# A New Shuffled Genetic-based Task Scheduling Algorithm in Heterogeneous Distributed Systems

**Mirsaeid Hosseini Shirvani** ✉[1]

*1)* Department of Computer Engineering, Sari Branch, Islamic Azad University,

Sari, Iran

Corresponding Author : mirsaeid_hosseini@iausari.ac.ir

## Abstract

Distributed systems such as Grid- and Cloud Computing provision web services to their users in all of the world. One of the most important concerns which service providers encounter is to handle total cost of ownership (TCO). The large part of TCO is related to power consumption due to inefficient resource management. Task scheduling module as a key component can has drastic impact on both user response time and underlying resource utilization. Such heterogeneous distributed systems have interconnected different processors with different speed and architecture. Also, the user application which is typically presented in the form of directed acyclic graph (DAG) must be executed on this type of parallel processing systems. Since task scheduling in such complicated systems belongs to NP-hard problems, existing heuristic approaches are no longer efficient. Therefore, the trend is to apply hybrid meta-heuristic approaches. In this paper, we extend a meta-heuristic shuffled genetic-based task scheduling algorithm to minimize total execution time, *makespan*, of user application. In this regard, we take benefit of other heuristics such as Heterogeneous Earliest Finish Time (HEFT) approach to generate smart initial population by applying a new shuffle operator which makes a fortune to explore feasible and promising individuals in the search space. We also conduct other genetic operators in right way to produce final near to optimal solution. To reach concrete results we have conducted several scenarios. Our proposed algorithm outperforms in term of average *makespan* compared with other existing approaches such as HEFT versions and QGARAR.

**Keywords:** Task Scheduling, cloud computing, directed acyclic graph (DAG)

## 1. Introduction

Distributed systems include unlimited computing and storage resources which are interconnected with high speed networks [1-3]. Users utilize distributed systems such as Grid Computing known as E-Science and Cloud Computing known is E-Commerce instead of procurement of overprovisioned resources to cover only peak demand in hotspot time [4]. Cloud- and Grid computing provision elastic and economic services to their subscribers based on pay-per use basis for the sake of economy of scale [2, 4]. For instance, several academic projects which are computation-intensive in nature request huge amount of processing resources in which it is not affordable for a university to have such systems. For another example, take a company that wants to backup massive amount of data relevant to several

months of day-to-day of myriad transaction reports. In this line, Amazon S3 service can figure out the aforementioned problem by low cost charges in $/month/GB storage basis [5]. On the other hand, response time is one of the most important quality of service (QoS) parameter in such systems because users would eventually release the providers with low received QoS [6]. In this regards, key element which is determinant in response time and distributed system's efficiency is scheduling module. Basically, inefficient scheduling approach leads degradation in resource utilization and increasing total execution time. So, design and implementation of smart and efficient scheduling framework is inevitable in such systems. Usually, application programs contains different subtasks which may have dependency between them; this type of application is presented in the form of directed acyclic graph (DAG). On the other hand, underlying infrastructure in distributed systems is named datacenter in which it comprises with different physical machines and servers interconnected with high speed local area networks (LANs); it also is scalable geographically. In the other words, datacenters can even be connected via wide area networks (WANs) although it appears to be a unique entity for its users so-called transparent system. Scheduling in such systems refers to allocating computing resources to user tasks which are placed in the application DAG; the goal is to minimize total execution time of all tasks subject to preserving sub-tasks inter-dependencies constraints. So, the afore-said problem computationally belongs to NP-hard category. Nevertheless, in literature, when a user requests an application to be done; task graph applications, shown in the form of DAG, are usually scheduled in heterogeneous distributed systems with list scheduler algorithms [11]. List schedulers provide a sorted list of subtasks based on predefined priority; in fact, each subtask is placed in the list based on the weight that the algorithm assigns to each node. One of the most famous heuristic list scheduler algorithm is Heterogeneous Earliest Finish Time (HEFT) approach [11]. For instance UpwardRank, DownwardRank and LevelRank are three different version of HEFT-based algorithms [10]. List schedulers are executed in two phases. In the first phase as mentioned, each subtasks which is presented as a node in DAG is placed in a sorted list based on predetermined priority. The sorted list should guarantee that the list is topological sort. In other words, it should satisfy precedence constraints. In the second phase, the scheduler should find the best processor to map the subtasks on it. It means that algorithm should find the processor which deliver output of subtasks with earliest finish time (EFT) in the second phase [13]. In this line, evolutionary algorithms have been adopted in literature to solve such combinatorial problems; but for the sake of ever-increasing complexity in such systems and problems too, existing heuristic approaches are no longer efficient. Therefore, the trend is to apply hybrid meta-heuristic approaches. It is the reason for extending a new meta-heuristic shuffled genetic-based algorithm to obviate shortcomings of current heuristics. We take benefit of other heuristic approaches in our new proposed genetic algorithm. For instance, to constitute individuals for making initial population, we create a multiple priority queues and insert sorted lists which produced by HEFT versions. Then, we shuffle the lists based on our new algorithm to take benefit of both exploration and exploitation in search space. We also conduct other GA operators in right way to gain near optimal and low overhead solutions. The result of implementation indicates that it is a promising technique. The main contributions of this paper are as follows:

1- To extend a new shuffled genetic-based heuristic algorithm by using multi priority queue; in this way, we can produce smart individuals in initial population by applying shuffle operator; it benefits of both exploration and exploitation in search space.

2- To apply HEFT approach for task-map-to-processors to find suitable processor which guarantees minimum EFT

The rest of the current paper is organized as follows; related works are discussed in section 2. System framework containing system and application models are placed in section 3. The HEFT definition is brought in section 4. Section 5 is dedicated to our proposed approach. Simulation and Evaluation of our work are brought in Section 6. Finally, Section 7 presents conclusion and future direction.

