Hindawi Publishing Corporation
Journal of Engineering
Volume 2013, Article ID 749132, 13 pages
http://dx.doi.org/10.1155/2013/749132

Research Article

Fault Tolerant PLBGSA: Precedence Level Based Genetic Scheduling Algorithm for P2P Grid

Piyush Chauhan and Nitin

Department of Computer Science & Engineering and Information & Communication Technology, Jaypee University of Information Technology, Waknaghat, Solan 173234, India

Correspondence should be addressed to Nitin; delnitin@ieee.org

Received 1 April 2013; Revised 30 September 2013; Accepted 21 October 2013

Academic Editor: Paolo Colantonio

Due to monetary limitation, small organizations cannot afford high end supercomputers to solve highly complex tasks. P2P (peer to peer) grid computing is being used nowadays to break complex task into subtasks in order to solve them on different grid resources. Workflows are used to represent these complex tasks. Finishing such complex task in a P2P grid requires scheduling subtasks of workflow in an optimized manner. Several factors play their part in scheduling decisions. The genetic algorithm is very useful in scheduling DAG (directed acyclic graph) based task. Benefit of a genetic algorithm is that it takes into consideration multiple criteria while scheduling. In this paper, we have proposed a precedence level based genetic algorithm (PLBGSA), which yields schedules for workflows in a decentralized fashion. PLBGSA is compared with existing genetic algorithm based scheduling techniques. Fault tolerance is a desirable trait of a P2P grid scheduling algorithm due to the untrustworthy nature of grid resources. PLBGSA handles faults efficiently.

1. Introduction

As the complexity of computational problems is increasing continuously, efficient use of computational resources becomes vital. Complex tasks are causing bottlenecks in performance throughout the technical arena. Organizations around the world use high-end computational devices, servers, and supercomputers to handle complex tasks. However, all organizations are not able to purchase such devices because of budget constraints. Grid computing has come up as a crusader to solve a highly complex task [1, 2]. Grid utilizes existing heterogeneous computational devices spread across multiple geographical locations [3]. This unification of computational resources yields manifold increase in computational capabilities. Initially central scheduler based scheduling algorithms were used by researchers to solve complex problems [4]. These techniques were effective in scheduling [5] complex task; however, they have many limitations, like the fact that failure of central scheduler causes collapse of the entire grid [6]. Limited capabilities of the central scheduler give way to scalability issues. Policies vary from company to company and political issues also caused the existence of central scheduler problematic [6].

Metascheduler deals with limitations of central scheduler to some extent [7]. In metascheduling, all clusters have their personal scheduler. DAG based tasks [8] are scheduled over the most capable cluster. Problem with global task scheduling arises when no cluster is capable of executing complex computational task. The drawback of metascheduler is that it cannot execute gigantic tasks using miniscule clusters and single computational resources, spread across various geographical domains. P2P technologies [9] are effective enough to act as decentralized grid scheduler. Decentralization makes our grid robust against grid node failures. Moreover, structure of P2P grid does not cause scalability issues and other bottlenecks. Further, complex problems are solved efficiently using genetic algorithm. P2P [10] grid also uses genetic algorithm to obtain good results [11]. Initially, to get results quickly, researchers schedule independent gigantic tasks [11] which are generated on single grid node over P2P grid resources [12]. Parallel execution of such tasks over various P2P resources produced results quickly. DAG

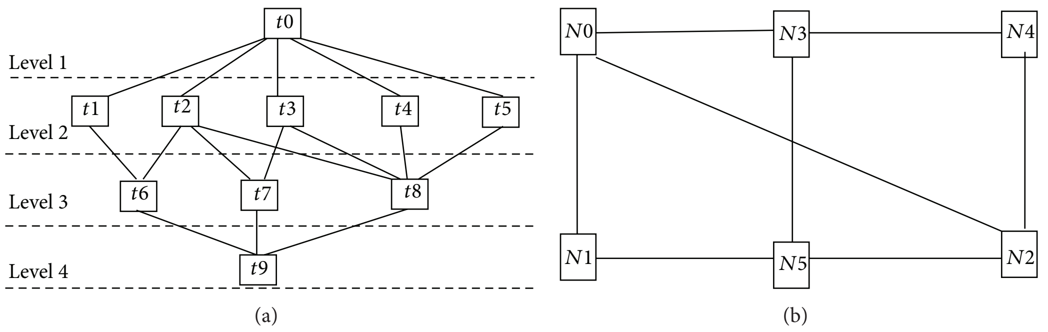

FIGURE 1: (a) DAG based sample task t; (b) virtual network topology.

based tasks [13] require extra precision in scheduling when scheduled over grid. Intertask dependencies makes it tough to schedule subtasks of DAG based task, as efficiently as independent tasks. Researchers in [14] have used genetic algorithm to schedule subtasks of DAG based task [15]. Authors of [14] have applied a genetic algorithm to find schedule for DAG based tasks in one go. The probability of finding nearly optimal results decreases as tasks of DAG are divided across various precedence levels.

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.

Rest of paper is organized as follows literature review is given in Section 2. Background of genetic algorithm and DAG based task scheduling using it is explained in Section 2. We have proposed fault tolerant precedence level based genetic scheduling algorithm for P2P Grid in Section 3. In Section 4, we have represented and discussed simulation results. Conclusion and future scope of work are discussed in Section 5. The symbols which are used throughout the paper are presented in the Abbreviations.

2. Literature Review of Decentralized Scheduling Techniques Using Genetic Algorithm to Schedule Tasks

Holland first explained genetic algorithm in 1975. In the last decade, genetic algorithms have been used by various researchers to schedule tasks over grid. Both independent and interdependent tasks were scheduled using genetic algorithms. Estimation of distribution algorithm (EDAs) is a new class of evolutionary algorithms. In EDAs, promising schedules are obtained by means of probabilistic model. EDAs give better schedules as compared to the evolutionary algorithms mentioned in [16]. In Section 2.1, we have described some remarkable decentralized scheduling techniques using genetic algorithm to schedule independent tasks. Section 2.2 highlights decentralized scheduling algorithm employing genetic algorithm to schedule subtasks of workflow.

2.1. Decentralized Scheduling Techniques Using Genetic Algorithm to Schedule Independent Tasks. One of the eminent papers which applied genetic algorithm to schedule independent tasks over heterogeneous resources in fully decentralized fashion is given in [17]. Scheduling applications using genetic algorithms (SAGA) were proposed in [17]. In the SAGA, computational nodes can connect and leave system dynamically. This system utilizes lookup services to work as decentralized scheduler. The SAGA emphasizes on splitting the task sets and heterogeneous resources into subparts. Moreover, algorithm is run on each subpart. In the SAGA, firstly scheduling request is put forward by user. By using grid monitoring service (MonALISA) [18], we obtain monitoring data and scheduling request. After this, near optimal schedule is obtained by using monitoring data and scheduling request. Execution services then execute this near optimal schedule. Discovery service is provided in the SAGA to handle failure and incorporate new computational resources. Schedule and task information of executed jobs are provided as feedback to the user. This algorithm decreases the number of generations required in a genetic algorithm to yield efficient schedule.

Decentralized grid scheduling is achievable using P2P technique and it was proved in [11]. In [11], after authorization and authentication checks, any grid node can issue a job submission query. Cyclone is recursively accessed to find $\beta \times N$ nodes. β is a parameter of the algorithm. N Nodes which decrease optimization function give the first schedule. It is impossible to investigate all possible permutations of N nodes out of total $\beta \times N$. Only in two cases (N is exceptionally petite or $\beta \approx 1$), we can investigate all permutations. Thus, genetic algorithm is used for the selection process to obtain nearly optimal schedules. A limitation of this algorithm is that it can only schedule independent tasks like SAGA.

2.2. Decentralized Scheduling Techniques Using Genetic Algorithm to Schedule Subtasks of DAG Based Task. In grid environment, genetic algorithm [19] was used to schedule [20] DAG based task in [14]. We know that scheduling subtasks of DAG based task over a grid is an NP hard problem. Genetic algorithm and other stochastic search algorithms are utilized to obtain near optimal schedule for DAG based task's scheduling on grid nodes.

DAG based workflow chosen by us to put into operation [14] is shown in Figure 1(a). DAG based task t is divided into 10 subtasks. These subtasks are further subdivided into four

Table 1: Fault tolerant PLBGSA for P2P grid.

