Hindawi Publishing Corporation
Journal of Engineering
Volume 2014, Article ID 202843, 14 pages
http://dx.doi.org/10.1155/2014/202843

Research Article

Decentralized Scheduling Algorithm for DAG Based Tasks on P2P Grid

Piyush Chauhan and Nitin

Department of CSE and IT, Jaypee University of Information Technology, P.O. Waknaghat, Solan, Himachal Pradesh 173234, India

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

Received 7 May 2013; Revised 18 November 2013; Accepted 27 November 2013; Published 22 January 2014

Academic Editor: Guisheng Zhai

Complex problems consisting of interdependent subtasks are represented by a direct acyclic graph (DAG). Subtasks of this DAG are scheduled by the scheduler on various grid resources. Scheduling algorithms for grid strive to optimize the schedule. Nowadays a lot of grid resources are attached by P2P approach. Grid systems and P2P model both are newfangled distributed computing approaches. Combining P2P model and grid systems we get P2P grid systems. P2P grid systems require fully decentralized scheduling algorithm, which can schedule interreliant subtasks among nonuniform computational resources. Absence of central scheduler caused the need for decentralized scheduling algorithm. In this paper we have proposed scheduling algorithm which not only is fruitful in optimizing schedule but also does so in fully decentralized fashion. Hence, this unconventional approach suits well for P2P grid systems. Moreover, this algorithm takes accurate scheduling decisions depending on both computation cost and communication cost associated with DAG's subtasks.

1. Introduction

Splitting a huge job into subtasks yields interdependent subtasks. Once predecessor subtasks return results only then will the execution of successor subtask take place. To characterize a set of subtasks and their dependency on each other we can use directed acyclic graph (DAG). Nodes represent subtasks and dependencies are denoted by arc joining the two nodes. Most of the DAG tasks are highly computation and communication intensive. Intertask dependencies lead to a very complex scenario to find a solution in an efficient manner. Moreover, because of financial constraints most of the organizations do not own high-end computational resources like cluster of supercomputers. The grid provides a solution to get out of this situation. We can access computational resources available on the grid and schedule our DAG based task upon them. Scheduling is the method to shortlist nodes from the available computational resources and then assign tasks upon them in an efficient manner. A lot of scheduling algorithms [1] are in place to schedule tasks upon grid [2, 3]. However, they use either single server as central scheduler or metascheduler approach. Due to political causes, depending upon central scheduler in a grid computing environment is not viable. Problem with metascheduler takes place when no single cluster has adequate computational resources to execute the bulky job. Moreover, scalability and bottleneck problems are present in both meta- and central-scheduler approach. These shortcomings directed the researcher's interest towards P2P and other decentralized approaches to the problem of grid scheduling. However, most of the initial P2P solutions for grid scheduling problem emphasized only the discovery of accessible computational resources [4–6]. In P2P approach, each of the resources present on grid takes scheduling decisions on its own [7]. P2P approach is also based the concept of decentralization like the ones proposed in [7–10]. Hence, P2P grid has come up as a tempting way to schedule the DAG based tasks. Scheduling algorithm targets at high throughput by utilizing idle nodes present in the P2P grid [11, 12]. Presently, most of the existing algorithms [13] schedule independent tasks over P2P [14] grid. Fully decentralized technique [15] (computing field scheduling) for scheduling tasks on the grid was proposed in [16]. The drawback of this approach [13, 16] is that it ignores the communication cost. In [17] we proposed fault tolerant decentralized scheduling

(FTDS) algorithm for grid, which schedules independent tasks by taking into consideration the communication and computational cost associated with tasks. However we require decentralized scheduling algorithm which schedules not only independent tasks but also interdependent tasks over P2P grid. While scheduling interdependent subtasks of huge job, scheduling algorithm should consider both the communication and computation cost associated with subtasks of the job. Scheduling subtasks of DAG based task on the heterogeneous decentralized grid is an NP-hard problem. Researchers have used a genetic algorithm to schedule DAG based tasks on a decentralized grid [18]. In this paper we propose a fully decentralized P2P grid scheduling (FDPGS) algorithm, which schedules subtasks of DAG based on communication and computation cost. FDPGS gives faster and better results in comparison to the genetic algorithm. The literature review is given in Section 2. Problem of DAG based task scheduling is explained in Section 3. FDPGS algorithm is proposed in Section 4. Simulation results are discussed in Section 5. Finally we conclude and mention future scope of the work in Section 6.

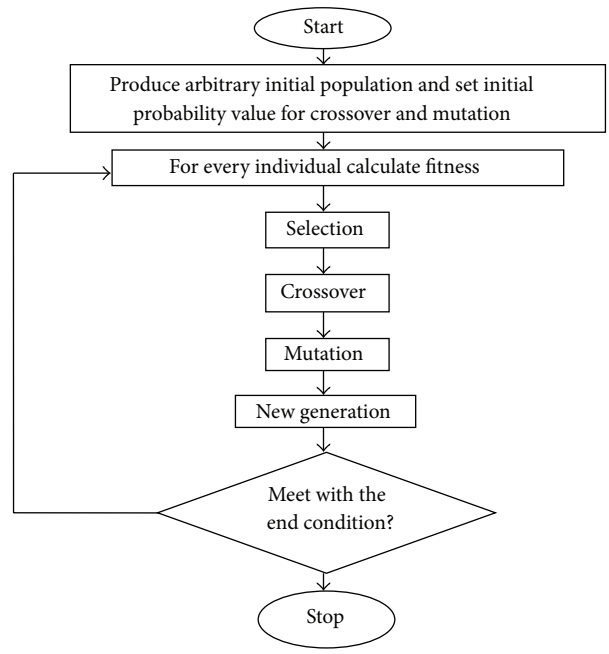

Figure 1: Basic genetic algorithm flow chart.

2. Related Work

Recently, a lot of researchers have proposed decentralized P2P grid scheduling techniques. The prominent one in them is [19]. It first shortlists resources from the grid, using CYCLON [14] gossip protocol. Then it schedules tasks on shortlisted computational resources using genetic algorithm. The limitation of this work is that it schedules only independent jobs. Another approach using P2P strategies for decentralized grid is proposed in [13]. It uses shaking algorithm originally used for video streaming in P2P network. In this approach authors ignore the cost to send input and output files and assume that negligible communication cost is required to send task to remote sites. As already mentioned fully decentralized technique [15] (computing field scheduling) for scheduling tasks on the grid was proposed in [16]. In [16] for any given task authors calculate computing field of that job on direct neighbors of grid resource where a task is generated. We store this data in the dynamic information list of node and schedule task on the node having the least magnitude of the computing field. The drawback of this approach is that it ignores the communication cost like in [13]. In our paper [17] we overcome this shortcoming by including the communication cost along with the computing field while making scheduling decision. In [17] we proposed fault tolerant decentralized scheduling (FTDS) algorithm for grid, which schedules independent tasks.

In [18], authors obtain schedule for DAG based task using optimization heuristic. The optimization heuristic used in [18] to obtain good schedule is genetic algorithm. The basic idea of genetic algorithms is given in Figure 1 [20]. In genetic algorithm we take initial population and then shortlist some parents from that population. These shortlisted parents are used to obtain new offspring utilizing genetic operators. From this new population, we shortlist those offspring which give the best results for desired properties. To obtain next generation we repeat the above steps till any offspring with desired values of properties is obtained.

Computer-executable generic variant of Fisher's formula [21] is known as genetic algorithm. This generalization as mentioned in [22] is expressed as follows.

Definition 1. "Concern with the interaction of genes on a chromosome, rather than assuming alleles act independently of each other, and enlargement of the set of genetic operators to include other well-known genetic operators such as crossing-over (recombination) and inversion."

