IEEE Transactions on Parallel and Distributed Systems, Vol. 7(5), 506-521.

[17] Topcuoglu, H., Hariri, S., and Wu,M.-Y.2002. Performanceeffective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, Vol. 13(3), 260-274.

[18] Daoud, M.I. and Kharma, N.2008. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, Vol. 68(4), 399-409.

[19] Boeres, C., Filho, J., and Rebello,V. 2004. A cluster-based strategy for scheduling task on heterogeneous processors. In 16th Symposium on Computer Architecture and High Performance Computing, (SBAC-PAD 2004),214-221.

[20] Liou J., and Palis, M.1996. An efficient task clustering heuristic for scheduling dags on multiprocessors. In Proceedings of Parallel and Distributed Processing Symposium,152-156.

[21] Fangfa, F., Yuxin, B., Xinaan, H., Jinxiang, W., Minyan, Y., and Jia, Z.2010. An objective-flexible clustering algorithm for task mapping and scheduling on clusterbased noc. In 2010 10th Russian-Chinese Symposium on Laser Physics and Laser Technologies (RCSLPLT) and 2010 Academic Symposium on Optoelectronics Technology (ASOT), 369-373.

[22] Medeiros, R., Cirne, W., Brasileiro, F., and Sauve, J. 2003.Faults in grids: why are they so bad and what can be done about it? In proceedings of the 4th international workshop, 18-24.

[23] Li, Y., and Lan, Z. 2006. Exploit failure prediction for adaptive fault tolerance in cluster. In Proceedings of the sixth IEEE International symposium on cluster computing and the grid, Vol 1, 531-538.

[24] Chauhan P., and Nitin 2012. Decentralized Computation and Communication Intensive Task Scheduling Algorithm for P2P Grid. In Proceedings of 14th International Conference on Computer Modelling and Simulation (UKSim), 516-521.

[25] Chauhan, P., and Nitin 2012. Resource Based Optimized Decentralized Grid Scheduling Algorithm. Advances in Computer Science, Engineering & Applications Advances in Intelligent and Soft Computing, Volume 167, pp. 1051-1060.

[26] Agarwal, N., Gupta, C., Khare, A. 2012. Task scheduling through limited duplication with processor utilization in grid computing system. In Proceedings of 2nd IEEE International Conference on Parallel Distributed and Grid Computing (PDGC), 921-926.

Author' biography with Photo

Neha Agarwal has done her B.Tech. in Computer Science and Engineering and currently pursuing M.Tech. in Computer Science and Engineering from Jaypee University of Information Technology, Waknaghat, Solan (H.P, India).

Piyush Chauhan has done his B.Tech. in Information Technology and M.Tech. in Computer Science and Engineering. Currently he is pursuing Ph.D. in Computer Science and Engineering from Jaypee University of Information Technology, Waknaghat, Solan (H.P, India).

International Journal of Computers & Technology
Volume 4 No. 3, March-April, 2013, ISSN 2277-3061

Dr.Nitin is Ex First Tier Bank Professor, University of Nebraska at Omaha, NE, USA. His permanent affiliation is with Jaypee University of Information Technology (JUIT), Waknaghat, Solan-173234, Himachal Pradesh, INDIA as a Associate Professor in the Department of Computer Science & Engineering and Information & Communication Technology. He joined Jaypee University of Information Technology in July 2003. He was born on October 06, 1978, in New Delhi, INDIA.
In July 2001, he received the B.Engg. in Computer Science & Engineering [Hons.] and M.Engg. in Software Engineering from Thapar Institute of Engineering and Technology, Patiala, Punjab, INDIA in March 2003. In 2008, he received his Ph.D. in Computer Science & Engineering from JUIT, INDIA. He has completed his Ph.D. course work from University of Florida, Gainesville, FL, USA.
He is a IBM certified engineer. He is a Life Member of IAENG, Senior Member IACSIT and Member of SIAM,IEEE and ACIS and has 125 research papers in peer reviewed International Journals & Transactions, Book Chapters, Symposium, Conferences and Position. His research interest includes Social Networks especially Computer Mediated Communications & Flaming, Interconnection Networks & Architecture, Fault-tolerance & Reliability, Networks-on-Chip, Systems-on-Chip, and Networks-in-Packages, Application of Stable Matching Problems, Stochastic Communication and Sensor Networks. Currently he is working on Parallel Simulation tools, BigSim using Charm++, NS-2 using TCL. He is referee for the Journal of Parallel and Distributed Computing, Elsevier Sciences, Computer Communications, Elsevier Sciences, Computers and Electrical Engineering, Elsevier Sciences, Mathematical and Computer Modelling, Elsevier Sciences. WSEAS Transactions, The Journal of Supercomputing, Springer and International Journal of System Science, Taylor & Francis.