Begin	Cost	Time
(1) Construct priority based subtask sequence β	c_1	1
(2) Assign level $l = 0$	c_2	1
(3) **Do**	c_3	$n+1$
(4) Choose all j subtasks present at level l from β	c_4	$n+1$
(5) **if** Ntl == 1	c_5	$n+1$
(6) Single subtask ST_S^l assigned to node having minimum $\text{Wld}_N^{\text{ST}_S}$ where $N \in [0, P-1]$	c_6	$n+1$
(7) **Endif**		
(8) **if** Ntl > 1	c_7	$n+1$
(9) **for** $i = 0$ to h	c_8	$(n+1)(n+1)$
(10) Using RnG assign all ST_j^l to set P	c_9	$n(n+1)$
(11) Calculate Ftm_ST_j^l	c_{10}	$n(n+1)$
(12) Finish time value of $\text{Sd}_{ji}^l = \text{Ftm\_ST}_j^l$	c_{11}	$n(n+1)$
(13) **Endfor**		
(14) GNo = 2	c_{12}	$n+1$
(15) **for** $i = h$ to $(u \times h) - 1$	c_{13}	$(n+1)(k \cdot n+1)$
(16) **if** $i\%h == r$ where $r \in [0, k-1]$	c_{14}	$k \cdot n(n+1)$
(17) Using RWS choose Sd_{ji}^l schedule from Rng_Sd where $\text{Rng\_Sd} \in [(i-h+r), (h-r+1)]$	c_{15}	$(n+1)(k \cdot n-1)$
(18) Calculate Ftm_ST_j^l	c_{16}	$(n+1)(k \cdot n-1)$
(19) **Endif**		
(20) **if** $i\%h == e$ where $e \in [k, (\text{GNo} \times h) - 1]$	c_{17}	$(k \cdot n)(n+1)$
(21) Shortlist $\text{Sd}_{1\text{st}}^l$ and $\text{Sd}_{2\text{nd}}^l$ from Rng_Sd where $\text{Rng\_Sd} \in [((\text{GNo} \times h) - h), ((\text{GNo} \times h) - h + k)]$	c_{18}	$(n+1)(k \cdot n-1)$
(22) using CxGO (occasional MtGO) on $\text{Sd}_{1\text{st}}^l$ and $\text{Sd}_{2\text{nd}}^l$ we obtain Sd_{ji}^l	c_{19}	$(n+1)(k \cdot n-1)$
(23) Calculate Ftm_ST_j^l	c_{20}	$(n+1)(k \cdot n-1)$
(24) **if** $e == (\text{GNo} \times h) - 1$	c_{21}	$(n+1)(k \cdot n-1)$
(25) GNo + +	c_{22}	$(n+1)(k \cdot n-2)$
(26) **Endif**		
(27) **if** $i == (u \times h) - 1$	c_{23}	$(n+1)(k \cdot n-1)$
(28) Shortlist $\text{Sd}_{\text{smallest}}^l$ from Rng_Sd where $\text{Rng\_Sd} \in [0, ((u \times h) - 1)]$	c_{24}	$(n+1)(k \cdot n-2)$
(29) **for** $f = h$ to $(u \times h) - 1$	c_{25}	$(k \cdot n-2)(k \cdot n+1)(n+1)$
(30) $\text{Sd}_{jf}^l = \text{Sd}_{\text{smallest}}^l$	c_{26}	$(n+1)(k \cdot n-2)(k \cdot n)$
(31) **Endfor**		
(32) **Endif**		
(33) **Endif**		
(34) **Endfor**		
(35) **Endif**		
(36) **if** NdF == 1	c_{27}	$(n+1)(n+1)$
(37) Remove failed node from set P	c_{28}	$n(n+1)$
(38) Goto step 3	c_{29}	$n(n+1)$
(39) **End if**		
(40) l + +	c_{30}	$(n+1)(n+1)$
(41) **while** $l <=$ Max Level	c_{31}	$(n+1)(n+1)$
(42) **End do While**		
(43) Finish time for $\text{Sd}_{\text{smallest}}^l$ at level l = Max Level will be the finish time of task t	c_{32}	1
END		

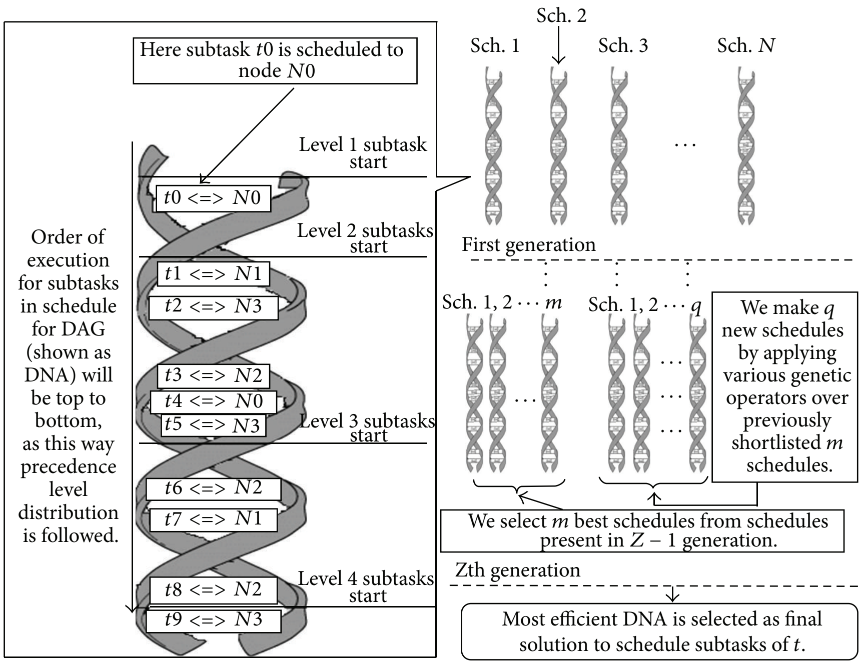
Here subtask $t0$ is scheduled to node $N0$
Order of execution for subtasks in schedule for DAG (shown as DNA) will be top to bottom, as this way precedence level distribution is followed.
Level 1 subtask start
$t0$ <=> $N0$
Level 2 subtasks start
$t1$ <=> $N1$
$t2$ <=> $N3$
$t3$ <=> $N2$
$t4$ <=> $N0$
$t5$ <=> $N3$
Level 3 subtasks start
$t6$ <=> $N2$
$t7$ <=> $N1$
Level 4 subtasks start
$t8$ <=> $N2$
$t9$ <=> $N3$
Sch. 1
Sch. 2
Sch. 3
Sch. N
First generation
Sch. 1, 2 ⋯ m
Sch. 1, 2 ⋯ q
We make q new schedules by applying various genetic operators over previously shortlisted m schedules.
We select m best schedules from schedules present in $Z - 1$ generation.
Zth generation
Most efficient DNA is selected as final solution to schedule subtasks of t.

Figure 2: DNA representing scheduling for subtasks of task t and use of genetic algorithm to obtain good schedule.

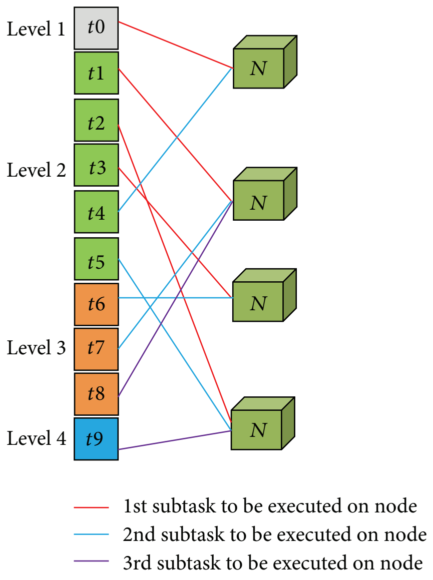
Level 1
Level 2
Level 3
Level 4
$t0$
$t1$
$t2$
$t3$
$t4$
$t5$
$t6$
$t7$
$t8$
$t9$
N
N
N
N
1st subtask to be executed on node
2nd subtask to be executed on node
3rd subtask to be executed on node

Figure 3: The sequence in which subtasks of t, assigned to the same node, will be executed.

precedence levels. Once all subtasks of previous precedence level have returned results to origin node, subtasks at next precedence level start executing in parallel. Subtasks present at the same precedence level are executed in parallel on the separate grid nodes. Virtual network topology followed to be simulated [14] by us is shown in Figure 1(b). DAG based task t is generated at origin node ($N0$). We used genetic algorithm to schedule subtasks of t over grid nodes $N0$, $N1$, $N2$, and $N3$.

Figure 2 represents how [14] used genetic algorithm to schedule subtasks of DAG based task t. Initial population of schedules is produced in [14] by arbitrarily assigning every subtask to a grid node. Offspring for the next generation are chosen from an initial population using roulette wheel selection method. Authors have applied genetic operators on these shortlisted schedules to obtain the rest of population for given generation. Genetic operators used are crossover and mutation. When stagnation in population arises, then the probability of mutation increases. From population, the best DNA representing schedule for subtasks of t is chosen.