Genetic algorithm is a four-step technique. In genetic algorithm an individual in a population is known as chromosome and symbolizes feasible solution to a dilemma. In scheduling, every chromosome gives a schedule of a batch of tasks on a group of computational resources. A chromosome can be denoted as a series of individual schedules (every single schedule is a queue of subtasks assigned to that node) for each computational resource in the group separated by a unique value. A second representation uses a matrix arrangement with computational nodes on one dimension and queues arranged on the second dimension. There is also third representation used in [18]. We used a variant of genetic algorithm given in [18] to compare with our work. In [18], each gene is represented as a twosome of values (T_j, P_i). This pair denotes that subtask T_j is deputed to processor P_i. This representation reduces computation costs as mentioned in [23]. We assign each subtask of DAG based task randomly on any processor to obtain an initial population of solutions. This work amplifies the mutation rate when population stagnates and vice versa. Genetic operator is applied on chromosomes to obtain a new population of chromosomes from previous chromosomes. Reference [18]

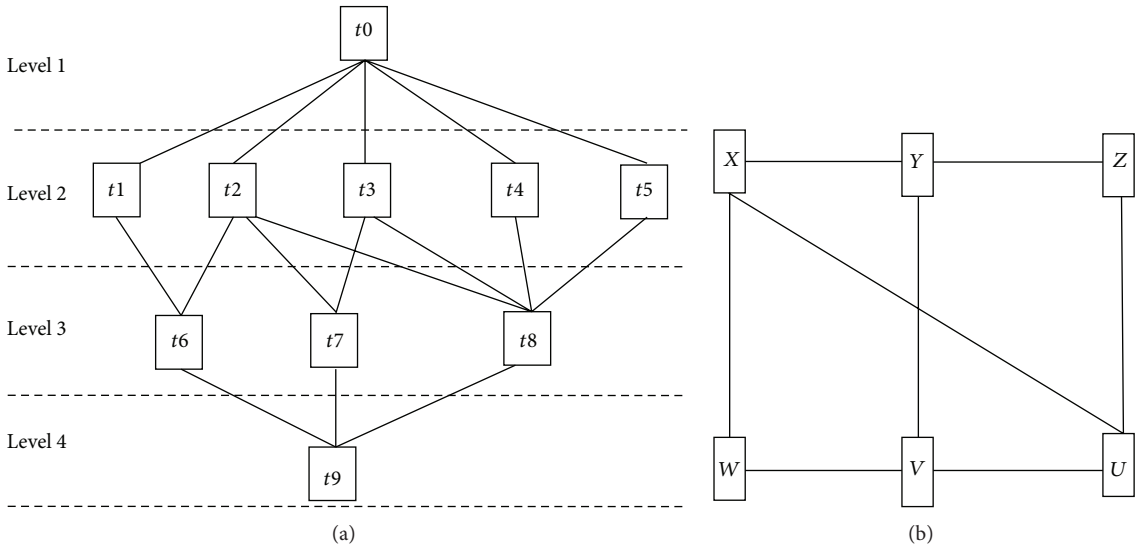

Figure 2: (a) DAG based sample task; (b) overlay P2P network.

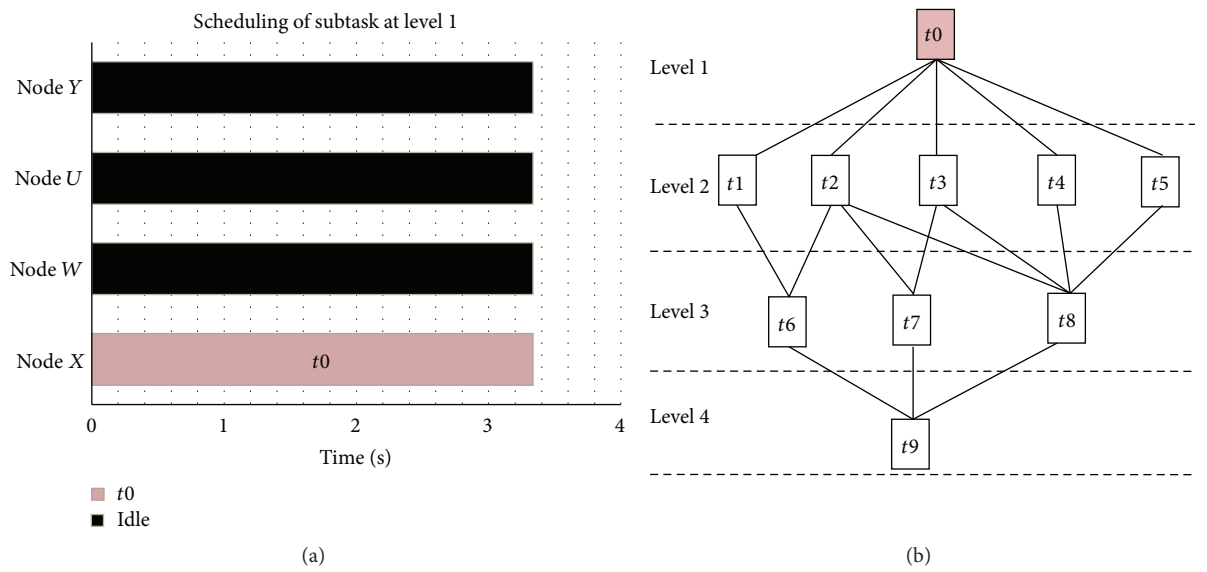

Figure 3: (a) Scheduling of origin subtask node present at level 1; (b) position of subtask in DAG based task at level 1.

put into practice the roulette wheel selection method [24]. However the genetic algorithm consumes a lot of time to find the schedule. Schedule length to finish complete DAG based task is taken as a parameter to shortlist parents to be used for crossover. Mutation rate increases when parent generation has schedule length same almost. Hence single test function is taken into consideration. As we increase the test functions complexity of genetic algorithm also increases. The algorithm proposed in this paper is compared with genetic algorithm. However to give genetic algorithm a fair chance we have used single parameter based genetic algorithm. Schedule length is taken as a parameter for selection.

We propose a new decentralized scheduling algorithm which efficiently schedules DAG based task on a P2P grid. Our algorithm takes scheduling decision based on computing field and communication cost associated with the DAG based task. Problem of DAG based task scheduling in decentralized grid is very complex; insight into this problem is given in the next section.

3. Problem of DAG Based Task Scheduling on Decentralized Grid

A computationally intensive task which consists of various subtasks interdependent on each other can be represented by directed acyclic graphs (DAG). The DAG based sample task is shown in Figure 2(a). $t0$ is an origin subtask node shown in Figure 2(a). Important fact about the origin subtask node is that there is no incoming edge. Hence $t0$ does not require a prerequisite output from any predecessor subtask because it is

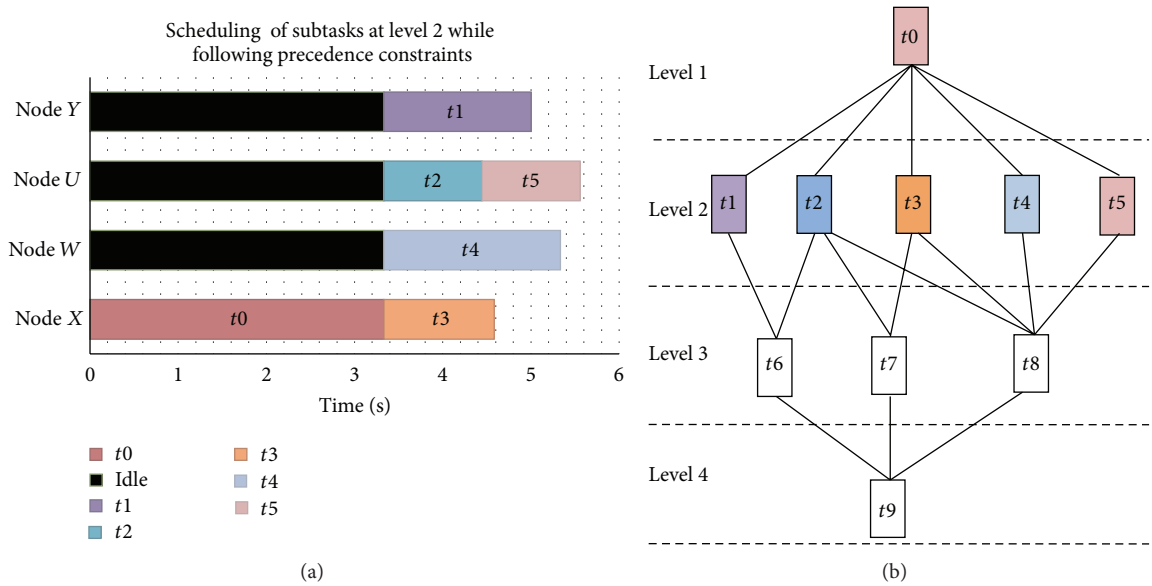

Figure 4: (a) Scheduling of subtasks present at level 2; (b) position of subtasks in DAG based task at level 2.

present at initial precedence level 1. Another type of subtask final node is $t9$. There is no outgoing edge in $t9$ because it is the last subtask of DAG based sample task. As soon as $t9$ finishes our complex task is completed. However $t9$ can start executing once subtasks present in previous precedence level (level 3) have finished and returned the results. Subtasks $t1$ to $t8$ also have such precedence level dependencies on parent subtask. We can schedule these subtask nodes on various computational resources of existing P2P network. The benefit is we can execute subtasks of DAG based task present at the same precedence level in parallel. However to schedule efficiently subtasks on available computational resources is an NP-hard problem.

Because of precedence constraints communication cost to send subtask from one computational resource to another varies. Moreover computational cost to calculate subtask on various computational resources varies on the basis of their computational capabilities. Further with increase in size of subtasks or number of available resources complexity of finding good schedule also increases manifolds. Makespan is time the to finish all subtasks of DAG based task. DAG based task scheduling problem targets at reducing makespan while following precedence constraints. This scheduling is better understood with a diagrammatic representation of level by level scheduling of sample DAG based task shown in Figure 2(a). We have taken overlay P2P network shown in Figure 2(b). We consider DAG based task is generated on node X. Further we execute subtasks either at X node or on direct neighbors of X which are nodes W, U, and Y.