## 2. Related Works

Several works have been done in literature to present clear solution for task scheduling problems in distributed systems. A PSO-based task scheduling algorithm has been proposed for efficient handling of cloud resources in scientific programs with respect to deadline defined by user [7]. This fitness function was reduction on execution time and cost based defined in fitness function [7]. Although the proposed algorithm is promising, it is specifically designed for scientific problems which are only computation-intensive. Another task scheduling algorithm based on ant colony optimization (ACO) was used to minimize *makespan* of tasks submitted in cloud environment [8]. The proposed algorithm has had good efficiency on cloud computing because it launches appropriate virtual machine based on requested tasks. Nevertheless, the algorithm is limited for only independent tasks. An economical and low-cost task scheduling algorithm operates based on two strategies; the first strategy allocates the best virtual machines to tasks based on Pareto dominance. The second strategy reduces overall cost by mapping unimportant tasks in next phase [9]. The main focus of scheduling algorithm of the paper was on economic viewpoint which may sacrifice user response time. Multi-Criteria Task Scheduling in Distributed Systems based on Fuzzy TOPSIS has been presented in 30$^{th}$ IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) [12]. In this work, authors applied Fuzzy TOPSIS approach for ranking of alternatives based on users' and providers' preferences. It is originally designed for real-time applications which have time limitation and independent ones. Another heuristic-based named Bacteria Foraging based has been proposed for independent task scheduling on cloud computing environment [14]. Despite its efficiency, it is not applicable for programs with dependent tasks. A quantum genetic algorithm with rotation angle refinement has been presented in literature to schedule dependent tasks on distributed systems such as cloud datacenters [15]. The proposed algorithm is essentially a good approach for quantum computing. It also relies on random generated population for initial phase leading more iteration rounds to reach satisfaction criteria. Literature review reveals that a hybrid meta-heuristic approach is favorable in which it would take benefit from other approaches to reach good results in terms of optimality and take benefit of both exploration and exploitation in search space.

## 3. System Framework

Our proposed system framework is depicted in Fig. 1. It has different components such as Front-end module, Cloud Broker, Scheduler, VMs and Datacenter.

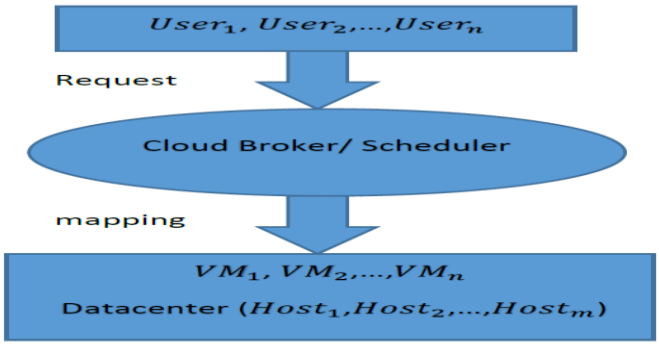


Fig. 1. Proposed system framework

Front-end module receives user applications to be executed on parallel physical machines. Cloud Broker delivers users semantic of cloud service ability and related quality of service. The datacenter contains a set of *m* different heterogeneous processors which are interconnected with high speed networks. The heterogeneity is based on architecture and speeds; each of which can run multiple VMs. Such system model is illustrated in Fig. 2.

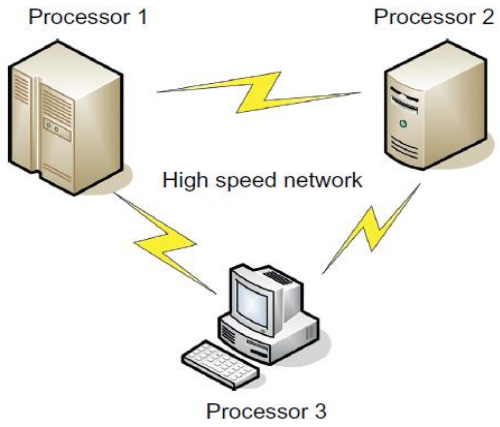


**Fig. 2**. A full connected parallel system with three heterogeneous computing system [10]

Moreover, application model is presented in the form of DAG. Fig.3 depicts such applications which have sub-tasks inter-dependencies.

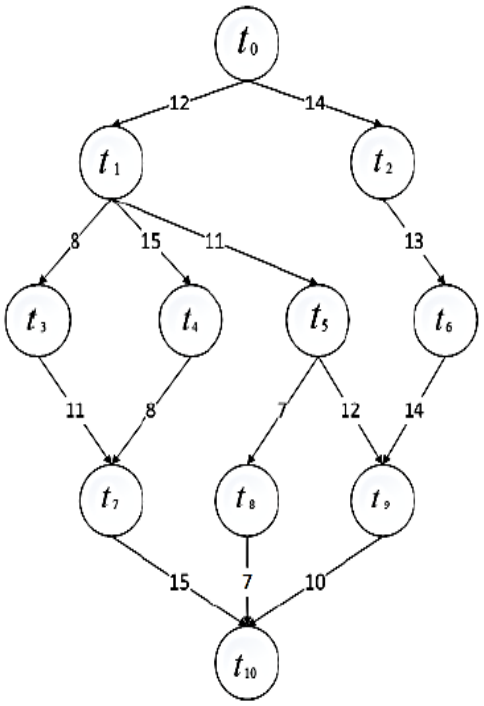


**Fig. 3**. An example of DAG application with 11 subtasks

Each task graph, a DAG, has several nodes which are defined for computations and edges which show average communication cost between nodes. The edge also indicates precedence constraint between nodes. Every subtask should be executed on one computational node from our target parallel system. Also, each DAG has two special nodes $T_{Entry}$ and $T_{Exit}$ that not have predecessor and successor nodes respectively. As the defined system is heterogeneous in nature, processing of each subtask of DAG over different computational nodes have different cost. For instance, Table 1 shows the different task execution times on different processing nodes. Moreover, the last column indicates average execution time.