We know subtasks of t are divided into precedence levels. Subtasks at the same precedence level can be executed in parallel on different grid nodes. A prerequisite for subtasks at level l to start execution is that subtasks at level $l - 1$ have finished and returned results. This sequence is followed in DNA representation of the schedule. Subtask at the first level is the first subtask at the top of DNA. If more than one subtask is assigned to the node, then the subtask which comes first in DNA will be executed first. Figure 3 explains the sequence in which subtasks of t assigned to same node will be executed. In Figure 3, subtasks at the same precedence level are having the same color. Subtasks of level 3 are represented by orange color. Level 3 subtasks will start executing once subtasks at level 2 have finished and delivered results. Level 2 tasks are represented by green boxes and will execute in parallel on different nodes. In case grid node leaves the grid, we have to reschedule all subtasks of t again using [14].

In P2P grid, nodes can leave freely. Hence, our algorithm exploits precedence level approach to handle node failures.

3. Fault Tolerant Genetic Algorithm Based Decentralized Scheduling Technique for P2P Grid

In this paper, we have proposed a decentralized scheduling technique for P2P grid, which utilizes a precedence level

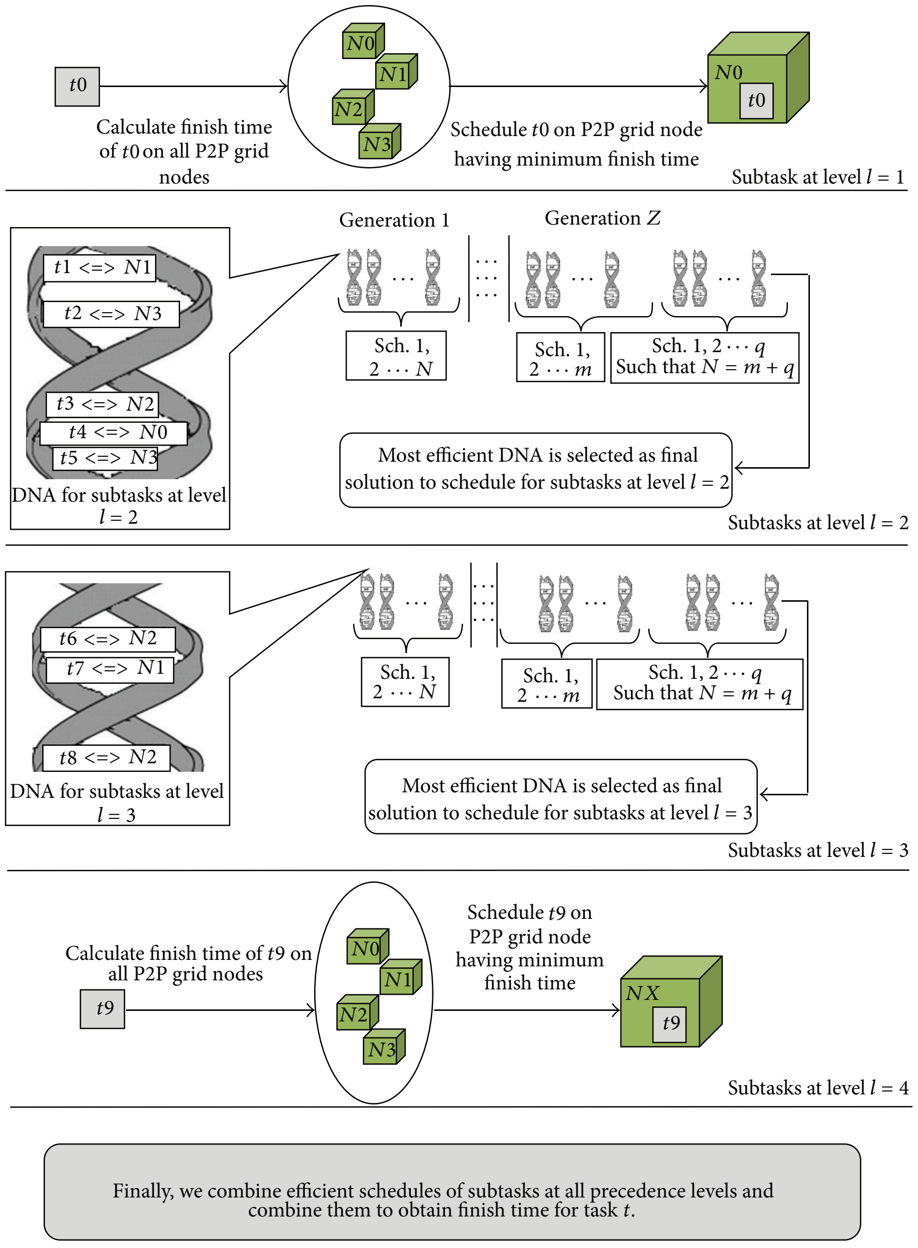
t0
Calculate finish time of t0 on all P2P grid nodes
N0
N1
N2
N3
Schedule t0 on P2P grid node having minimum finish time
N0
t0
Subtask at level l = 1
t1 <=> N1
t2 <=> N3
t3 <=> N2
t4 <=> N0
t5 <=> N3
DNA for subtasks at level l = 2
Generation 1
Generation Z
Sch. 1, 2 ⋯ N
Sch. 1, 2 ⋯ m
Sch. 1, 2 ⋯ q Such that N = m + q
Most efficient DNA is selected as final solution to schedule for subtasks at level l = 2
Subtasks at level l = 2
t6 <=> N2
t7 <=> N1
t8 <=> N2
DNA for subtasks at level l = 3
Sch. 1, 2 ⋯ N
Sch. 1, 2 ⋯ m
Sch. 1, 2 ⋯ q Such that N = m + q
Most efficient DNA is selected as final solution to schedule for subtasks at level l = 3
Subtasks at level l = 3
Calculate finish time of t9 on all P2P grid nodes
t9
N0
N1
N2
N3
Schedule t9 on P2P grid node having minimum finish time
NX
t9
Subtasks at level l = 4
Finally, we combine efficient schedules of subtasks at all precedence levels and combine them to obtain finish time for task t.

Figure 4: Diagrammatic presentation of PLBGSA.

based genetic algorithm (PLBGSA) to schedule subtasks of DAG based task t. In DAG based task, subtasks have intertask dependencies. In addition to schedule subtasks over various grid nodes, we have to find out the associated computation and communication cost [21]. We store in all P2P nodes a list α mentioned in [22]. This list α is modified whenever scheduling happens. Accordingly, neighbors also modify their list α. In [23], authors put forward the concept of workload (computing field) of subtask over any P2P grid resource. It is given in (1) as follows:

$$\mathrm{CF} = \frac{\sum_{a=1}^{b} T_a}{\mathrm{Pr} \times \mathrm{MIPS}_{\mathrm{Pr}}}, \qquad (1)$$

where Pr represents the total number of processing elements present in P2P grid node. $\mathrm{MIPS}_{\mathrm{Pr}}$ the gives number of million instructions per second single processing element can process. T_a is size in million instructions of ath waiting subtask in the task queue of b length on grid node. Communication cost [22] is the time to send subtask from one node to another, explained by (2) as follows:

$$\mathrm{Trt}_{XN}^{\mathrm{ST}_{ji}^{l}} = \left(\frac{T_a^{\mathrm{Kb}}}{\mathrm{Wsz}_{xy}}\right) \times \mathrm{RT}_{xy}. \qquad (2)$$

In the above equation, Wsz_{xy} is window size and RT_{xy} is round trip time between nodes x and y. Size of subtask a in Kb is represented by T_a^{Kb}. Subtask will also depend upon time consumed to finish subtasks at a previous precedence level and to return results to the origin node. Previous algorithms, have used DNA containing details of all subtasks of DAG based task t. Genetic algorithm was applied using initial population of randomly generated DNAs. Task t shown in Figure 1(a) is divided into precedence levels. In our proposed approach, when any precedence levels contain only one subtask, we need not to apply genetic algorithm on that particular subtask. Instead, we calculate finish time of subtask on all available P2P grid nodes. Finally, we schedule single subtask on the node which gives fastest result. As shown in Figure 4, single subtask $t0$ is scheduled without using genetic algorithm. This scheduling value for $t0$ is stored in list α. This value will be taken as prerequisite to schedule subtasks at the next level.

On the other hand, if precedence level of subtask contains more than one subtask, we use a genetic algorithm to find good schedule. As shown in Figure 4, at precedence level 2, five subtasks are present. Schedule to finish these 5 subtasks is represented by DNA. N such DNAs are randomly generated for initial generation. The Z such generations are generated by applying genetic operators on shortlisted DNAs of previous generation. Crossover and mutation are genetic operators used in this paper. The roulette wheel selection technique is used to shortlist DNAs from all DNAs present at any generation. We select DNA from all these generations such that it finishes subtasks the fastest. We schedule using the best schedule among all generations. Values are stored accordingly in list α. Again, we apply genetic algorithm for subtasks at level 3 and find good schedule. Scheduling is performed according to this good schedule and list α will be updated. Again, there is a single task $t9$ in level 4, just like in level 1. Hence, $t9$ is scheduled on node giving results fastest. This way all subtasks of DAG based task t are scheduled.