In Figure 3(a) we have shown how first subtask is scheduled without worrying about precedence constraints as there is no parent task present before task $t0$. When $t0$ is scheduled at X node then the rest of the nodes do not execute any other subtask of sample DAG based task because all subtasks present at level 2 require results of $t0$.

As visible in Figure 4(a) we have scheduled all subtasks of level 2 in parallel once $t0$ returns results. Further subtasks of level 3 start executing in parallel on available computational resources when their parent tasks at level 2 have returned results. Scheduling of the subtasks of level 3 is shown in Figure 5(a).

Finally subtasks present at level 4 start executing on a suitable node as per scheduling algorithms policies, once all subtasks present at level 3 are complete. Figure 6(a) shows scheduling of subtask $t9$.

Scheduling these subtasks of DAG based task requires the scheduler to make decisions based on scheduling algorithms for DAG based tasks. Firstly all subtasks are assigned priority and then arranged on the basis of their priority. Subtask at lower precedence level gets superior priority as compared to subtask at higher precedence level. Subtask with top priority receives access to computational resources first. Once this top most origin node gets schedule, then subtask with second highest priority gets access to computational resources available. Subtask into consideration is scheduled to computational resource using grid scheduling algorithm for dependent task. These scheduling algorithms are further divided into two subparts static and dynamic. Static scheduling algorithms are of various types like list algorithm, cluster algorithm, and duplication based algorithm. In list scheduling algorithm firstly priority is assigned to all subtasks and then the subtask with highest priority is scheduled to node giving earliest start time, whereas our approach schedules subtask on computational resource finishing subtask faster.

4. Proposed Algorithm

We have proposed a fully decentralized P2P grid scheduling (FDPGS) algorithm for DAG based tasks on the grid. In the next section, we have used multiple variants of DAG based job to do exhaustive analysis. However, in this section to

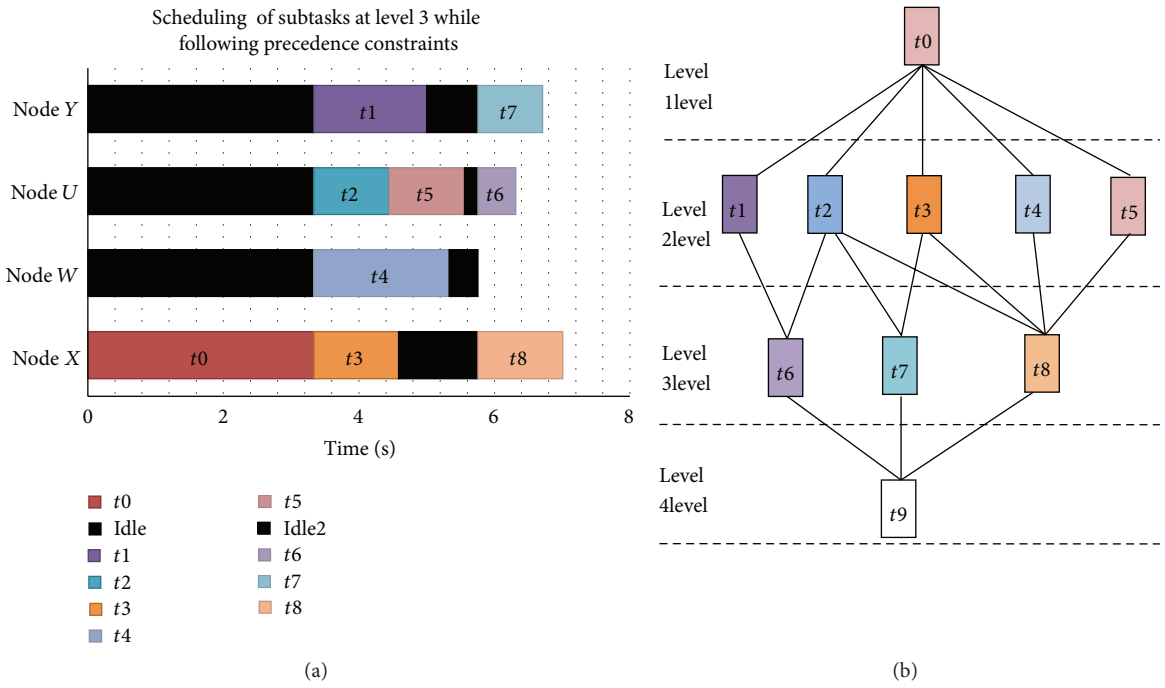

Figure 5: (a) Scheduling of subtasks present at level 3; (b) position of subtasks in DAG based task at level 3.

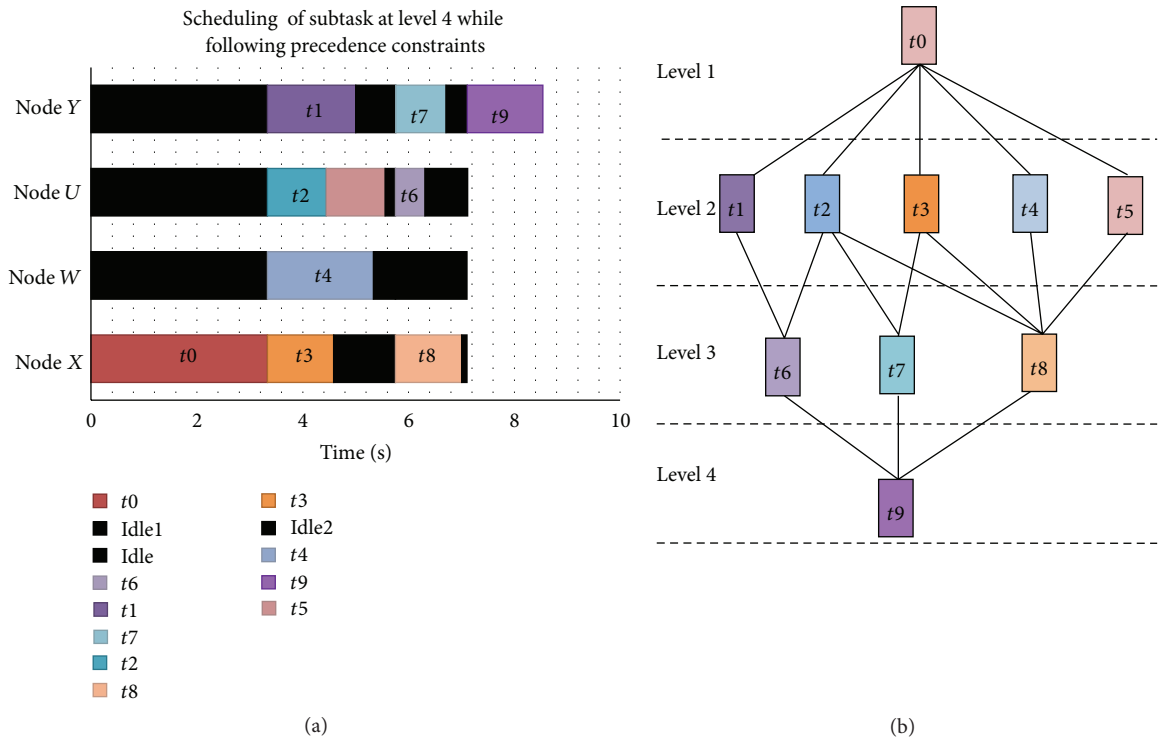

Figure 6: (a) Scheduling of subtasks present at level 4; (b) position of subtasks in DAG based task at level 4.

understand the basic work of FDPGS we have used single DAG based job consisting of 10 interdependent subtasks. We take scheduling decision with the help of contents of modified information list l present on each node. Task's subtasks are interdependent and they are represented by DAG. Sample DAG taken into consideration is shown in Figures 2(a) and 2(b) and represents overlay P2P network. All subtasks are scheduled based on computing field and communication cost attached to that subtask. In [16], the authors put forward the concept of computing field [16] to illustrate the workload

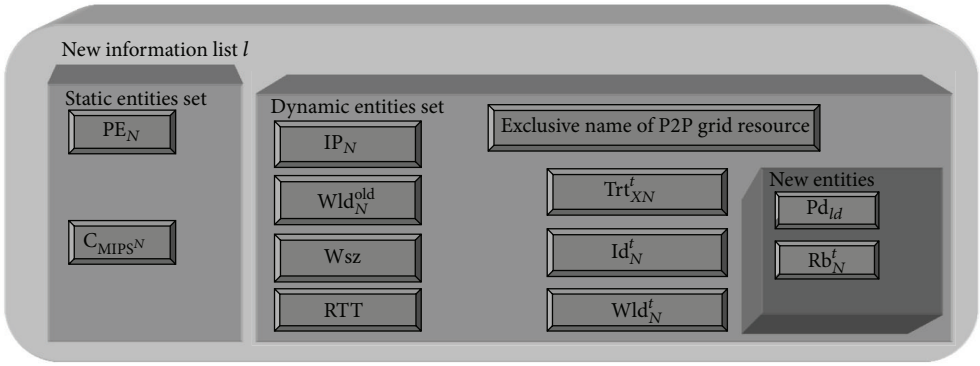

Figure 7: Information list l with all its subsets.

of grid node in a consolidated manner. Method to calculate computing field (CF) [16] is as given in (1) of the following definition.