**Table 1**. Different task execution times on different processing nodes

| Tasks | Processors | | | Average computation Cost |
|---|---|---|---|---|
| | $P_0$ | $P_1$ | $P_2$ | $\bar{\omega}$ |
| $t_0$ | 7 | 9 | 8 | 8 |
| $t_1$ | 10 | 9 | 14 | 11 |
| $t_2$ | 5 | 7 | 6 | 6 |
| $t_3$ | 6 | 8 | 7 | 7 |
| $t_4$ | 10 | 8 | 6 | 8 |
| $t_5$ | 11 | 13 | 15 | 13 |
| $t_6$ | 12 | 15 | 18 | 15 |
| $t_7$ | 10 | 13 | 7 | 10 |
| $t_8$ | 8 | 9 | 10 | 9 |
| $t_9$ | 15 | 11 | 13 | 13 |
| $t_{10}$ | 8 | 9 | 10 | 9 |

## 4. Heterogeneous Earliest Finish Time (HEFT)

List schedulers are famous scheduling algorithms in distributed systems. The Heterogeneous Earliest Finish Time (HEFT) belongs to list scheduler category. It was firstly introduced by Topcuoglu et al. for static task scheduling on limited heterogeneous parallel processing systems [11]. On the other hand, Grid Computing is static in nature whereas Cloud Computing is a dynamic paradigm. Moreover, scheduling can be either static or dynamic; static scheduling applies prior information and does not vary with time while a dynamic scheduling applies system-state run time information [15]. To apply such static-oriented algorithm in dynamic environment, we can determine static time window to treat the problem with static fashion [6]. HEFT applies two important functions: Earliest Finish Time (EFT) and Earliest Start Time (EST). The former indicates the earliest time in which a processor $P_j$ can executes subtask $T_i$ whereas the latter indicates the earliest time that the execution can be started. The earliest start time for entry task in DAG is zero in which equation (1) calculates. Moreover, functions EST and EFT for other nodes are calculated by equations (2) and (3) respectively [11].

$$\mathrm{EST}(n_{entery}, p_j) = 0 \tag{1}$$

$$\mathrm{EST}(n_i, p_j) = \max\{\, avail\,\{j\}, \max\left(AFT(n_m) + c_{m,i}\right)_{n_m \in pred(n_i)}\} \tag{2}$$

$$\mathrm{EFT}(n_i, p_j) = w_{i,j} + \mathrm{EST}(n_i, p_j) \tag{3}$$

The function $pred(n_i)$ in equation (2) indicates to a set of all predecessor nodes of $n_i$ in DAG. Moreover, the term $avail\,\{j\}$ illustrates the time that processor $p_j$ accomplished the last task on itself and it is ready for the next task to execute. The inner *max* in equation (2) means that the actual finish time (AFT) of the last task in $pred(n_i)$ should be determined. Meanwhile, the outer *max* indicates that it may happen the situation that the output of last subtask of $n_i$ in $pred(n_i)$ becomes ready later than $avail\,\{j\}$. On the other words, despite $p_j$ readiness, the execution time is postponed until the time which the last precedence subtask of $n_i$ is ready; because this procedure precludes of swerving in dependency constraints violation in task graph DAG. On the other hand, if the value of $avail\,\{j\}$ is greater than the finish time of last subtask in $pred(n_i)$, despite execution of all subtasks in $pred(n_i)$ , the actual execution will be started until the processor $p_j$ is ready. The parameter $c_{m,i}$ indicates average transfer time between processors $p_m$ and $p_i$ . If $m=i$ then $c_{m,i} = 0$. The actual execution time for subtask $n_i$ on processor $p_j$ is calculated by equation (4) [11]. Moreover, the total execution time of DAG so called *makespan*, is calculated by equation (5) [11].

$$AFT(n_i, p_j) = min_{1 \le l \le m} EFT(n_i, p_l) \tag{4}$$

$$makespan = \max\{\, AFT(n_{exit})\} \tag{5}$$

The aforesaid algorithm is executed in two phases. In the first phase subtasks are sorted based on predetermined priority. Upward ranking, Downward ranking and Level ranking are three typical approaches in this ambit [10]. For instance, in Upward ranking approach each subtask is assigned with a weight from *exit* subtask to *entry* subtask. This value for *exit* subtask is

calculated by equation (6) [11]; for other subtasks the value is calculated by equation (7) recursively [11].

$$rank_u(n_{exit}) = \overline{w_{exit}} \tag{6}$$

$$rank_u(n_i) = \overline{w_i} + max_{n_j \in succ(n_i)}(\overline{c_{i,j}} + rank_u(n_j)) \tag{7}$$

In addition to, average execution time for each node is calculated by equation (8) [11]. Function *succ($n_i$)* indicates to all successors of $n_i$ in DAG application.

$$\overline{w_i} = \sum_{j=1}^{q} {}^{w_{i,j}}/_{q} \tag{8}$$

On the other side, in Downward approach, priority value is calculated from *entry* node to *exit* node. This value is considered zero for *entry* subtask whereas for other subtasks the value is recursively calculated by equation (9) [11].

$$rank_d(n_i) = max_{n_j \in pred(n_i)}(rank_d(n_j) + \overline{w_j} + \overline{c_{i,j}}) \tag{9}$$

In the Level ranking approach, level for each subtask is calculated by equation (10) [11]. Then, each subtask is placed in sorting list based on its level value in increasing order. In case of the same level for two different subtasks, the subtask which have greater value in sum of Upward rank and Downward values is selected from left to right in ordered list.

$$Level(T_i) = \begin{cases} 0, & if\ T_i = T_{entery} \\ \max(Level(T_j))_{T_j \in pred(T_i)} + 1, & otherwise \end{cases} \tag{10}$$

For instance, for DAG depicted in Fig. 3, Table 2 shows the priority values of different approaches for each node.

