International Journal of Communication Networks and Security

Volume 2 | Issue 2 Article 12

April 2013

AN EFFICIENT TASK SCHEDULING IN DISTRIBUTED COMPUTING SYSTEMS BY IMPROVED GENETIC ALGORITHM

K. SUNITHA
Department of Computer Science & Engineering, Mahatma Gandhi Institute of Technology, Gandipet,Hyderabad-500075., A.P., Affiliated to JNTUH, sunithak@mgit.ac.in

MRS. P V SUDHA
Department of Computer Science & Engineering, University College of Engineering (Autonomous),Osmania University , Hyderabad-500 007, sudhapv23@gmail.com

AN EFFICIENT TASK SCHEDULING IN DISTRIBUTED COMPUTING SYSTEMS BY IMPROVED GENETIC ALGORITHM

K.SUNITHA[1], MRS. P V SUDHA[2]

[1]Assistant Professor, Department of Computer Science & Engineering, Mahatma Gandhi Institute of Technology, Gandipet,Hyderabad-500075., A.P., Affiliated to JNTUH.
[2]Assistant Professor, Department of Computer Science & Engineering, University College of Engineering (Autonomous),Osmania University , Hyderabad-500 007.
E-mail: sunithak@mgit.ac.in, sudhapv23@gmail.com

Abstract - Task Scheduling problem for heterogeneous systems is concerned with arranging the various tasks to be executed on various processors of a system so that computing resources are utilized most effectively. Parallel processing refers to the concept of speeding-up the execution of a task by dividing the task into multiple fragments that can execute simultaneously, each on its own processor i.e. it is the simultaneous processing of the task on two or more processors in order to obtain faster results. It can be effectively used for tasks that involve a large number of calculations, have time constraints and can be divided into a number of smaller tasks. The scheduling problem deals with the optimal assignment of a set of tasks onto parallel multiprocessor system and orders their execution so that the total completion time is minimized. An Optimal scheduling of parallel tasks with some precedence relationship, onto a parallel machine is known to be NP-complete. This precedence relationship among tasks can be represented as Directed Acyclic Graph (DAG). In this paper, a scheduling algorithm has been proposed to schedule DAG tasks on Heterogeneous processor which uses Genetic algorithm to get optimal schedule. The scheduling problem is also considered. This study includes a search for an optimal mapping of the task and their sequence of execution and also search for an optimal configuration of the parallel system. An approach for the simultaneous optimization of all these three components of scheduling method using genetic algorithm is presented and its performance is evaluated in comparison with the Min-Min and Max-Min scheduling methods.

Keywords- *Task Scheduling, Parallel Processing, DAG, Min-Min, Max-Min, Genetic Algorithm.*

1. INTRODUCTION

Distributed Systems are very powerful and helpful computer systems that are known to solve tasks and problems in a feasible and fast way. "A distributed computing system consists of multiple autonomous processors that does not share primary memory, but cooperate by sending messages over a communications network."[1]. Distributed System is a large scale computing environment that includes many subscribed resources to perform tasks more rapidly, stability, accuracy and availability.[2]

1.1 Heterogeneous Computing System

The Heterogeneity of processors means that the processors have different speeds or processing capabilities according to the given problem. Heterogeneous computing systems refer to electronic systems that use a variety of different types of computational units. A computational unit could be a General-Purpose Processor (GPP), a special-purpose processor (i.e. Digital Signal Processor (DSP) or Graphics Processing Unit (GPU). In general, a heterogeneous computing platform consists of processors with different instruction set architectures.

The demand for increased heterogeneity in computing systems is partially due to the need for high-performance, highly reactive systems that interact with other environments (audio/video systems, control systems, networked applications, etc.).

1.2 Task Scheduling

Task scheduling in parallel environment concept is being proposed by using the Genetic Algorithm (GA) approach with Roulette Wheel Selection. A GA approach starts with a generation of individual and produces more generations in iteration mode. The aim of this paper is to present a GA which uses a novel proposed method, to decrease the computation time for finding a suboptimal schedule i.e. to minimize the makespan. However, this new method is general and could be applied to any evolutionary method having GA tool [3].

In a DCS an efficient scheduling of the tasks of a parallel program plays an important role for the improvement on system performance. A parallel program is divided into a set of tasks. Each task has its attributes such as its execution time, precedence relations and data communication times among other tasks. The tasks in the program can be represented as a DAG. A node in the DAG corresponds to a task which is a basic unit of scheduling [4].