Algorithm for PLBGSA is shown in Figure 5. In this algorithm, first we arrange all subtasks in priority based task sequence β. Assign level l value 1. Choose all j subtasks present at level l from β. If only one subtask is present at level l, then single subtask ST_S^l is assigned to a node having minimum $\mathrm{Wld}_N^{\mathrm{ST}_S}$. $\mathrm{Wld}_N^{\mathrm{ST}_S}$ is workload after subtask ST_S^l is assigned to node N. Range of N is 0 to $P-1$, where P represents the number of P2P grid nodes available for scheduling. If more than one subtask is present at level l, we use genetic algorithm to schedule all subtasks on set P. While applying genetic algorithm, first we generate an initial population of N DNAs. To obtain single DNA we use RnG and assign all ST_j^l to set $P \cdot \mathrm{ST}_j^l$ represents a set of j subtasks present at level l from β. Then we calculate $\mathrm{Ftm\_ST}_j^l$ for DNA. Finish time value of Sd_{ji}^l is made equal to $\mathrm{Ftm\_ST}_j^l$. Further, generations are obtained by applying genetic operators on the previous generation. $\mathrm{Ftm\_ST}_j^l$ will be calculated by scheduling subtasks at level 2 one by one in sequence in which they are found in β. Consider

$$\mathrm{Ftm\_ST}_j^l = \min_{i\in[0,j]} \left\{ \mathrm{CF}_c^{\mathrm{ST}_{ji}^l} \right\}. \tag{3}$$

When we schedule subtask, workload of the node on which subtask is scheduled will also vary. This new workload will be as follows:

$$\begin{gathered} \mathrm{CF}_c^{\mathrm{ST}_{ji}^l} = \mathrm{MEF} + \mathrm{tld}_C^{\mathrm{ST}_{ji}^l}, \\ \mathrm{MEF} = \max\left\{ \mathrm{CF}_c^{\mathrm{old}}, \mathrm{Trt}_{XN}^{\mathrm{ST}_{ji}^l}, \mathrm{Rbk}^{l-1} \right\}. \end{gathered} \tag{4}$$

Here, MEF is the most efficient factor and will be the greatest of these three values. First value is the old workload on P2P grid node $\mathrm{CF}_c^{\mathrm{old}}$. Second is transport time $\mathrm{Trt}_{XN}^{\mathrm{ST}_{ji}^l}$ to send a task from one node to another. Rbk^{l-1} is the third value which gives time when all subtasks at previous level will be finished and had returned results. An origin node where task t is generated will use these values to make a scheduling decision. However, entities in list α are changed when we have found the best schedule using genetic algorithm. We shortlist m schedules from an initial population by applying roulette wheel selection method (RWS). This way second generation's first m schedules will be obtained from predecessors. We choose two schedules $\mathrm{Sd}_{1\mathrm{st}}^l$, $\mathrm{Sd}_{2\mathrm{nd}}^l$ from these m schedules and apply CxGO, MtGO genetic operators. Two new schedules will be obtained by this method. In this manner q new schedules for second generation are obtained. MtGO is a mutation operator which will be applied more often if stagnation in schedules occurs. We shortlist $\mathrm{Sd}_{\mathrm{smallest}}^l$ from Rng_Sd and schedule according to $\mathrm{Sd}_{\mathrm{smallest}}^l$. Update list α according to $\mathrm{Sd}_{\mathrm{smallest}}^l$. Similarly, we calculate $\mathrm{Sd}_{\mathrm{smallest}}^l$ for all levels and update list α accordingly for all levels. Finally, at level l having value Max Level, $\mathrm{Sd}_{\mathrm{smallest}}^l$ represents finish time for task t. The schedule obtained using this algorithm is better than the algorithm presented in [14]. Our proposed algorithm is depicted in Table 1.

In Table 1, cost is the statement that takes c_i steps to execute and c_i step executes n times. Hence, we find that, in the worst case, the running time of the above scheduling algorithm is

$$\begin{aligned} T(n) = {} & c_1 \cdot 1 + c_2 \cdot 1 + c_3 \cdot (n+1) + c_4 \cdot (n+1) \\ & + c_5 \cdot (n+1) + c_6 \cdot (n+1) + c_7 \cdot (n+1) \\ & + c_8 \cdot [(n+1)(n+1)] + c_9 \cdot [n \cdot (n+1)] \\ & + c_{10} \cdot [n \cdot (n+1)] + c_{11} \cdot [n \cdot (n+1)] \\ & + c_{12} \cdot (n+1) + c_{13} \cdot [(k \cdot n + 1)(n+1)] \\ & + c_{14} \cdot [k \cdot n (n+1)] + c_{15} [(k \cdot n - 1)(n+1)] \\ & + c_{16} [(k \cdot n - 1)(n+1)] + c_{17} \cdot [k \cdot n (n+1)] \\ & + c_{18} [(k \cdot n - 1)(n+1)] + c_{19} \cdot [(k \cdot n - 1)(n+1)] \\ & + c_{20} \cdot [(k \cdot n - 1)(n+1)] + c_{21} \cdot [(k \cdot n - 1)(n+1)] \\ & + c_{22} \cdot [(k \cdot n - 2)(n+1)] + c_{23} \cdot [(k \cdot n - 1)(n+1)] \\ & + c_{24} \cdot [(k \cdot n - 2)(n+1)] \\ & + c_{25} \cdot [(k \cdot n - 2)(k \cdot n + 1)(n+1)] \end{aligned}$$

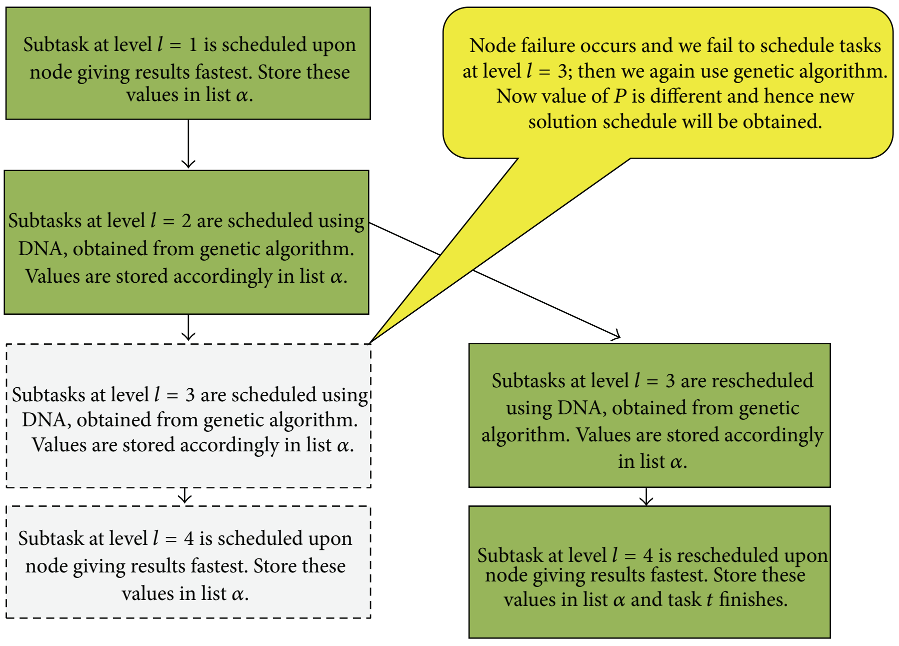
Subtask at level $l = 1$ is scheduled upon node giving results fastest. Store these values in list α.
Subtasks at level $l = 2$ are scheduled using DNA, obtained from genetic algorithm. Values are stored accordingly in list α.
Subtasks at level $l = 3$ are scheduled using DNA, obtained from genetic algorithm. Values are stored accordingly in list α.
Subtask at level $l = 4$ is scheduled upon node giving results fastest. Store these values in list α.
Node failure occurs and we fail to schedule tasks at level $l = 3$; then we again use genetic algorithm. Now value of P is different and hence new solution schedule will be obtained.
Subtasks at level $l = 3$ are rescheduled using DNA, obtained from genetic algorithm. Values are stored accordingly in list α.
Subtask at level $l = 4$ is rescheduled upon node giving results fastest. Store these values in list α and task t finishes.

FIGURE 5: Working of fault tolerance in PLBGSA.