**Table 2**. Task priority based on three approaches

| Tasks | $rank_u(t_i)$ | $rank_d(t_i)$ | Level | $rank_u(t_i) + rank_u(t_i)$ |
|---|---|---|---|---|
| $t_0$ | 102 | 0 | 0 | 102 |
| $t_1$ | 79 | 20 | 1 | 99 |
| $t_2$ | 80 | 22 | 1 | 102 |
| $t_3$ | 52 | 39 | 2 | 91 |
| $t_4$ | 50 | 46 | 2 | 96 |
| $t_5$ | 57 | 42 | 2 | 99 |
| $t_6$ | 61 | 41 | 2 | 102 |
| $t_7$ | 34 | 62 | 3 | 96 |
| $t_8$ | 25 | 62 | 3 | 87 |
| $t_9$ | 32 | 70 | 3 | 102 |
| $t_{10}$ | 9 | 93 | 4 | 102 |

Hence, three queues of subtasks [ $t_0t_2t_1t_6t_5t_3t_4t_7t_9t_8t_{10}$],[ $t_0t_1t_2t_3t_6t_5t_4t_7t_8t_9t_{10}$] and [$t_0t_2t_1t_6t_4t_3t_9t_7t_9t_8t_{10}$] are valid topological sort, based on Upward, Downward and Level ranking approaches [10]. As stated before, in the second phase, a subtask is picked up from in front of sorted list to be scheduled; the algorithm searches to find a processor which guarantees the earliest finish time for that subtask. Therefore, different states will be produced by small input DAG; even worse, when

the input DAG is huge, it cannot be harnessed by deterministic algorithms; the reason why we develop heuristic approach to figure out such problem.

## 5. Proposed Genetic Algorithm (GA) for Task Scheduling in Heterogeneous Systems

In this section, we present our proposed heuristic method which take benefit from other approaches, i.e., Upward, Downward and Level rankings in their initial population. One of the oldest and applicable heuristic method is genetic algorithm (GA) which inspires from nature. In the other words, each individual adapting itself with environment circumstance will survive. So, emulated problems can work accordingly. Every candidate solution, individual in GA, which has more vicinity to optimal solution will remain in next generation. GA has some operators to produce appropriate generation. Our proposed method is depicted in Fig. 4.

**Input**: GA and DAG characteristics
**Output**: An optimal task scheduling
1. Call **INIT** procedure to create initial population
2. **Repeat**
3. Call **Mapper** procedure to apply task-to-processor mapping and evaluate fitness
4. Copy the elitism individuals directly to the next generation
5. **Repeat**
6. Call **Roulette-wheel** operator to select candidates
7. Call **Crossover** operator
8. Call **Mutation** operator
9. **Until** the new population is completed
10. Replace the old population with new one
11. **Until** the termination criteria are met
12. **Return** an optimal schedule

**End**

**Fig. 4**. Algorithm for Proposed Method

### 5.1 Gens, Chromosomes and INIT procedure

In GA, phenotype should be converted to genotype. In this regards, arbitrary encoding can be extended. In this paper, each subtask can be used as a gen; so, sequence of gens makes a chromosome. For instance, $[\ t_0 t_1 t_2 t_3 t_6 t_5 t_4 t_7 t_8 t_9 t_{10}]$ can be known as a valid chromosome because each subtask is visited after its parents. To create initial population, we call INIT procedure to make current generation with *Popsize* fixed individuals. In this work we take *Popsize*=100. To take benefit from other approaches, we take three chromosomes from Upward, Downward and Level rankings. The rest chromosomes are created randomly and shuffled from multi queue with suitable distribution because inefficient dispersion of individuals has drastic affection on algorithm's optimality, speed and convergency. The pseudo code of INIT procedure is depicted in Fig 5. It also uses a Shuffle procedure which is detailed in Fig 6. Also in Fig. 6, procedure feasible at line #18 checks whether a chromosome is valid or not.

Procedure **INIT**
**Input**: A DAG with its characteristics, *PopSize*, *n* as length of Population and chromosome respectively
**Output**: An initialize Population with *PopSize*
1. Size=1;
2. Ind1= Call HEFT-UpwardRank to make first individual
3. Copy Ind1 into Population

4. Size=size+1;
5. Ind2= Call HEFT-DownwardRank to make second individual
6. Copy Ind2 into Population
7. Size=size+1;
8. Ind3= Call HEFT-LevelRank to make third individual
9. Copy Ind3 into Population
10. Size=size+1;
11. Pop1= **Shuffle** ( Population(1) , n )
12. Pop2= **Shuffle** ( Population(2) , n )
13. Pop3= **Shuffle** ( Population(3) , n )
14. Copy *PopSize*/3 of individuals from Pop1 to Population so that containing its first individual.
15. Copy *PopSize* /3 of individuals from Pop2 to Population so that containing its first individual.
16. Copy *PopSize* /3 of individuals from Pop3 to Population so that containing its first individual.

Return *Population* with *PopSize*

**Fig 5**. INIT procedure for making initial population

Procedure **Shuffle ( *Individual, n* )**
**Input**: An *individual* and *n* as length of its chromosome size
**Output**: A list containing *k* members of individuals known as *CurrentPop* with *CurrentPopSize.*

1. CurrentPopSize=1;
2. CurrentPop= [];
3. Copy individual to *CurrentPop*
4. For i=1 to n-1 do
5. Find $T_j$ as the last predecessor of $T_i$.
6. Find $T_k$ as the first successor of $T_i$.
7. Temp=$T_i$
8. L=k-1;
9. For q=i to k-2 do
10. $T_q = T_{q+1}$
11. End-For
12. $T_L = Temp$
13. If **feasible**(new individual) then
14. Add new individual to CurrentPop
15. CurrentPopSize=CurrentPopSize+1;
16. End-if
17. // Again it repeats for original individual to make new one
18. L=j+1;
19. For q=i down to j+2 do
20. $T_q = T_{q-1}$
21. End-For
22. $T_L = Temp$
23. If **feasible**(new individual) then
24. Add new individual to CurrentPop
25. CurrentPopSize=CurrentPopSize+1;
26. End-if
27. End-For

Return *CurrentPop* with *CurrentPopSize*

**Fig 6**. Shuffle procedure to explore search space

### 5.2 Mapper Procedure

In this phase, each chromosome which is representative of a candidate solution can be mapped on multi heterogeneous processors by HEFT algorithm, c.f. in 2.2. Then, *makespan* can be considered as a fitness value for evaluation; namely, each chromosome with lowest *makespan* is the fittest.