Definition 2. Consider

$$\mathrm{CF} = \frac{\sum_{j=1}^{m} T_j}{\mathrm{PE} \times \mathrm{MIPS}_{\mathrm{PE}}}. \quad (1)$$

Here jth waiting job in a queue of length m has size of T_j million instructions. The number of cores in the node is given by PE. Single core of node can process $\mathrm{MIPS}_{\mathrm{PE}}$ number of million instructions per second. Computing field for a node is calculated with the help of (1). Entities required to calculate computing field are obtained from the dynamic information list present on that node. The dynamic information list contains values of various properties of the node and its direct neighbor. These values are called entities of that node and its neighbor.

In our approach modified information list l (shown in Table 1) present on all P2P grid resources will contain twelve entities. The first entity in the dynamic information list l contains distinctive name of P2P grid nodes. IP addresses of these nodes are stored in the subsequent row of list l, represented by IP_N for node N. Each node contains different number of processing elements mentioned in the fourth row of list l. PE_N symbolizes the total number of processing elements present on node N. Further, processing elements of each node hold different magnitude of processing capacity, calculated in terms of million instructions per second (MIPS). C_{MIPS^N} stands for MIPS of single processing element present on node N and is stored in the fifth row of list l. Each node can have different number of processing elements. However, processing elements present on the same node contain identical value of C_{MIPS^N}. Sixth row holds previous work load values of P2P grid nodes. Previous workload value is utilized to calculate new workload after new subtask is assigned to a node. $\mathrm{Wld}_N^{\mathrm{Old}}$ represents the old workload existing in P2P grid node N. This is required to calculate computation cost of subtask under consideration at a particular node. Wsz_{NZ} stands for window size between two nodes N and Z. Unit of the window size is kilobits per second and it is stored in row seven of information list l. Along with Wsz_{NZ} we require round trip time (RTT_{NZ}) between nodes N and Z to calculate communication cost. Magnitude of round trip time is stored in list l's eighth row.

Finally using values of seventh and eighth rows we get the cost of transferring any subtask t from node N to node Z, which is stored in the ninth row of list l. This entity is represented by Trt_{NZ}^t. Tenth row stores the load (ld_W^t) of individual subtask t on node N. We add weight of subtask to old workload and get assumed workload (Wld_N^t). Eleventh row stores the assumed workload (Wld_W^t) on P2P grid resource N if subtask t is assigned to it.

New entities in it are third and twelfth. The rest of entities are the same as in [17]. Third entity gives us the waiting time ($\mathrm{pd}_{\mathrm{ld}}$) for selected subtask t to start executing on any node. Waiting time is the time required by subtask (which consumes maximum time to finish) in previous precedence level to send back results after executing it. Twelfth row gives the time (Rb_N^t) when the output of subtask t is returned back to origin node from node N where it is executed.

Entities in information list l can be split into two sets; first set consists of static entities and second set consists of dynamic entities. Two new entities added to information list l used in [17] are shown as subsets of the second set in Figure 7.

According to changes in the value of dynamic entities, information list of node and its neighbors will be updated. If there is no change in the value of dynamic entities even then after a fixed time interval information list will be refreshed. This causes extra network traffic [19]. However network traffic due to time-to-time updating of information list on neighbor nodes will be extremely low [19].

Figure 8 explains graphically the basic working of FDPGS algorithm. Any P2P grid node where task T is located is known as origin node. Origin node either executes subtasks of task T itself or forwards the subtask to any of its direct neighbors, such that task T finishes in the minimum possible time.

According to precedence order one by one we schedule subtasks of DAG based task T, using fully decentralized P2P grid scheduling (FDPGS) algorithm shown in Algorithm 1. Algorithm consists of the following steps.

Table 1: An example of an information list l on P2P grid node X.

Serial number	Twelve entities: Name of entity	Immediate neighbors of P2P grid resources X: Node W	Node U	Node Y	Node X itself
1	Exclusive name of P2P grid resource	W	U	Y	X
2	IP address of P2P grid node, IP_N	IP_W	IP_U	IP_Y	IP_X
3	Waiting time for subtask t to begin execution, Pd_{ld}	Pd_{ld}	Pd_{ld}	Pd_{ld}	Pd_{ld}
4	Number of processing elements present in P2P grid node, PE_N	PE_W	PE_U	PE_Y	PE_X
5	MIPS of each processing element, C^N_{MIPS}	C^W_{MIPS}	C^U_{MIPS}	C^Y_{MIPS}	C^X_{MIPS}
6	"Previous workload on P2P grid node, Wld^{Old}_N"	Wld^{Old}_W	Wld^{Old}_U	Wld^{Old}_Y	Wld^{Old}_X
7	"Window size between two nodes, Wsz"	Wsz_{WX}	Wsz_{UX}	Wsz_{YX}	Wsz_{XX}
8	"Round trip time between origin and direct neighbor node, RTT"	RTT_{WX}	RTT_{UX}	RTT_{YX}	RTT_{XX}
9	"Cost to send subtask t on grid resource, Trt^t_{XN}"	Trt^t_{XW}	Trt^t_{XU}	Trt^t_{XY}	Trt^t_{XX}
10	"Individual workload of subtask t on grid resource, ld^t_N"	ld^t_W	ld^t_U	ld^t_Y	ld^t_X
11	"Assumed workload on grid resource after subtask t is assigned to it, Wld^t_N"	Wld^t_W	Wld^t_U	Wld^t_Y	Wld^t_X
12	"Time to return the result of subtask t, to origin node, Rb^t_N"	Rb^t_W	Rb^t_U	Rb^t_Y	Rb^t_X

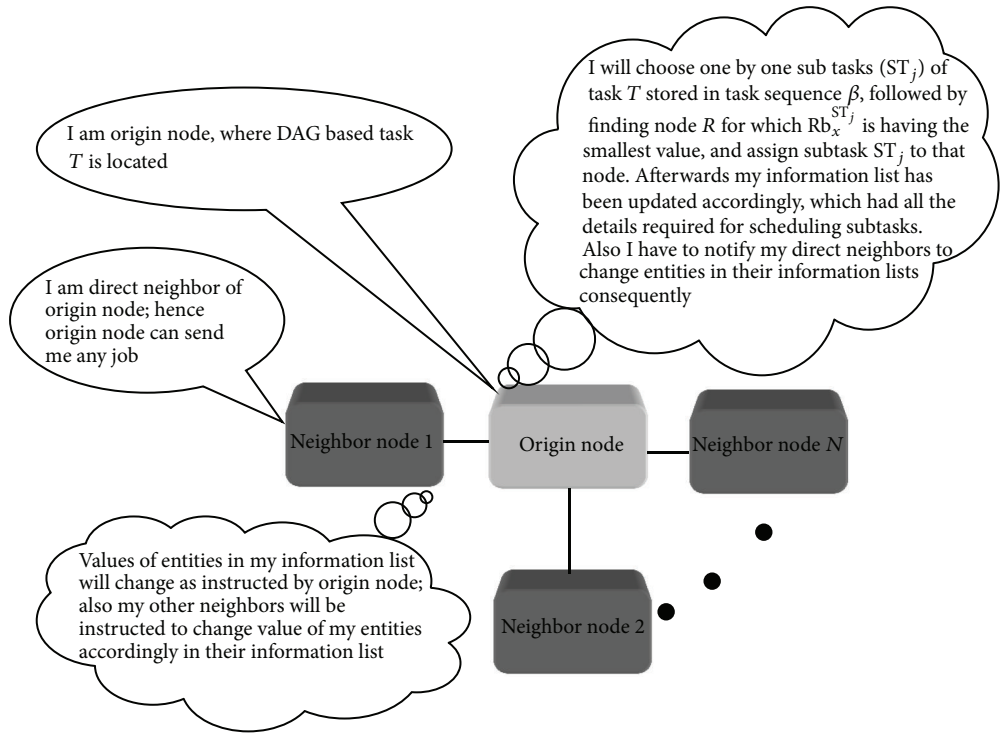

Figure 8: Basic logic of FDPGS algorithm.

Priority Based Task Sequence. If massive job is present at any node, then we arrange subtasks of job in nonincreasing order of their execution. DAG based task T is generated on the origin node (N_{origin}). Task T consist of various subtasks. Which are present on various precedence levels. We make priority based task sequence β of subtasks of T on the basis of precedence level.

Selection of Subtask for Scheduling and Predecessor Prerequisite. Now we choose first unscheduled subtask ST_j from β. If precedence level of subtask ST_j is one, then there is no predecessor of the subtask. Hence time for the predecessor subtask to finish and return results (Pd_{ld}) will be zero. However if the precedence levels of previous and present subtask are not the same, Pd_{ld} becomes equal to resultback

Input: DAG based task t generated on origin node $\left(N_{\text{origin}}\right)$.