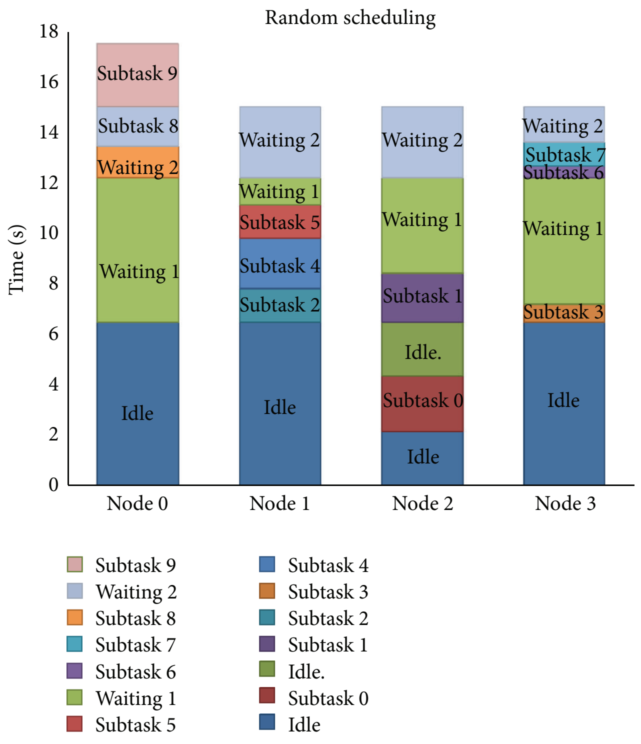

FIGURE 6: Detailed timewise schedule for all subtasks of DAG based task t using random scheduling.

$$+ c_{26} \cdot [(k \cdot n - 2) \times k \cdot n (n + 1)]$$
$$+ c_{27} \cdot [(n + 1)(n + 1)]$$
$$+ c_{28} \cdot [n (n + 1)] + c_{29} \cdot [n (n + 1)]$$
$$+ c_{30} \cdot [(n + 1)(n + 1)] + c_{31} \cdot [(n + 1)(n + 1)] \cdot$$
$$+ c_{32} \cdot 1, \tag{5}$$

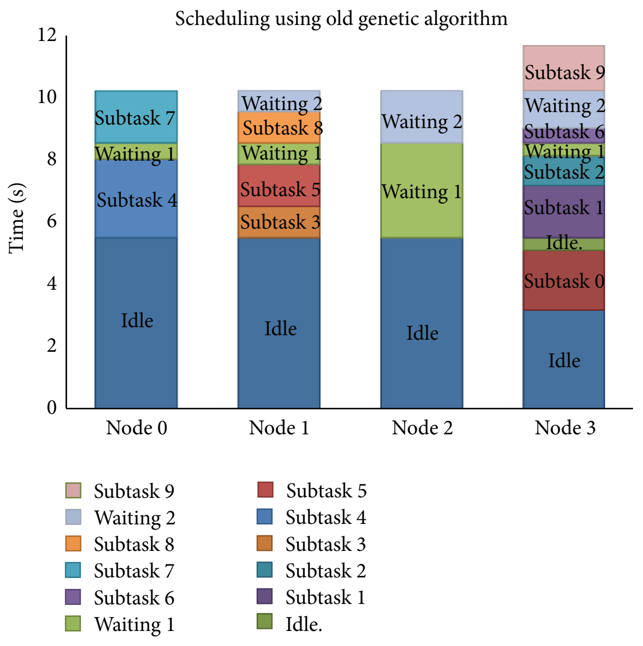

FIGURE 7: Detailed timewise schedule for all subtasks of DAG based task t using genetic algorithm for DAG scheduling in grid.

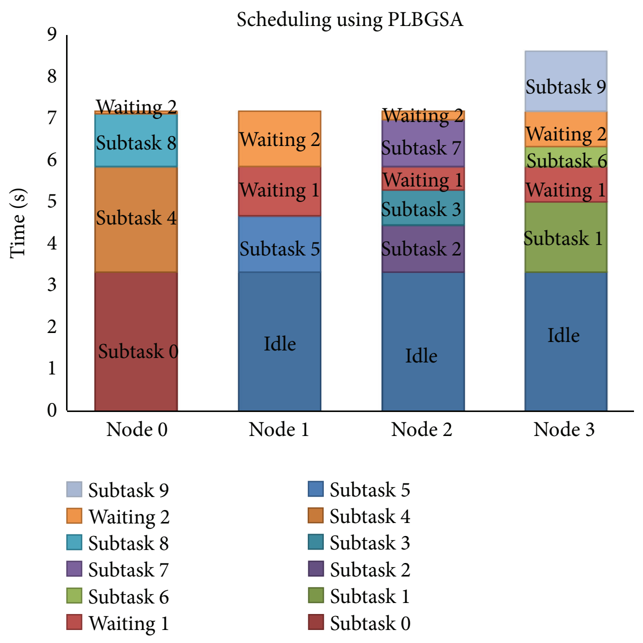

FIGURE 8: Detailed timewise schedule for all subtasks of DAG based task t using PLBGSA.

which on simplifying gives

$$\begin{aligned}T(n) = {}& \left(c_{25}\cdot k^2 + c_{26}\cdot k^2\right)n^3 \\ & + \left(c_8 + c_9 + c_{10} + c_{11} + c_{13}\cdot k + c_{14}\cdot k + c_{15}\cdot k\right. \\ & \quad + c_{16}\cdot k + c_{17}\cdot k + c_{18}\cdot k + c_{19}\cdot k + c_{20}\cdot k \\ & \quad + c_{21}\cdot k + c_{22}\cdot k + c_{23}\cdot k + c_{24}\cdot k \\ & \quad + c_{25}\cdot\left(k^2 - k\right) + c_{26}\cdot\left(k^2 - 2k\right) + c_{27} \\ & \quad \left. + c_{28} + c_{29} + c_{30} + c_{31}\right)n^2 \\ & + \left(c_3 + c_4 + c_5 + c_6 + c_7 + 2\cdot c_8 + c_9 + c_{10} + c_{11}\right. \\ & \quad + c_{12} + c_{13}\cdot(k+1) + c_{14}\cdot k + c_{15}\cdot(k-1) \\ & \quad + c_{16}\cdot(k-1) + c_{17}\cdot k + c_{18}\cdot(k-1) + c_{19}\cdot(k-1) \\ & \quad + c_{20}\cdot(k-1) + c_{21}\cdot(k-1) + c_{22}\cdot(k-2)\end{aligned}$$

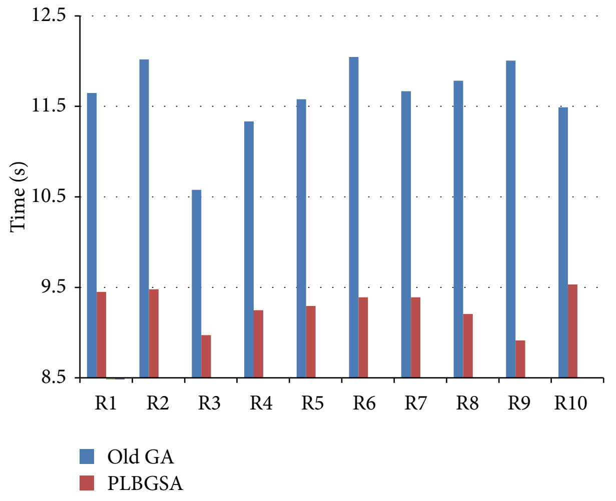

FIGURE 9: Ten runs of old GA and PLBGSA, with u generations.

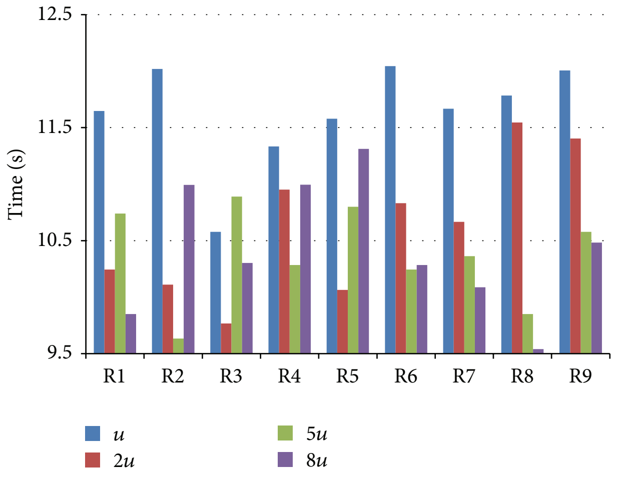

FIGURE 10: Ten runs of old GA algorithm with $u, 2u, 5u,$ and $8u$ generations.