### 5.3 Roulette-Wheel procedure

Portion of individuals, with special probabilistic, are selected to create new generation. This will happen by crossover operator. In roulette-wheel, we take the procedure in which each chromosome with high fitness has greater chance to be selected. The equation (11), is the formula that indicates each chromosome chance to be selected.

$$p_i = \frac{fitness_i}{\sum_{j=1}^{PopSize} fitness_j} \tag{11}$$

### 5.4 Crossover operator

To create new generation, GA utilizes crossover operator. After couple of individuals are selected by roulette-wheel operator, individuals are input for crossover operator. From the outset, one random number is generated in [1..chromosome-length]. Then, it halves parents in two subsections. The left part of the first parent is placed for left part of the first child; then the right part of child is filled by gens which not appeared in this child when second parent's gens are visited from left to right. Accordingly, the left part of second parent is placed to left part of second child; the rest will be filled by gens which not appeared in this child when first parent's gens are visited from left to right. It is well illustrated in Fig. 7. In this phase, we take crossover rate, known as *crate*, 80 percent of *PopSize*.

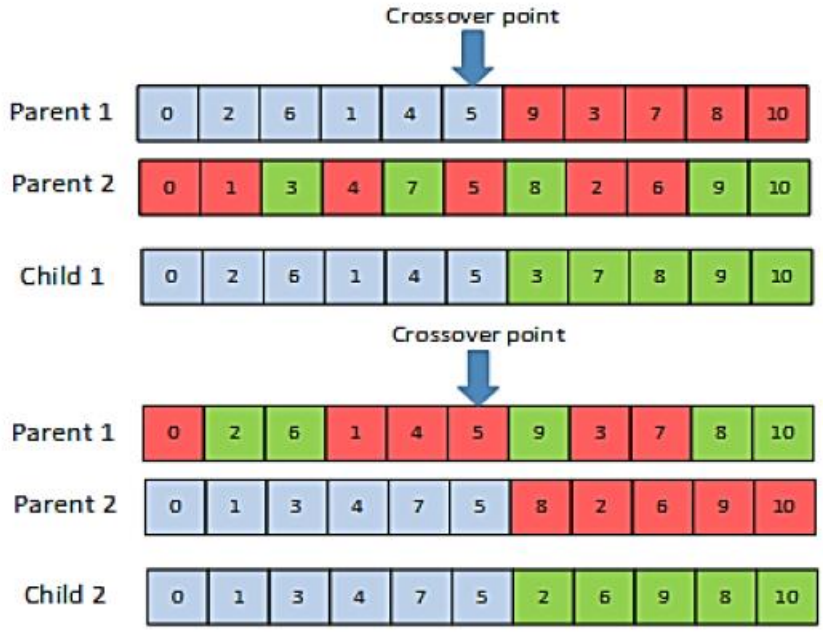


**Fig. 7**. Crossover operator in GA

### 5.5 Mutation Operator

The majority of evolutionary algorithm will get stuck in local optimum. The GA is not exempted from this event. To avoid from this phenomenon, the GA take benefit of mutation operator to create a chance for opening new room for searching unexplored search space which may have good solution. In mutation, one gen is randomly selected as $t_i$ then the first successor of this gen, subtask, is found kwon as $t_j$. Then the random gen, subtask $t_k$, where *k* belongs to

[i+1..j-1] can be selected subject to all predecessor of this subtasks are ahead of $t_i$ in the sorted list. Then, the subtasks $t_i$ and $t_k$ can be substituted. In this way, a new individual is generated. Fig. 8 is depicted to show mutation operator in GA. As Fig. 8 indicates, subtask $t_7$ in position 6th is randomly selected. The first successor of $t_7$ is $t_{10}$ in DAG. Then, the random gen, subtask $t_k$, should be selected in postion $\epsilon$ [7..10] so that all of predecessor of $t_k$ is ahead of $t_7$ in chromosome. For instance, in position 7th, the subtask $t_8$ is selected because its only predecessor, i.e. $t_5$ is ahead of $t_7$ in chromosome. Consequently, gens $t_7$ and $t_8$ can be substituted. In this phase, we take mutation rate, known as *mrate*, 20 percent of *PopSize*.

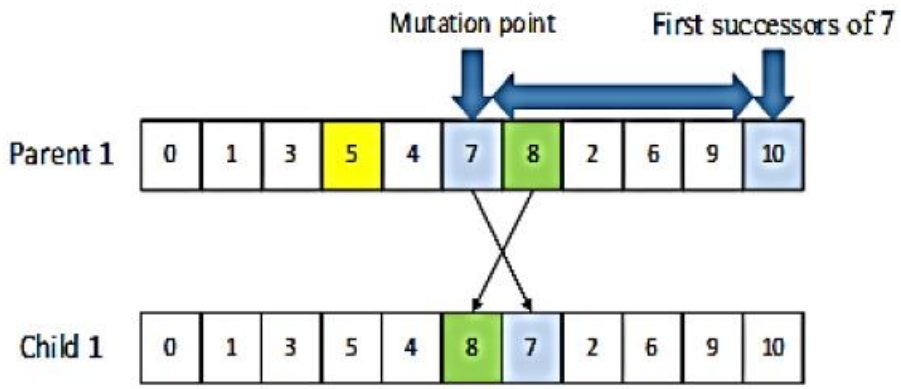


**Fig. 8**. Mutation Operator in GA

### 5.6 Termination Criteria

GA is endless algorithm unless the user cut the program which reaches to special point. Typical termination criteria in such algorithms are predetermined rounds, reaching to appropriate fitness function and etc. In this algorithm, we iterate the program 100 times. The number 100 is achieved experimentally.

## 6. Simulation and Evaluation

In this section, we applied series of experiments to evaluate the effectiveness of our proposed methodology. In this regard, we assess our work in term of total execution time, *makespan*, related to task graphs against HEFT versions and an existing approach such as Quantum genetic algorithm with rotation angle refinement for dependent task scheduling on distributed systems known as QGARAR[15].