BEGIN
(1) Construct priority based task sequence β.
(2) Do {
(3) Choose first unscheduled subtask $\left(\text{ST}_j\right)$ from β.
(4) if (precedence level of $\text{ST}_j = 1$)
(5) {
(6) $\text{Pd}_{\text{ld}} = 0$.
(7) }
(8) else if (precedence level of ST_{j-1}! = precedence level ST_j)
(9) {
(10) $\text{Pd}_{\text{ld}} = \text{RbK}$.
(11) }
(12) for (all nodes present in list l of N_{origin}).
(13) {
(14) compute $\text{ld}_N^{\text{ST}_j}$ on node using (4) and store in list l
(15) calculate $\text{Trt}_{XN}^{\text{ST}_j}$ on node using (2) and store in list l
(16) if $\left(\text{Wld}_N^{\text{Old}} \geq \text{Trt}_{XN}^{\text{ST}_j} \&\& \text{Wld}_N^{\text{Old}} \geq \text{Pd}_{\text{ld}}\right)$
(17) {
(18) Calculate assumed workload $\text{Wld}_N^{\text{ST}_j}$ using (5)
(19) }
(20) else if $\left(\text{Trt}_{XN}^{\text{ST}_j} \geq \text{Wld}_N^{\text{Old}} \&\& \text{Trt}_{XN}^{\text{ST}_j} \geq \text{Pd}_{\text{ld}}\right)$
(21) {
(22) Calculate assumed workload $\text{Wld}_N^{\text{ST}_j}$ using (6)
(23) }
(24) else if $\left(\text{Pd}_{\text{ld}} \geq \text{Trt}_{XN}^{\text{ST}_j} \&\& \text{Pd}_{\text{ld}} \geq \text{Wld}_N^{\text{Old}}\right)$
(25) {
(26) Calculate assumed workload a $\text{Wld}_N^{\text{ST}_j}$ using (7)
(27) }
(28) Calculate $\text{Rb}_N^{\text{ST}_j}$ for subtask ST_j and store in list l.
(29) }
(30) Find node R for which $\text{Rb}_N^{\text{ST}_j}$ is having smallest value and assign task to R.
(31) Update value of $\text{Wld}_R^{\text{Old}}$ in list l on node R and in list of all its direct neighbors.
(32) if (precedence level of $\text{ST}_j = 1$ || precedence level of ST_{j-1}! = precedence level ST_j)
(33) {
(34) $\text{RbK} = \text{Rb}_R^{\text{ST}_j}$
(35) }
(36) else if (precedence level of ST_{j-1} = precedence level ST_j)
(37) {
(38) if $\left(\text{RbK} \leq \text{Rb}_R^{\text{ST}_j}\right)$
(39) {
(40) $\text{RbK} = \text{Rb}_R^{\text{ST}_j}$
(41) }
(42) }
(43) } while (there are unscheduled subtasks in β)
END

Output: Schedule for subtasks of DAG based task t.

Algorithm 1: Fully decentralized P2P grid scheduling (FDPGS) algorithm for DAG based task.

(RbK). RbK is time taken by predecessor subtask to finish execution and return results. If there are more than one subtask at previous level, then predecessor subtask taking the maximum time to return result is taken as resultback(RbK).

Discovering the Most Suitable Node for Execution of Selected Subtask. Now for all nodes present in list l, Algorithm 1 will compute the load $\text{ld}_N^{\text{ST}_j}$ of single subtask (ST_j) and store in list l. Also we will store in list l time $(\text{Trt}_{XN}^{\text{ST}_j})$ to send subtask

from N_{origin} to neighbor node N. Assumed load ($\text{Wld}_N^{\text{ST}_j}$) is calculated for all neighbor nodes and N_{origin}.

Assumed workload ($\text{Wld}_N^{\text{ST}_j}$) will depend upon three factors which are $\text{Wld}_N^{\text{Old}}$, $\text{Trt}_{XN}^{\text{ST}_j}$, and Pd_{ld}. This dependency is caused because ready time $\text{Rtm}_N^{\text{ST}_j}$ of node N for subtask ST_j depends upon magnitude of these three entities.

Transport time $\text{Trt}_{XN}^{\text{ST}_j}$ to send subtask ST_j from node X to node N is calculated

$$\text{Trt}_{XN}^{\text{ST}_j} = \left(\frac{T_{\text{ST}_j}}{\text{Wsz}_{XN}} \right) \times \text{RTT}_{XN}. \tag{2}$$

In the above equation T_{ST_j} is the size of subtask in Kb. Wsz_{XN} is the window size between nodes X and N. RTT_{XN} is the round trip time between nodes X and N.

A third entity in list l, shown as third row in Table 1 is waiting time (Pd_{ld}). Waiting time (Pd_{ld}) is zero when level of ST_j is one and it is equal to RbK when precedence level of ST_{j-1} is not equal to precedence level of ST_j.

Assumed workload is calculated with the help of the general equation.

$$\text{Wld}_N^{\text{ST}_j} = \text{Rtm}_N^{\text{ST}_j} + \text{ld}_N^{\text{ST}_j}. \tag{3}$$

Here *individual weight* $\text{ld}_N^{\text{ST}_j}$ of single subtask ST_j on a grid node is calculated using

$$\text{ld}_N^{\text{ST}_j} = \frac{T_{\text{ST}_j}}{\text{PE} \times C_{\text{MIPS}}^N}. \tag{4}$$

T_{ST_j} is the size of subtask ST_j in million instructions. The number of processing elements is present in the node represented by PE. The processing capability of a single core in node N is symbolized as C_{MIPS}^N in (4).

Assumed workload $\text{Wld}_N^{\text{ST}_j}$ on node N when subtask ST_j is assigned to N is calculated using one of the three equations stated below. The equation whose condition is satisfied will be shortlisted to calculate $\text{Wld}_N^{\text{ST}_j}$.

Case 1. When $\text{Wld}_N^{\text{Old}} \geq \text{Trt}_{XN}^{\text{ST}_j}$ and $\text{Wld}_N^{\text{Old}} \geq \text{Pd}_{\text{ld}}$, then $\text{Rtm}_N^{\text{ST}_j} = \text{Wld}_N^{\text{Old}}$; hence (3) becomes

$$\text{Wld}_N^{\text{ST}_j} = \text{Wld}_N^{\text{Old}} + \text{ld}_N^{\text{ST}_j}. \tag{5}$$

Case 2. When $\text{Trt}_{XN}^{\text{ST}_j} \geq \text{Wld}_N^{\text{Old}}$ and $\text{Trt}_{XN}^{\text{ST}_j} \geq \text{Pd}_{\text{ld}}$, then $\text{Rtm}_N^{\text{ST}_j} = \text{Trt}_{XN}^{\text{ST}_j}$; hence (3) becomes

$$\text{Wld}_N^{\text{ST}_j} = \text{Trt}_{XN}^{\text{ST}_j} + \text{ld}_N^{\text{ST}_j}. \tag{6}$$

Case 3. When $\text{Pd}_{\text{ld}} \geq \text{Trt}_{XN}^{\text{ST}_j}$ and $\text{Pd}_{\text{ld}} \geq \text{Wld}_N^{\text{Old}}$, then $\text{Rtm}_N^{\text{ST}_j} = \text{Pd}_{\text{ld}}$; hence (3) becomes

$$\text{Wld}_N^{\text{ST}_j} = \text{Pd}_{\text{ld}} + \text{ld}_N^{\text{ST}_j}. \tag{7}$$

Using $\text{Wld}_N^{\text{ST}_j}$ we calculate time $\text{Rb}_N^{\text{ST}_j}$ for subtask to return results to N_{origin}. Once we have calculated $\text{Rb}_N^{\text{ST}_j}$ for all nodes in list l, we assign subtask to node with the minimum value of $\text{Rb}_N^{\text{ST}_j}$.

Updating Information List and Finding Value of Resultback for Next Subtask in Task Sequence. Finally we update value of $\text{Wld}_R^{\text{Old}}$ in list l on node R and in list of all its direct neighbors. After this, when precedence level of ST_j is one or precedence level of ST_{j-1} is not equal to precedence level ST_j, then RbK becomes $\text{Rb}_R^{\text{ST}_j}$. If precedence levels are equal and magnitude of RbK is less than $\text{Rb}_R^{\text{ST}_j}$, then RbK becomes $\text{Rb}_R^{\text{ST}_j}$.