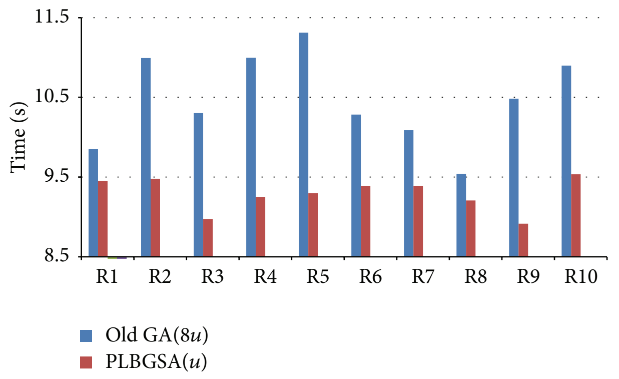

FIGURE 11: Comparison of 10 runs of old GA algorithm with $8u$ generations and PLBGSA with u generations.

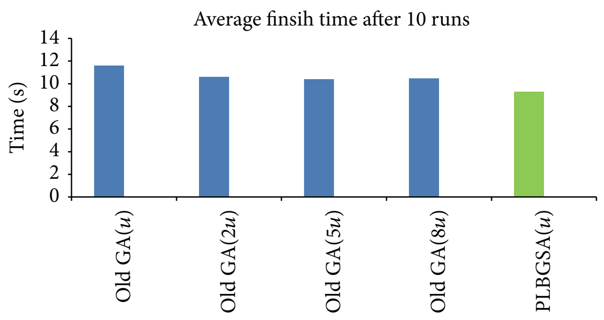

FIGURE 12: Comparison of average finish time for task t using old GA with u, $2u$, $5u$, and $8u$ generations and PLBGSA with u generations.

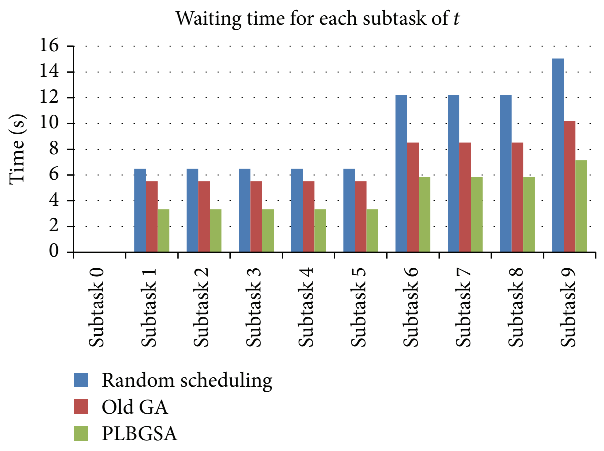

FIGURE 13: Waiting time for each subtask of t with random scheduling, old GA, and PLBGSA.

$$\begin{aligned}&\quad + c_{23}\cdot(k-1) + c_{24}\cdot(k-2) - c_{25}\cdot(k+2)\\&\quad -c_{26}\cdot 2k + c_{27}\cdot 2 + c_{28} + c_{29} + c_{30}\cdot 2 + c_{31}\cdot 2\big) n\\&+ \big(c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_{12}\\&\quad + c_{13} - c_{15} - c_{16} - c_{18} - c_{19} - c_{20} - c_{21} - 2\cdot c_{22}\\&\quad -c_{23} - 2\cdot c_{24} - 2\cdot c_{25} + c_{27} + c_{30} + c_{31} + c_{32}\big)\cdot 1\\&= O\left(n^3\right).\end{aligned} \tag{6}$$

The running time of the algorithm is the sum of running times for each statement executed. We can express the above equation in the form of an^3+bn^2+cn+d for constants $a, b, c, d,$ and k that again depends on statement costs c_i; it is thus a quadratic function of n, that is, n^3.

The concept of fault tolerance is also introduced in our algorithm. Fault tolerance [24] mechanism used in this approach is the modified version of fault tolerance [25, 26] mechanism of our previous work [22]. Two components present on all P2P grid nodes are notification generator and notification receiver in order to handle failure situation. These components either transfer or receive the three types of messages, the heartbeat message, the task completion

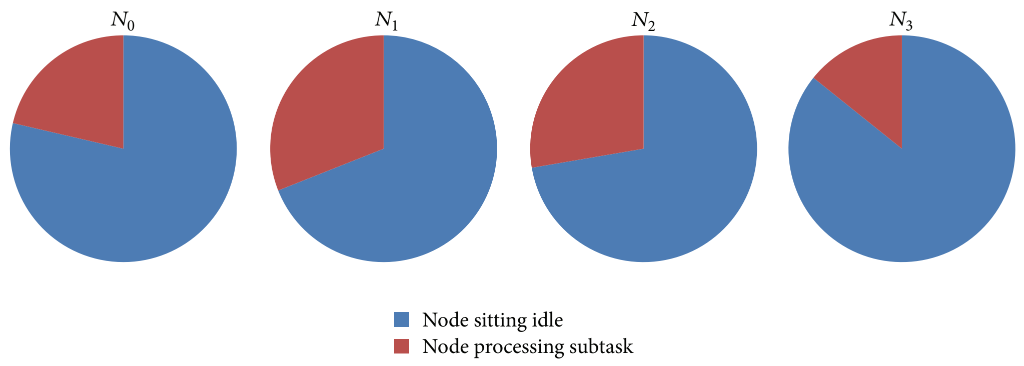

FIGURE 14: Utilization of P2P nodes in random scheduling.

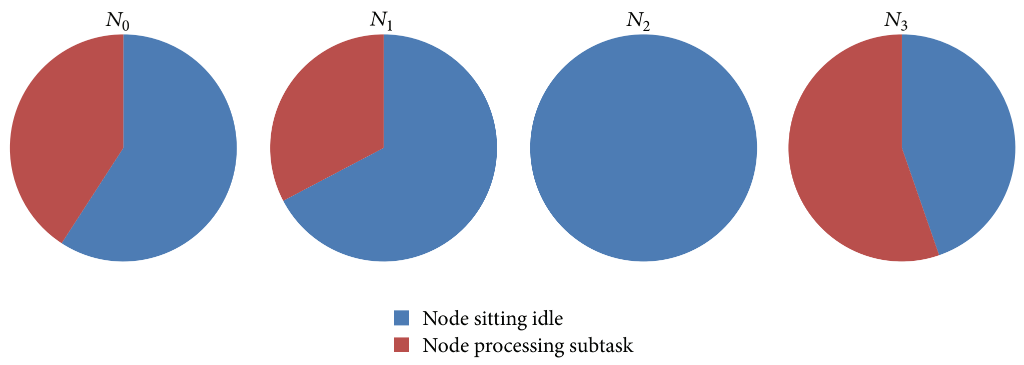

FIGURE 15: Utilization of P2P nodes in old GA.

message, and task failure message. If no message is received and periodic time expires, then an automatic task failure message is generated at the node. When failure message is generated then we reschedule subtasks at level l. Further, levels beyond l are again rescheduled, accordingly.

In this manner, we achieve two goals by applying genetic algorithm separately for each level instead of applying on all levels at once. Firstly, we obtain better schedule. Also, fault tolerance approach will be applicable because of the unreliable nature of P2P grid nodes. Simulation results and discussion are given in the next section in support of PLBGSA algorithm.

4. Simulation Results and Discussion

Virtual network topology followed in this paper is shown in Figure 1(b). $N0$ node is the origin node where DAG based task t is generated. Nodes set $P\{N0, N1, N2, N3\}$ represents P2P grid nodes and the number of cores in P2P grid nodes is $\{2, 3, 4, 3\}$, respectively. Window size for each node is $\{75.0, 100.0, 75.0, 100.0\}$. Round trip time for P2P grid nodes is $\{0.1, 0.4, 0.2, 0.4\}$. Computation capacity in million instructions per second for each P2P grid node is $\{1.2, 1.0, 0.9, 1.4\}$.

DAG based task t consists of 10 subtasks $\{t0, t1, \ldots, t9\}$ and sizes of each subtask in million instructions are $\{8.0, 7.0, 4.0, 3.0, 6.0, 4.0, 2.0, 4.0, 3.0, 6.0\}$, respectively. Size in Kb for each subtask is $\{800, 700, 400, 300, 600, 400, 200, 350, 350, 550\}$. Origin node of all subtasks is the same ($N0$). Subtask level is also visible in Figure 1(a).

Now, when random scheduling is used to schedule subtasks of t, finish time of t comes at 17.551 seconds as shown in Figure 6. Then we use genetic algorithm for scheduling DAG based task t, as mentioned in [14]. Results are obtained for task t in 12.019 seconds. Detailed scheduling of all subtasks is shown in Figure 7. When we use precedence level based genetic algorithm for decentralized scheduling of task t on P2P grid nodes, we get results in 8.973 seconds as shown in Figure 8.