### 6.1 Simulation Environment

All experiments were implemented using Sony VAIO laptop with a 2.26 GHz Intel Core 2 Duo processor and 4 GB RAM and using Matlab software 2015.

### 6.2 Evaluation of Methodology

To evaluate proposed algorithm, we conduct this section in two folds. Firstly, we bring an example to show the effectiveness and application of the proposed algorithm. Secondly, we define complicated scenarios to reach concrete results. At first, take a DAG depicted in Fig. 3. We run our program and compare the result with three heuristic HEFT algorithms; namely, Upward, Downward, Level ranking approaches [10] and one newest heuristic such as QGARAR [15]. Fig. 9, Fig. 10, Fig.11, Fig. 12 and Fig. 13 illustrate Upward, Downward, Level

ranking and QGARAR and our proposed approach respectively. Moreover, in these figures, the skewed lines indicate data transfer between different processors.

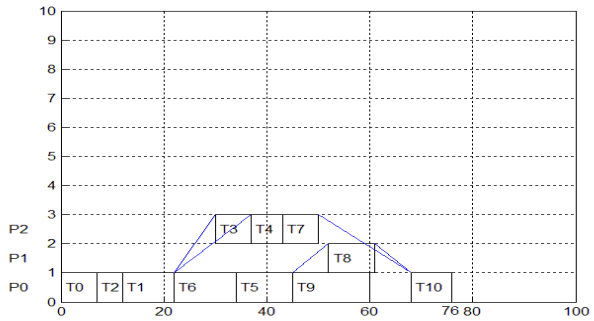


**Fig. 9**. Upward Ranking Priority [10].

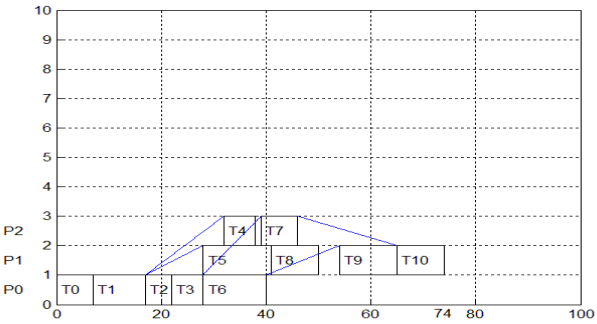


**Fig. 10**. Downward Ranking Priority [10].

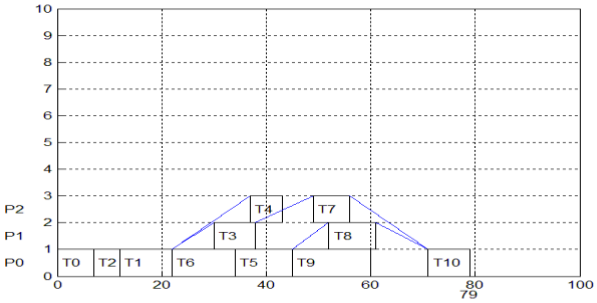


**Fig. 11**. Level Ranking Priority [10].

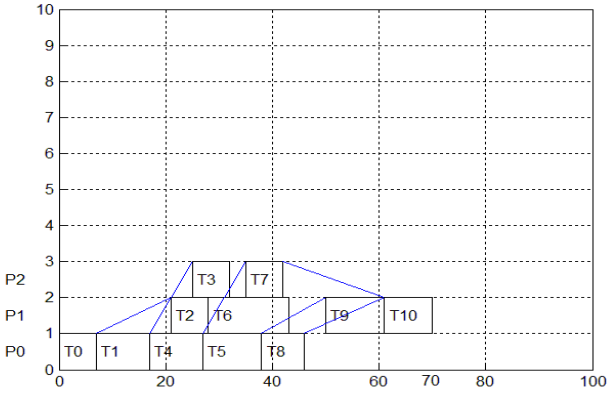


**Fig. 12**. QGARAR [15].

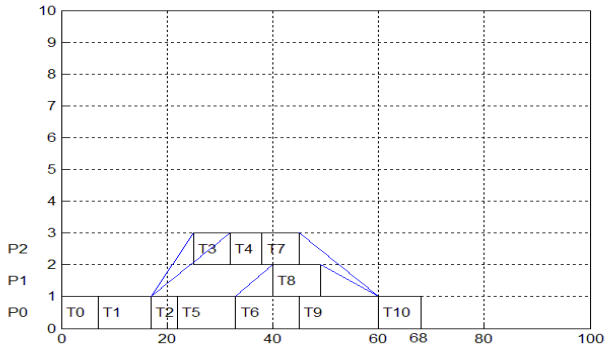


**Fig. 13**. Our Proposed Scheduling

Also, Fig. 14 illustrates comparison of total execution time, *makespan*, between approaches for DAG depicted in Fig. 3. As it depicts, our proposed method has the lowest *makespen* in comparison among others.

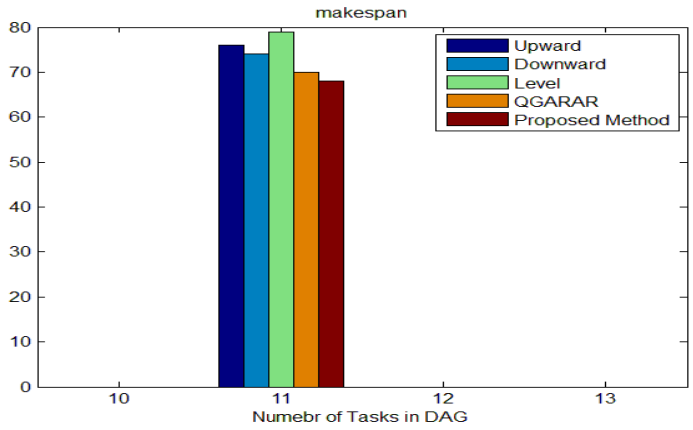


**Fig. 14**. Comparison of *makespan* between approaches for DAG depicted in Fig.3

Secondly, to reach concrete results we conduct 9 different scenarios. We define 3 types of graphs; namely, light, middle and heavy graphs with 30, 50 and 100 processing nodes each of which would run on 5, 8 and 10 heterogeneous parallel processors respectively. We take weight numbers which are produced with uniform distribution in [10..20] for processing nodes and uniform distribution weight numbers from [10..30] for communication costs between processors. Apparently, we define 9 scenarios, but for the sake of reasons which will be told, we analyze only 3 complicated scenarios, i.e., light, middle and heavy graphs on 5, 8 and 10 processors respectively. For instance, take a random light DAG with 30 nodes that has aforementioned computation and communication features. We run it on parallel processors from 2 to 10 processors by increasing one processor in each step. We have recorded *makespan* for afore-said DAG as Fig. 15 illustrates.