Resultbacktime ($\text{Rb}_N^{\text{ST}_j}$) is the twelfth entity in list l. It is the time required by a node to calculate subtask and send the results back to the node where subtask is generated. Resultbacktime depends upon two factors; first one is assumed workload of the node where subtask ST_j is assigned including the load of subtask ST_j. Second factor is the round trip time to send result from node where it is executed to origin node where DAG based task T was initially present. Resultbacktime is calculated with the help of

$$\text{Rb}_N^{\text{ST}_j} = \text{Wld}_N^{\text{ST}_j} + \text{RTT}_{XN}. \tag{8}$$

This way we will schedule all subtasks of DAG using fully decentralized P2P grid scheduling (FDPGS) algorithm. Our algorithm is fully decentralized as for every huge job present on various nodes those nodes will acts as origin nodes. In this example task T is present at X node and node X act as origin node. If task T is present at the Y, then Y will act as an origin node. The same is true for other nodes of P2P grid.

FDPGS algorithm gives results better than genetic algorithm as can be confirmed from the next section.

5. Simulation Results

We have considered a DAG task T which is divided into 10 subtasks shown in Figure 2(a). Each subtask is of million instructions in size and Kb in magnitude. Details of the subtasks of DAG based task T are shown in Table 2.

Now this DAG based task T is generated on node 0 which is shown in virtual network topology as node X. Node X is having direct neighbors: node 1 as W, node 2 as U, and node 3 as Y in overlay P2P network. Specifications of P2P nodes are shown in Table 3.

When we schedule randomly our subtask $t0$ of task T, it is assigned to node 3 with new magnitude of workload on node 3. However, our subtask $t1$ has to wait for more time before it can execute. Cause of this delay is that we have to first transfer $t0$ to node 3 from node 0, which takes some time. First, we add new workload and transport time and then we finally add roundtrip time between node 0 and node 3 to it. This is how we get waiting time for subtask 1.

Similarly we schedule the rest of subtasks and task T finishes at 16.10 seconds as shown in Figure 9(a). Random schedule is obtained by running 20 times random scheduling

Table 2: Details of 10 subtasks of DAG based task T.

Subtask name	$t0$	$t1$	$t2$	$t3$	$t4$	$t5$	$t6$	$t7$	$t8$	$t9$
Precedence level	1	2	2	2	2	2	3	3	3	4
Magnitude in Kb	800	700	400	300	600	400	200	350	350	550
Size in MI	8.0	7.0	4.0	3.0	6.0	4.0	2.0	4.0	3.0	6.0

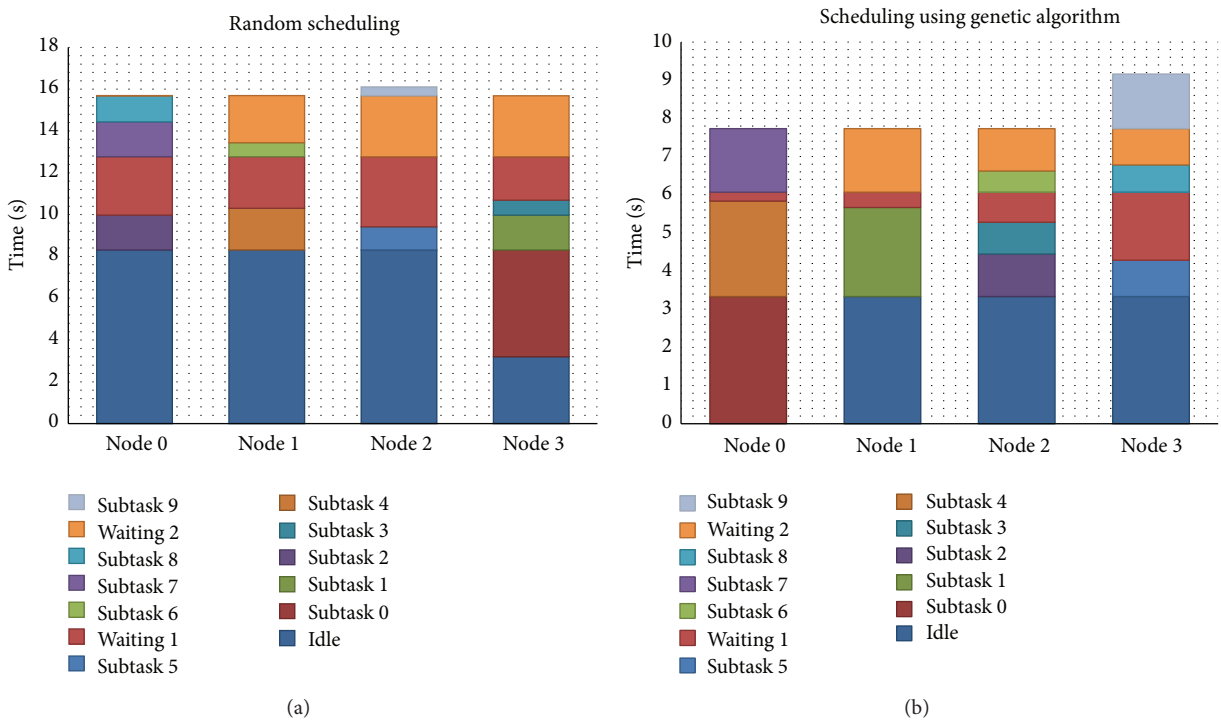

Figure 9: (a) Detailed timewise schedule for all subtasks of DAG based task T using random scheduling; (b) genetic algorithm.

Table 3: Specifications of nodes X, W, U, and Y.

Node name	X (origin node)	W	U	Y
Number of PE	2	3	4	3
MIPS of single PE (seconds)	1.2	1.0	0.9	1.4
Round trip time (seconds)	0.1	0.4	0.2	0.4
Window size (Kb)	75.0	100.0	75.0	100.0

algorithm and then choosing the random schedule with minimum finish time in these 20 runs. In Figure 9(b) we have represented scheduling of all subtasks of Task T using genetic algorithm. Genetic algorithm simulated has makespan as single parameter. Finish time of task T reduces drastically, and also waiting time of individual subtasks of T reduces as is visible in Figure 11 when we schedule using genetic algorithm. However, time to find schedule increases manifolds as illustrated in Figure 17 when we reckon up schedule using genetic algorithm. Memory being used to find schedule also increases when compared to random scheduling as shown in Figure 16. Now we are scheduling subtasks of the same DAG based task T on the very same overlay P2P network using FDPGS algorithm. Finish time of subtask $t0$ is 3.33 seconds when assigned to node 0. Subtask $t0$ is assigned to node 0 because it returns results faster as calculated with the help of FDPGS algorithm. Details of scheduling subtasks according to FDPGS algorithm are shown in Figure 10. It is visible in Figure 11 that waiting time of subtasks has reduced when we apply FDPGS algorithm. It is visible in Figure 15 that DAG based task T finishes with the FDPGS algorithm in almost half the time as compared to random scheduling. Moreover FDPGS also utilizes P2P grid nodes more uniformly as shown in Figure 14. Utilization of P2P nodes when genetic and random scheduling are done is shown in Figures 13 and 12, respectively.

FDPGS takes 98.61% less time than a genetic algorithm to find the schedule. Moreover, schedule of FDPGS algorithm gives results for DAG based task T in 6.83% less time than genetic algorithm. However to find a schedule FDPGS algorithm consumes 29.40%, 22.72% more memory than random scheduling and genetic algorithm, respectively. Nowadays P2P grid resources come with abundant amount of memory. Therefore memory consumption is a miniscule issue, when compared to time in which schedule is calculated. In addition, small in magnitude schedule is obtained using a FDPGS algorithm for DAG based task T, which is the most sought-after trait required in any decentralized grid scheduling algorithm.

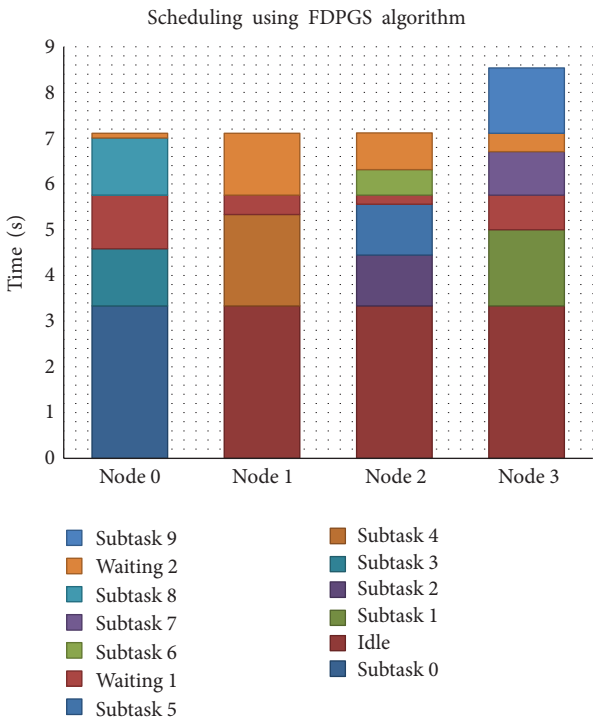

Figure 10: Fully decentralized P2P grid scheduling (FDPGS) algorithm.