After executing 10 times the proposed algorithm and the algorithm proposed in [14], schedules obtained by our techniques always come better as shown in Figure 9. Here, the number of generations was equal. In Figure 10, the algorithm of [14] is run with $u, 2u, 5u$, and $8u$ generations. However, with the increase in generation, schedules obtained in 10 runs have not much reduced in size. In Figure 11, comparison of schedules obtained by running 10 times algorithm proposed in [14] and our algorithm is shown. Reference [14] is having $8u$ number of generations and our algorithm is run only with u generations. However, schedules of previous algorithm were not as good as compared to our proposed algorithm. This fact is demonstrated in Figure 11.

Figure 12 shows that average finish time of our algorithm with only u generations is much better than the average

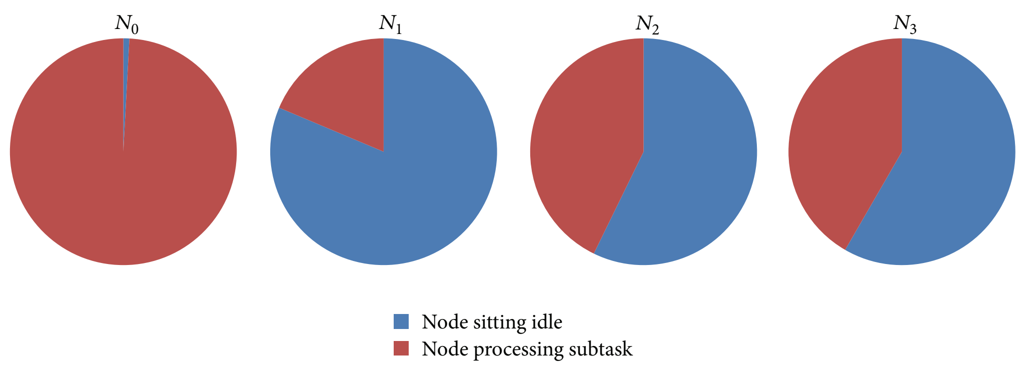

FIGURE 16: Utilization of P2P nodes in PLBGSA.

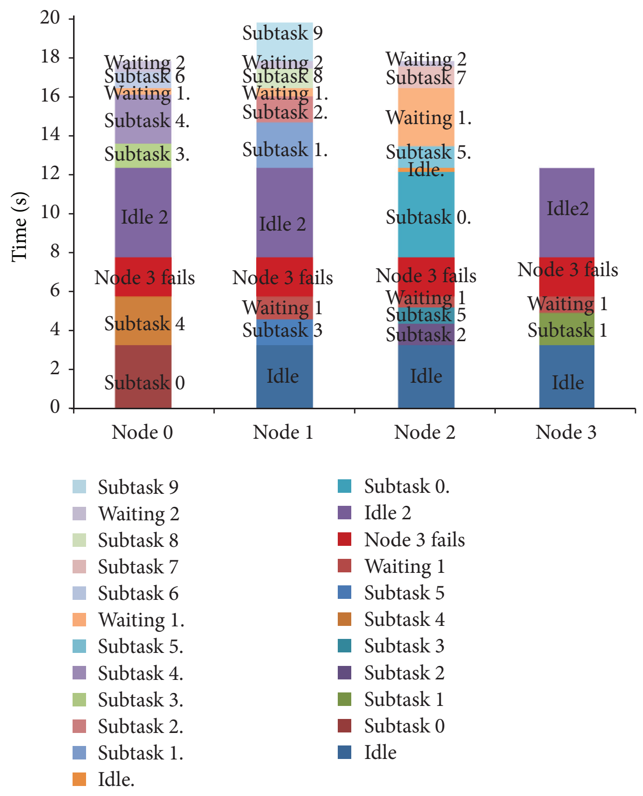

FIGURE 17: Detailed time wise schedule for all subtasks of DAG based task t using old GA when $N3$ fails at level 3 of task t. Here we reschedule using old GA.

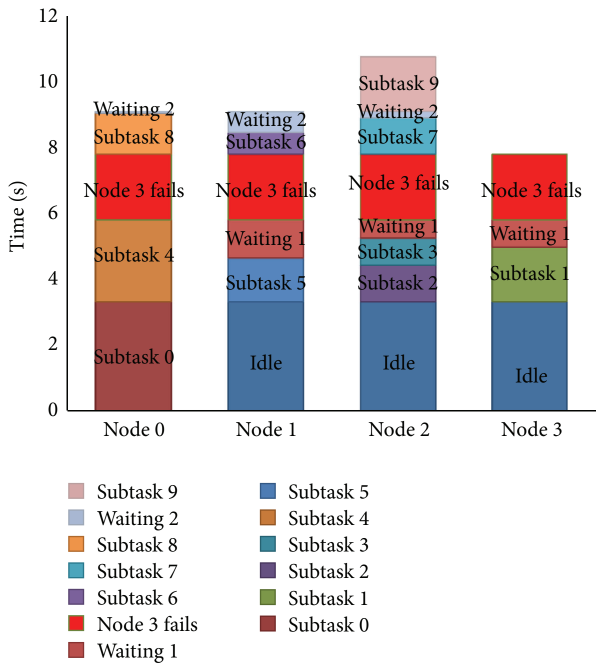

FIGURE 18: Detailed time wise schedule for all subtasks of DAG based task t using PLBGSA when $N3$ fails at level 3 of task t. Here we reschedule using PLBGSA.

finish time of [14], even when the number of generations is increased to $8u$. Also, waiting time for subtasks at all levels is decreased with our algorithm as shown in Figure 13.

Node utilization for random scheduling is shown in Figure 14. When we schedule subtasks using old GA proposed in [14], node 2 is sitting idle throughout all precedence levels. This is shown in Figure 15. Nodes are utilized more uniformly in PLBGSA as shown in Figure 16.

If a distinct P2P grid nodes are to be arranged along p subtasks of task t, where repetition of nodes is allowed, then the total number of ways of doing this scheduling is q; here $q = a^p$. To find out the best schedule among all possible schedules is a very exhaustive task. Hence, using a genetic algorithm we generate g number of schedules out of q possible schedules. Finally, we select the fastest schedule among these g schedules. Now the probability of finding the best schedules in randomly selected g schedules is as follows:

$$\text{Pr.}_{\text{best}}^{q} = \left(\frac{C_{g-1}^{q-1} \times C_1^1}{C_g^q}\right), \quad \text{where } c_{x_2}^{x_1} = \frac{x_1!}{x_2!\,(x_1 - x_2)!}. \tag{7}$$

On solving the right-hand side of (7), we get

$$\text{Pr.}_{\text{best}}^{q} = \left(\frac{g}{q}\right). \tag{8}$$

Our approach first schedules tasks at level 2 then at level 3 and henceforth up to Lth level (second last level) of DAG based task t. Subtasks at level 2 are p_1. Hence the total number of ways of doing scheduling is w_1 (here $w_1 = a^{p_1}$). Now the probability of finding best schedules in randomly selected g schedules is as follows:

$$\mathrm{Pr.}_{\mathrm{best}}^{w_1} = \left(\frac{C_{g-1}^{w_1 - 1} \times C_1^1}{C_g^{w_1}} \right) = \frac{g}{w_1}. \quad (9)$$

Level 3 contains p_2 subtasks; hence, the total number of ways of doing scheduling is w_2 (here $w_2 = a^{p_2}$). Consider

$$\mathrm{Pr.}_{\mathrm{best}}^{w_2} = \left(\frac{C_{g-1}^{w_2 - 1} \times C_1^1}{C_g^{w_2}} \right) = \frac{g}{w_2}. \quad (10)$$

Similarly, at level L,

$$\mathrm{Pr.}_{\mathrm{best}}^{w_{L-1}} = \left(\frac{C_{g-1}^{w_{L-1} - 1} \times C_1^1}{C_g^{w_{L-1}}} \right) = \frac{g}{w_{L-1}}. \quad (11)$$

Now, since $p_1 + p_2 \cdots + p_{L-1} < p$, therefore $w_1 \times w_2 \times \cdots \times w_{L-1} = a^{p_1} \times a^{p_2} \times \cdots \times a^{p_{L-1}} = a^{p_1 + p_2 + \cdots + p_{L-1}} < q$.