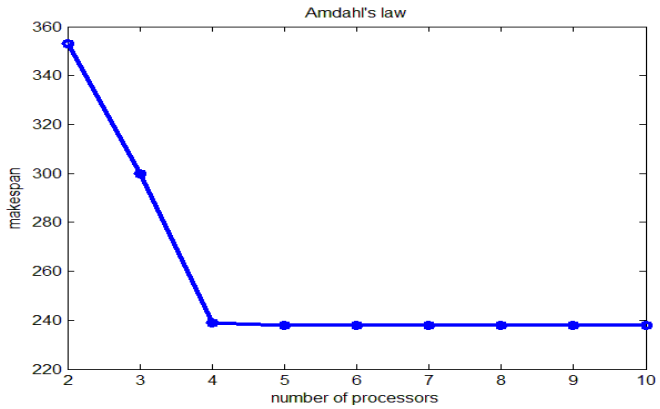


**Fig. 15.** *makespan* trajectory by increasing processors

As the figure depicts the makespan are 353, 300, 239, 238 for 2,3,4 and 5 processors respectively. It indicates if we spend more processors there is not any improvement in makespan; it is a clear-cut proof of amdahl's law because the dependecies between tasks suppress against speedup [۲۶]. In the other words, for the processor numbers $P^{*}$>5 there is no changes in *makespan*. Anyway, 3 complicated scenarios have been executed 50 times; then, the average *makespan* as a result of each approach was recorded. Fig. 16 illustrates analysis of the first scenario with light random DAGs running on 5 heterogeneous parallel processors.

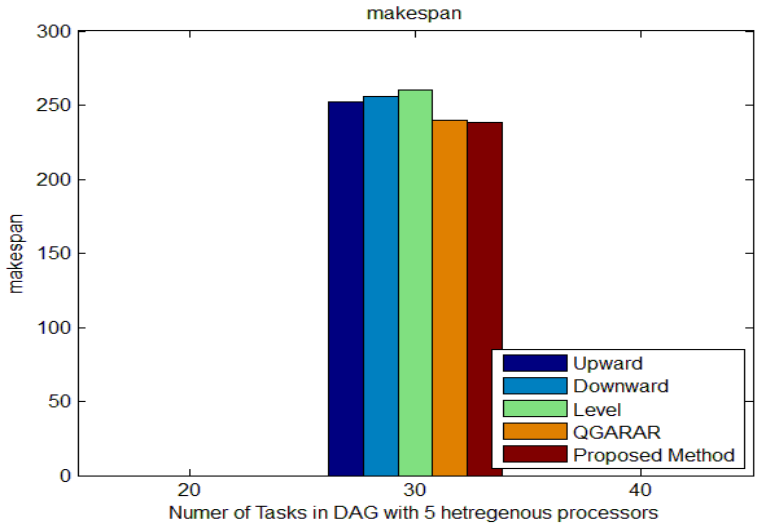


**Fig. 16**. Comparison of approaches in term of *makespan* in the first scenario

Also, Fig. 17 illustrates analysis of the second scenario with middle random DAGs running on 8 heterogeneous parallel processors.

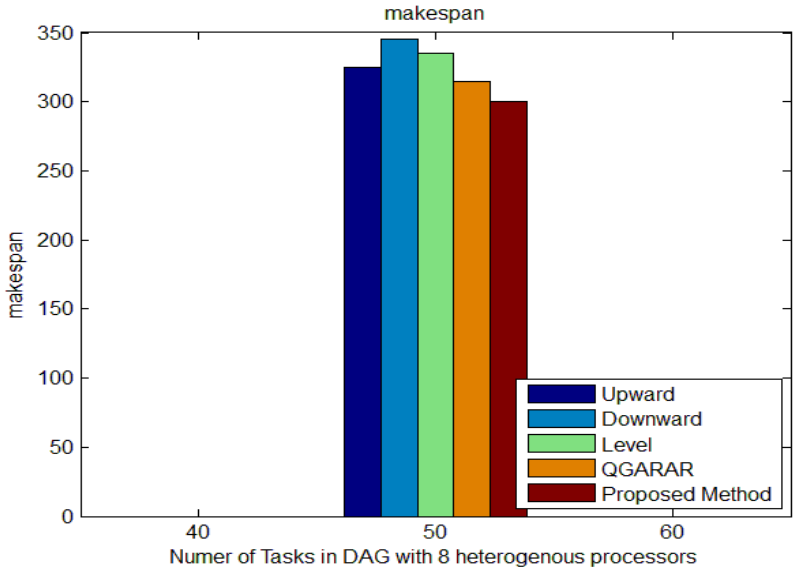


**Fig. 17**. Comparison of approaches in term of *makespan* in the secomd scenario

Finally, Fig. 18 illustrates analysis of the third scenario with heavy random DAGs running on 10 heterogeneous parallel processors.