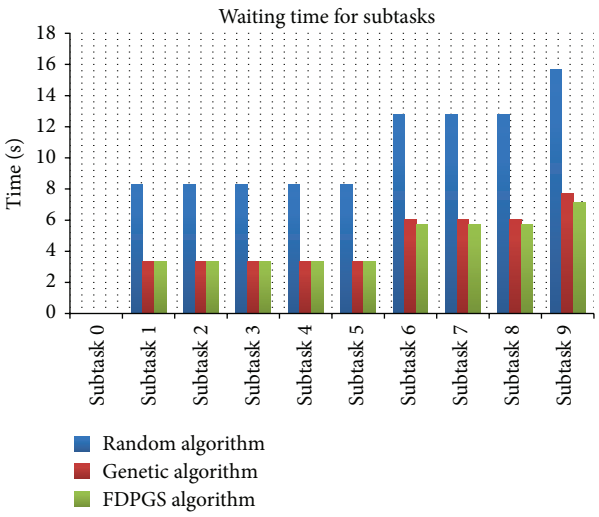

Figure 11: Waiting time for subtasks of DAG based task T when we schedule using random scheduling, genetic algorithm, and FDPGS algorithm.

Comparison of genetic algorithm and FDPGS algorithm finish time for 10 different DAG based tasks is shown in Figure 18. All DAG based tasks gave good results with FDPGS algorithm over a variant of genetic algorithm proposed in [18]. As shown in Figure 19, when we schedule using FDPGS a communication intensive DAG based task, then the last subtask in priority based task sequence β will have waiting a time always less as compared to the one obtained by genetic

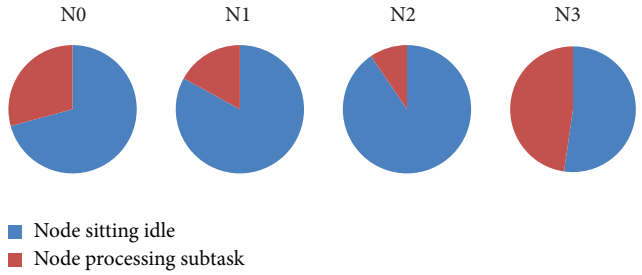

Figure 12: Utilization of P2P nodes in random scheduling.

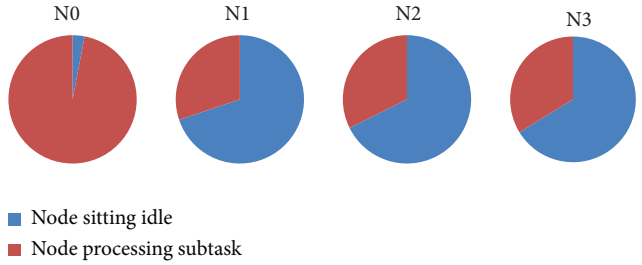

Figure 13: Utilization of P2P nodes when genetic algorithm is used for scheduling.

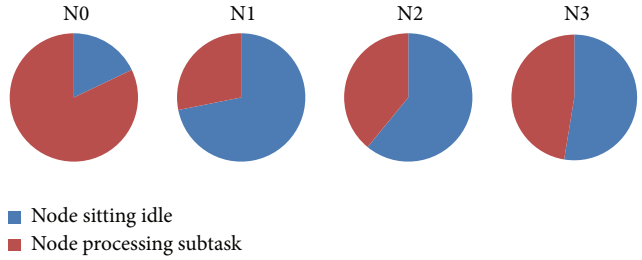

Figure 14: Utilization of P2P nodes when FDPGS algorithm is used for scheduling.

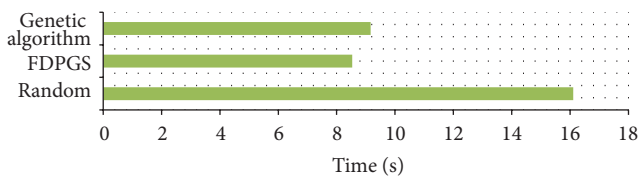

Figure 15: Time is taken by random, genetic algorithm and FDPGS to finish DAG based task T.

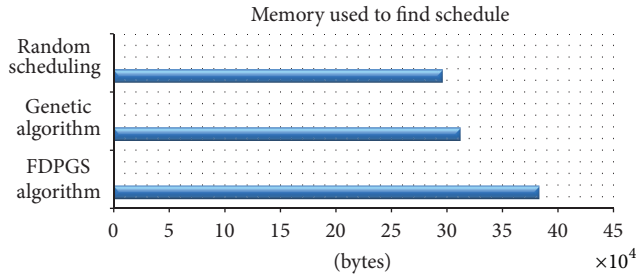

Figure 16: Memory used to find the schedule in bytes to calculate a schedule using random scheduling, genetic algorithm and FDPGS algorithm.

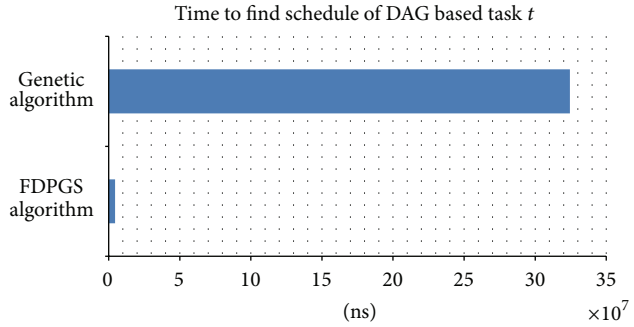

Figure 17: Time in nanoseconds required to calculate a schedule using random scheduling, genetic algorithm, and FDPGS algorithm.

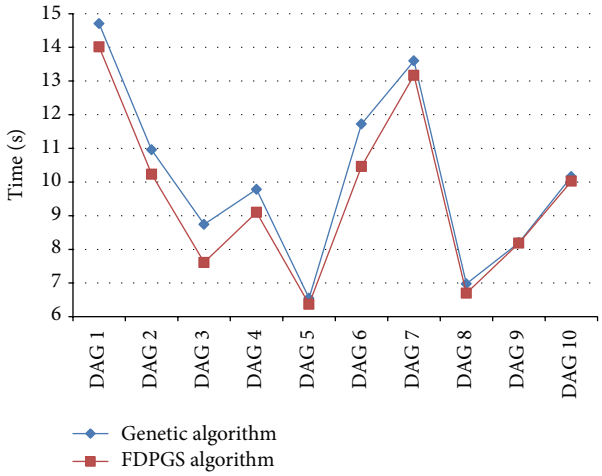

Figure 18: Comparison of genetic algorithm and FDPGS algorithm finish time for 10 DAG based tasks.

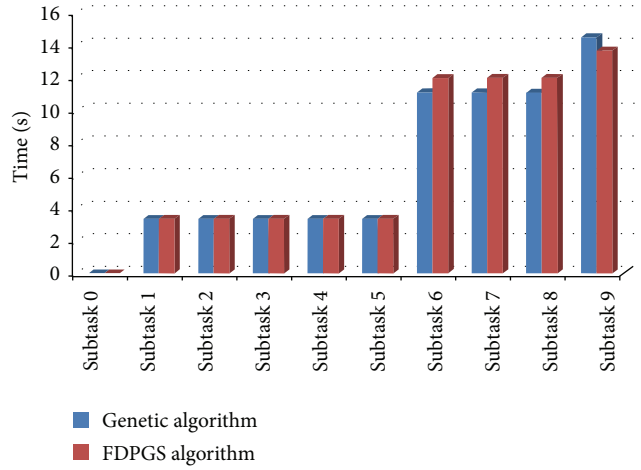

Figure 19: Waiting time for subtasks of communication intensive DAG based task using genetic algorithm and FDPGS algorithm.

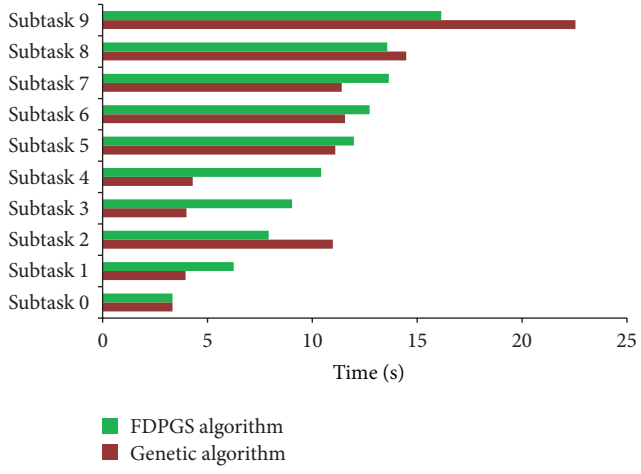

Figure 20: Finish time of subtasks of communication intensive DAG based task using genetic algorithm and FDPGS algorithm.

algorithm. Also Figure 20 explains that finish time of last subtask in priority based task sequence β of communication intensive DAG based task will be less when we use FDPGS algorithm. Figure 21 shows that FDPGS gives better results than genetic algorithm [18] for 10 examples of communication intensive DAG based task. For computation intensive DAG based task all subtasks will have less or the same waiting time as shown in Figure 22, when FDPGS is used. Also finish time of last subtask will always be less as shown in Figure 23, when we schedule using FDPGS algorithm. Figure 24 shows that FDPGS gives better results than genetic algorithm [18] for 10 examples of computation intensive DAG based tasks also.