From (8)–(11),

$$\begin{aligned} \mathrm{Pr.}_{\mathrm{best}}^{q} &< \mathrm{Pr.}_{\mathrm{best}}^{w_1}, \quad \text{since } (q > w_1), \\ \mathrm{Pr.}_{\mathrm{best}}^{q} &< \mathrm{Pr.}_{\mathrm{best}}^{w_2}, \quad \text{since } (q > w_2). \end{aligned} \quad (12)$$

Similarly,

$$\begin{gathered} \mathrm{Pr.}_{\mathrm{best}}^{q} < \mathrm{Pr.}_{\mathrm{best}}^{w_{L-1}}, \quad \text{since } (q > w_{L-1}) \\ \mathrm{Pr.}_{\mathrm{best}}^{q} < \left(\mathrm{Pr.}_{\mathrm{best}}^{w_1} \times \mathrm{Pr.}_{\mathrm{best}}^{w_2} \times \cdots \times \mathrm{Pr.}_{\mathrm{best}}^{w_{L-1}}\right), \quad (13) \\ \text{since } (q > w_1 \times w_2 \times \cdots \times w_{L-1}). \end{gathered}$$

Hence, the probability of getting better schedule using our approach is higher. Moreover, we store the results after scheduling all subtasks at any precedence level; we can incorporate fault tolerance in our approach. If we schedule using [14] and if some node fails, then again we have to schedule all subtasks present at all the levels shown in Figure 17. However, PLBGSA assigns genetic algorithm for subtasks of all levels separately; hence, we reschedule subtasks at a level where node failure happened and subtasks, beyond that level. This way we obtain results much faster as shown in Figure 18.

5. Conclusion and Future Scope of Work

We have applied genetic algorithm in every precedence level to schedule subtasks on P2P grid nodes. Moreover, PLBGSA is better and efficient than the algorithm proposed by Pop et al. [14]. Probability of finding good schedule is higher than the previous works. P2P grid resources are utilized more uniformly with PLBGSA. Further, fault detection and recovery mechanism is proposed in PLBGSA. This fault tolerance mechanism is yielding good results. We obtain near optimal schedule with a reduced number of generations in PLBGSA. In the future scope of work, we can apply other optimization heuristics using precedence level based scheduling for P2P grid. Also, we will incorporate task duplication technique before applying genetic scheduling at each precedence level in our future algorithm.

Abbreviations

PLBGSA:	Fault tolerant precedence level based genetic scheduling algorithm for P2P grid
ST_S^l:	Single subtask present on level l
$\mathrm{Wld}_N^{\mathrm{ST}_S}$:	Workload of task ST_S on node N
P:	Number of P2P grid nodes available
RnG:	Random number generator
$\mathrm{Ftm\_ST}_j^l$:	Time required to finish all subtasks j of level l
j:	Number of subtasks present at level l
ST_j^l:	Set of j subtasks present at level l
RWS:	Roulette wheel selection method
Rng_Sd:	Range of schedules
Sd_j^l:	Individual schedule to finish set j of subtasks at level l
Sd_{ji}^l:	ith schedule to finish set j of subtasks at level l
$\mathrm{Sd}_{1\mathrm{st}}^l$ and $\mathrm{Sd}_{2\mathrm{nd}}^l$:	Pair of schedules shortlisted for genetic operations from a defined range in previous schedules
$\mathrm{Sd}_{\mathrm{smallest}}^l$:	Schedule having the smallest $\mathrm{Ftm\_ST}_j^l$ from all schedules for level l
CxGO:	Crossover genetic operator
MtGO:	Mutation genetic operator
Max Level:	Highest level present in DAG based task t
u:	Total number of generations
GNo:	Generation number
h:	Size of generation
Ntl:	Number of subtasks at level l
NdF:	Node failure flag
c:	Node to which subtask is assigned in any schedule
$tld_C^{\mathrm{ST}_{ji}^l}$:	Load on c node of subtask i at level l
$\mathrm{CF}_c^{\mathrm{ST}_{ji}^l}$:	Workload after task i at level l is added to any node c
$\mathrm{CF}_c^{\mathrm{old}}$:	Old workload on node c
$\mathrm{Trt}_{XN}^{\mathrm{ST}_{ji}^l}$:	Time required in sending subtask i at level l from node X to node N
MEF:	Factor having maximum magnitude among 3 factors
Rbk^{l-1}:	Time at which results will be returned for all subtasks at previous level
k:	Number of DNAs selected from previous generation.

References

[1] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: state of the art and open problems," Tech. Rep. 2006- 504, School of Computing, Queen's University Kingston, Ontario, Canada, 2006.

[2] I. Foster and C. Kesselman, *The Grid: Blueprint for A New Computing Infrastructure*, Morgan Kaufmann, San Francisco, Calif, USA, 1998.

[3] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: enabling scalable virtual organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[4] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of job-scheduling strategies for grid computing," in *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing*, pp. 191–202, 2000.

[5] F. Pop and V. Cristea, "Intelligent strategies for DAG scheduling optimization in grid environments," in *Proceedings of the 16th International Conference on Control Systems and Computer Science (CSCS16 '07)*, pp. 98–103, Bucharest, Romania, 2007.

[6] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor—a hunter of idle workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems*, pp. 104–111, 1988.

[7] H. Mohamed and D. Epema, "KOALA: a co-allocating grid scheduler," *Concurrency Computation Practice and Experience*, vol. 20, no. 16, pp. 1851–1876, 2008.

[8] A. Agarwal and P. Kumar, "Economical task scheduling algorithm for grid computing systems," *Global Journal of Computer Science and Technology*, vol. 10, no. 11, pp. 48–53, 2010.

[9] S. Voulgaris, D. Gavidia, and M. Van Steen, "CYCLON: inexpensive membership management for unstructured P2P overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–216, 2005.

[10] A. J. Chakravarti, G. Baumgartner, and M. Lauria, "The organic grid: self-organizing computation on a peer-to-peer network," *IEEE Transactions on Systems, Man, and Cybernetics A*, vol. 35, no. 3, pp. 373–384, 2005.

[11] M. Fiscato, P. Costa, and G. Pierre, "On the feasibility of decentralized grid scheduling," in *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW '08)*, pp. 225–229, October 2008.

[12] P. Chauhan and Nitin, "Resource based optimized decentralized grid scheduling algorithm," *Advances in Computer Science, Engineering & Applications*, vol. 167, pp. 1051–1060, 2012.

[13] B. Simion, C. Leordeanu, F. Pop, and V. Cristea, "A hybrid algorithm for scheduling workflow applications in grid environments (icpdp)," in *Proceedings of on the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE and IS*, vol. 4804 of *Lecture Notes in Computer Science*, pp. 1331–1348, 2007.

[14] F. Pop, C. Dobre, and V. Cristea, "Genetic algorithm for DAG scheduling in Grid environments," in *Proceedings of the IEEE 5th International Conference on Intelligent Computer Communication and Processing (ICCP '09)*, pp. 299–305, August 2009.

[15] H. Cao, H. Jin, X. Wu, S. Wu, and X. Shi, "DAGMap: efficient scheduling for DAG grid workflow job," in *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (GRID '08)*, pp. 17–24, October 2008.

[16] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Springer, 2001.

[17] G. V. Iordache, M. S. Boboila, F. Pop, C. Stratan, and V. Cristea, "A decentralized strategy for genetic scheduling in heterogeneous environments," *Multiagent and Grid Systems*, vol. 3, no. 4, pp. 355–367, 2007.

[18] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu, "MonALISA: a distributed monitoring service architecture," in *Proceedings of the Computing in High Energy and Nuclear Physics (CHEP '03)*, pp. 1–8, La Jola, Calif, USA, 2003.

[19] A. Y. Zomaya and Y.-H. Teh, "Observations on using genetic algorithms for dynamic load-balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 9, pp. 899–911, 2001.

[20] U. Fissgus, "Scheduling using genetic algorithms," in *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS '00)*, pp. 662–669, April 2000.

[21] P. Chauhan and Nitin, "Decentralized computation and communication intensive task scheduling algorithm for p2p grid," in *Proceedings of the 14th International Conference on Computer Modelling and Simulation (UKSim '12)*, pp. 516–521, 2012.

[22] P. Chauhan and Nitin, "Fault tolerant decentralized scheduling algorithm for p2p grid," in *Proceedings of the 2nd International Conference on Communication, Computing & Security (ICCCS '12)*, vol. 6, pp. 698–707, Procedia Technology, 2012.

[23] Z. Dong, Y. Yang, C. Zhao, W. Guo, and L. Li, "Computing field scheduling: a fully decentralized scheduling approach for grid computing," in *Proceedings of the 6th Annual ChinaGrid Conference (ChinaGrid '11)*, pp. 68–73, August 2011.

[24] P. Townend and J. Xu, "Fault tolerance within a grid environment," in *Proceedings of the UK e-Science All Hands Meeting*, pp. 272–275, 2003.

[25] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauve, "Faults in Grids: why are they so bad and what can be done about it?" in *Fourth Workshop on Grid Computing*, pp. 18–24, 2003.

[26] S. Hwang and C. Kesselman, "A flexible framework for fault tolerance in the grid," *Journal of Grid Computing*, no. 13, pp. 251–272, 2003.