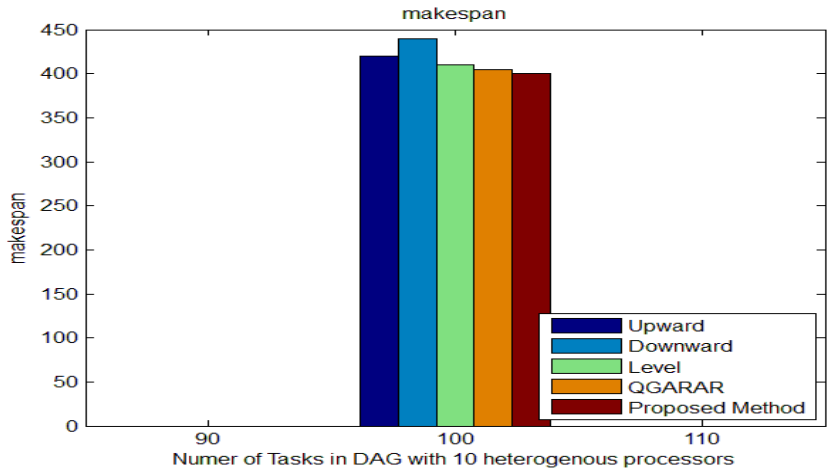


**Fig. 18**. Comparison of approaches in term of *makespan* in the third scenario

Since heuristic and evolutionary algorithms are not deterministic, but are stochastic in nature, we only analyze the outcome by several executions of different simulations. Therefore, we have conducted different scenarios to simulate. The simulation results show the high superiority dominance of proposed approach against multiple version of HEFT and marginal dominance against QGARAR; it is because of the strategies which proposed algorithm has taken. In the initial phase, it smartly constituted initial population from different heuristics to exploit efficient solutions and in the next phase it shuffled search space to explore promising ones. Although QGARAR has good treatment, our proposed approach are semi-random and

applies benefit of both exploration and exploitation in search space. It is the reason the proposed algorithm has predominance against QGARAR in all scenarios.

## 7 Conclusion and Future Work

Task scheduler module is one of the most important component in distributed systems such as in Grid- and Cloud Computing. Inefficient resource allocation owing to inefficient task scheduling does not lead the economic sense. Because the underlying utilized resource is not rationale for the tasks being executed. On the other hand, users will definitely get rid of from inefficient providers thanks to competitive open market such as in multi-cloud environment. To survive in this competitive market, it compels providers to improve response time as a key element in quality of service parameters. The good responsiveness is achievable via smart scheduler the reason why we have extended a genetic-based heuristic task scheduling algorithm in such heterogeneous systems. To have good optimality, speed and convergence, we have taken benefit from other heuristic approaches in which we smartly put multi queue in initial population; consequently, our approach has had better result in term of average *makespan* in comparison with other existing approaches. It is because of the strategies which proposed algorithm has taken. In the initial phase, it smartly constituted initial population from different heuristics to exploit efficient solutions and in the next phase it shuffled search space to explore promising ones. Although QGARAR has good treatment, our proposed approach are semi-random and applies benefit of both exploration and exploitation in search space. It is the reason the proposed algorithm has predominance against QGARAR in all scenarios. We envisage to extend a smart scheduling model which simultaneously improve both response time and resource utilization by applying dynamic voltage frequency scaling (DVFS) for tasks which do not have any time constraints. In other words, we intend to develop scheduling algorithm to find slack time of each task; by knowing time information of each task we can adjust frequency and voltage pair based on current workload.

**References**:

[1] Armbrust M, Fox A, Griffith R, D. Joseph A and Katz R, ‘‘Above the Clouds: A Berkeley View of Cloud Computing’’. Technical report EECS-2009-28, UC Berkeley, 2009.
[2] P. Mell, T. Grance, The NIST definition of cloud computing, Natl. Inst. Stand. Technol. 53 (6) (2009) 50.
[3] A.S. Tanenbaum & M.V Steen, distributed systems: principles and paradigms, second edition, prentice hall (2007).

[4] Hosseini Shirvani M, Rahmani AM, Sahafi A. An iterative mathematical decision model for cloud migration: A cost and security risk approach. *Softw Pract Exper*. 2017;1-37. https://doi.org/10.1002/spe.2528

[5] Amazon FAQ on S3. https://aws.amazon.com/s3/faqs/ [6 Jaunarary 2018].
[6] A. Burkimsher, I. Bate, L. S. Indrusiak, A survey of scheduling metrics and an improved ordering policy for list schedulers operating on workloads with dependencies and a wide variation in execution times, Future Generation Computer Systems 29 (2013) 2009–2025.
[7] T. S. Somasundaram and K. Govindarajan, "CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud," Future Generation Computer Systems, vol. 34, pp. 47-65, 5// 2014.
[8] L. Wang and L. Ai, "Task Scheduling Policy Based on Ant Colony Optimization in Cloud Computing Environment," in LISS 2012: Proceedings of 2nd International Conference on Logistics, Informatics and Service Science, Z. Zhang, R. Zhang, and J. Zhang, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 953-957.
[9] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, "Cost-efficient task scheduling for executing large programs in the cloud," Parallel Computing, vol. 39, pp. 177-188, 4// 2013.
[10] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," Information Sciences, vol. 270, pp. 255-287, 6/20/ 2014.
[11] H. Topcuoglu, S. Hariri, and W. Min-You, "Performance-effective and low-complexity task scheduling for heterogeneous computing," Parallel and Distributed Systems, IEEE Transactions on, vol. 13, pp. 260-274, 2002.
[12] M. Hosseini Shirvani , N. Amirsoleimani, S. Salimpour, A. Azab, Multi-Criteria Task Scheduling in Distributed Systems based on Fuzzy TOPSIS, 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE).
[13] A.Ghaffari and A.Kamalinia, “Hybrid Task Scheduling Method for Cloud Computing by Genetic and PSO algorithm”, Journal of Information Systems and Telecommunication, 4(4), 2016.

[14] J. Verma, S. Sobhanayak, S. Sharma, A. K. Turuk and B. Sahoo, "Bacteria foraging based task scheduling algorithm in cloud computing environment," 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, 2017, pp. 777-782.

[15] T. Gandhi, Nitin and T. Alam, "Quantum genetic algorithm with rotation angle refinement for dependent task scheduling on distributed systems," 2017 Tenth International Conference on Contemporary Computing (IC3), Noida, India, 2017, pp. 1-5.

[16] Amdahl, Gene M., "Amdahl's Law in the Multicore Era", Computer, 41 (7), 33–38, 2008.