Hence it is visible that our proposed algorithm FDPGS gives results faster and helps in uniformly scheduling DAG based task on neighbors of origin node.

6. Conclusion and Future Scope of Work

At present internet age, vast pool of high potential computation resources spread worldwide. Most of the existing scheduling techniques are not decentralized, whereas P2P grid resources connected over the internet work with highly decentralized fashion. The recent proposed decentralized scheduling algorithms only schedule independent tasks. However, in today's modern age, huge DAG based tasks are common. In this paper, we have introduced a fully decentralized P2P grid scheduling (FDPGS) algorithm which schedules subtasks of DAG based task. FDPGS schedules the subtasks by taking into contemplation three factors. The first two factors are computation cost and communication cost related to the subtasks. Final factor is the waiting time for the subtask because of predecessors and precedence constraints. Our algorithm yields good results and we plan to further use task duplication technique for scheduling DAG based tasks on P2P grid. Another aspect which could be considered for future research is fault tolerance.

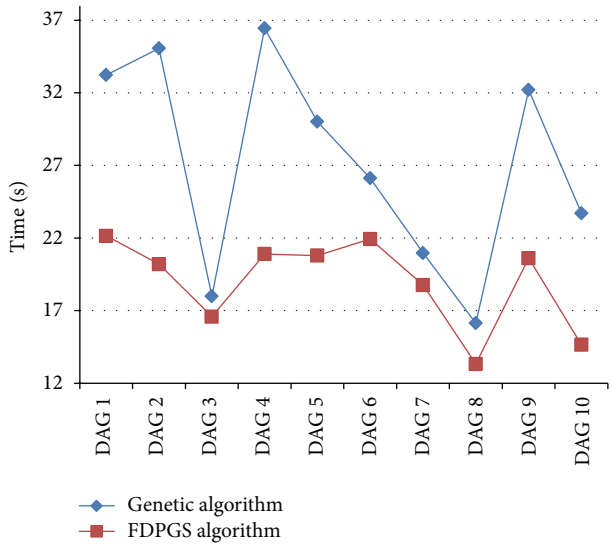

Figure 21: Comparison of genetic algorithm and FDPGS algorithm for communication intensive 10 DAG tasks.

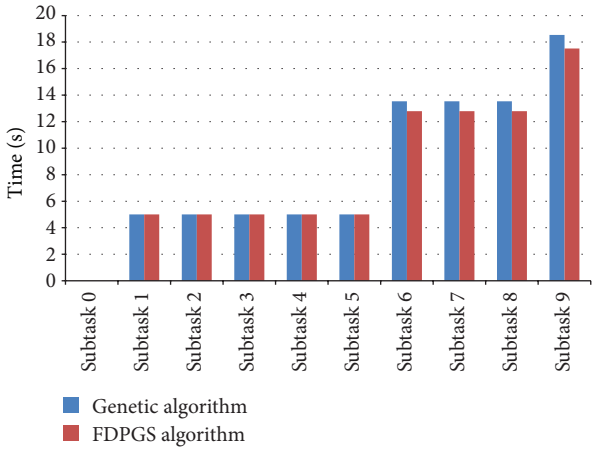

Figure 22: Waiting time for subtasks of computation intensive DAG based task using genetic algorithm and FDPGS algorithm.

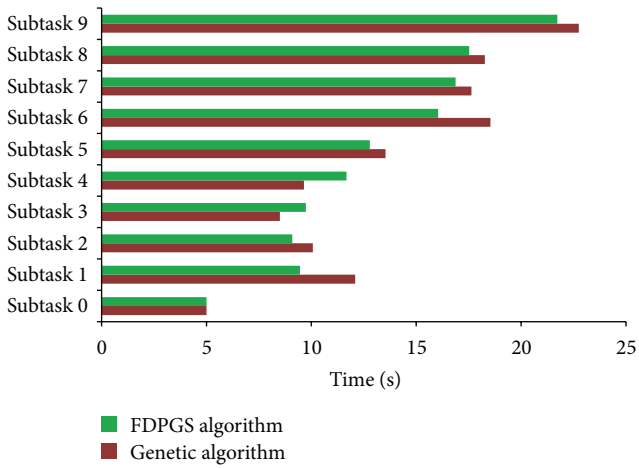

Figure 23: Finish time of subtasks of computation intensive DAG based task using genetic algorithm and FDPGS algorithm.

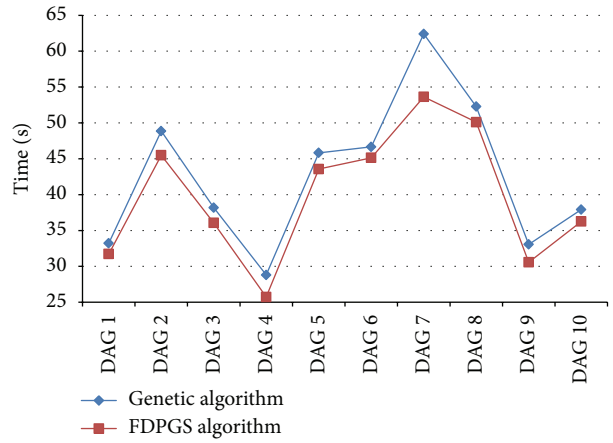

Figure 24: Comparison of genetic algorithm and FDPGS algorithm for computation intensive 10 DAG tasks.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

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, Canada, 2006.

[2] 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.

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

[4] 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.

[5] N. Drost, R. V. van Nieuwpoort, and H. Bal, "Simple locality-aware co-allocation in peer-to-peer supercomputing," in *Proceedings of the 6th IEEE International Workshop on Global and Peer-2-Peer Computing*, vol. 2, pp. 8–14, May 2006.

[6] G. Iordache, M. Boboila, F. Pop, C. Stratan, and V. Cristea, "A decentralized strategy for genetic scheduling in heterogeneous environments," in *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, vol. 4276 of *Lecture Notes in Computer Science*, pp. 1234–1251, Springer, Berlin, Germany, 2006.

[7] 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.

[8] P. Chauhan and Nitin, "Resource based optimized decentralized grid scheduling algorithm," in *Advances in Computer Science, Engineering & Applications*, vol. 167 of *Advances in Intelligent and Soft Computing*, pp. 1051–1060, Springer, Berlin, Germany, 2012.

[9] R. Bertin, A. Legrand, and C. Touati, "Toward a fully decentralized algorithm for multiple bag-of-tasks application scheduling on grids," in *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (GRID '08)*, pp. 118–125, October 2008.

[10] X. Vasilakos, J. Sacha, and G. Pierre, "Decentralized as-soon-as-possible grid scheduling: a feasibility study," in *Proceedings of the 2nd IEEE Workshop on Grid and P2P Systems and Applications (GridPeer '10)*, pp. 1–6, August 2010.

[11] A. A. Azab and H. A. Kholidy, "An adaptive decentralized scheduling mechanism for peer-to-peer desktop grids," in *Proceedings of the 2008 International Conference on Computer Engineering and Systems*, pp. 364–371, November 2008.

[12] X. Wen, W. Zhao, and F. Meng, "Research of grid scheduling algorithm based on P2P-Grid model," in *Proceedings of the International Conference on Electronic Commerce and Business Intelligence (ECBI '09)*, pp. 41–44, June 2009.

[13] C. Grimme, J. Lepping, J. M. Picon, and A. Papaspyrou, "Applying P2P strategies to scheduling in decentralized Grid computing infrastructures," in *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW '10)*, pp. 295–302, September 2010.

[14] 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. 197–217, 2005.

[15] X. Vasilakos, J. Sacha, and G. Pierre, "Decentralized as-soon-as-possible grid scheduling: a feasibility study," in *Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN '10)*, pp. 1–6, August 2010.

[16] 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*, pp. 68–73, August 2011.

[17] P. Chauhan and Nitin, "Fault tolerant decentralized scheduling algorithm for P2P grid," in *Proceedings of the 2nd International Conference on Communication, Computing and Security (ICCCS '12)*, vol. 6, pp. 698–707, 2012.

[18] 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.

[19] 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*, pp. 225–229, October 2008.

[20] S. G. Li and Z. M. Wu, "Business performance forecasting of convenience store based on enhanced fuzzy neural network," *Neural Computing and Applications*, vol. 17, no. 5-6, pp. 569–578, 2008.

[21] J. Holland, *Hidden Order: How Adaptation Builds Complexity*, Addison-Wesley, Reading, Mass, USA, 1995.

[22] Lastaccessed: 21. 1. 2013, http://www.scholarpedia.org/article/Genetic_algorithms.

[23] M. Wu and D. D. Gajski, "Hypertool: a programming aid for message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 330–343, 1990.

[24] A. Y. Zomaya and Y. 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.