1.3 Statement of the Problem

The complexity of the problem increases when task scheduling is to be done in a heterogeneous environment, where the processors in the network may not be identical and take different amounts of time to execute the same task. The present research proposes a new genetics-based approach to scheduling parallel tasks on heterogeneous

processors. Traditional DAG scheduling algorithms assume that the system is time-invariant, where the exact performance of the system can be known a priori. For example, it is assumed that the execution time of a task can be estimated and does not change during the course of execution. However, due to the resource sharing among multiple users in HDCS, the performance of the system can vary during the execution of the application. Under this condition, there is a need to find schedules that are less vulnerable to the variance of system performance, i.e. more robust. Minimizing the schedule length (makespan) based on the estimated system performance is not enough, since short makespan does not necessarily guarantee a small turn-around time in a real computing environment. A good schedule must also be robust. The complexity of the scheduling problem is very depended to the DAG (Directed Acyclic Graph) the number of processors, the execution time of tasks and also the performance criteria which would to be optimized.

1.4 Objectives

- To design an algorithm to schedule the DAG tasks on heterogeneous processors in such a way that minimizes the total completion time (Makespan).
- To define makespan as a measure of the throughput of the heterogeneous computing system (execution time + waiting time or idle time).
- To compare makespan for different number of processors, number of tasks, population size and number of generations to be done.

2. REVIEW OF RELATED LITERATURE

2.1 Min-Min Algorithm

Min-Min begins with a set of tasks which are all unassigned. First, it computes minimum completion time for all tasks on all resources. Secondly, the minimum value is selected among this minimum time which is the minimum time among all the tasks on any resources.

Then that task is scheduled on the resource on which it takes the minimum time and the available time of that resource is updated for all the other tasks. It is updated in this manner; suppose a task is assigned to a machine and it takes 20 seconds on the assigned machine, then the execution times of all the other tasks on this assigned machine will be increased by 20 seconds.

After this the assigned task is not considered and the same process is repeated until all the tasks are assigned resources [5].

Pseudo code for Min-Min Algorithm

1. for all tasks Ti in meta-task Mv
2. for all resources R_j
3. $C_{ij} = E_{ij} + r_j$
4. do until all tasks in Mv are mapped
5. for each task in Mv find the earliest completion time and the resource that obtains it.
6. find the task T_k with the minimum earliest completion time.
7. Assign task T_k to resource that gives the earliest completion time.
8. delete task T_k from Mv .
9. update r_j
10. update C_{ij} for all i.
11. end do.

Figure 1: The Min-Min Algorithm

2.2 Max-Min Algorithm

Max-Min is almost same as the min-min algorithm except the following: in this after finding out the completion time, the minimum execution times are found out for each and every task. Then among these minimum times the maximum value is selected which is the maximum time among all the tasks on any resources. Then that task is scheduled on the resource on which it takes the minimum time and the available time of that resource is updated for all the other tasks. The updating is done in the same manner as for the Min-Min. All the tasks are assigned resources by this procedure [6].

Max-min strategy resolves the difficulty of Min-min, by giving priority to large tasks. The Max-min algorithm selects the task with the maximum completion time and assigns it to the resource on which achieve minimum execution time. It is clear that Max-min seems better choice whenever the number of small tasks is much more than large ones. But in other cases, early executing large tasks lead for increasing in total completion time of submitted tasks so Min-min is better choice and visa-verse.

Pseudo code for Max-Min Algorithm

1. for all submitted tasks in meta-task; T_i
2. for all resources; R_j
3. $C_{ij} = E_{ij} + r_j$
4. While meta-task is not empty
5. find task Tk consumes maximum completion time.
6. assign Tk to the resource Rj which gives minimum execution time.
7. remove Tk from meta-tasks set
8. update rj for selected R_j
9. update C_{ij} for all j

Figure 2: The Max-Min Algorithm

2.3 Genetic Algorithm

Genetic algorithm is a method of scheduling in which the tasks are assigned to resources according to individual solutions (which are called schedules in context of scheduling). They report about which resource is to be assigned to which task. Genetic Algorithm is based on the biological concept of population generation [7].

A GA starts with a pool of feasible solutions (population) and a set of biologically inspired operators defined over the population itself. At each iteration, a new population of solutions is created by breeding and mutation, with the fitter solutions being more likely to procreate. According to evolutionary theories, only the most suited elements in a population are likely to survive and generate offspring, transmitting their biological inheritance to the next generation.

GAs operates through a simple cycle of stages: creation of a population strings, evaluation of each string, selection of the best strings, and reproduction to create a new population. Individuals are encoded as strings known as chromosomes composed over an alphabet.

2.3.1 Genetic Operators
Function of genetic operators is to create new search nodes based on the current population of search nodes. By combining good structures of two search nodes, it may result in an even better one. For MPS problem, the genetic operators used must enforce the intraprocessor precedence relations, as well as completeness and uniqueness of the tasks in the schedule. For MPS, certain portions of the schedule may belong to the optimal schedule. By combining several of these optimal parts, the optimal schedule can be found [8].

The main terms used in genetic algorithms are:
a. Initial Population .Initial population is the set of all the individuals that are used in the genetic algorithm to find out the optimal solution. Every solution in the population is called as an individual. And every individual is represented as a chromosome for making it suitable for the genetic operations. From the initial population the individuals are selected and some operations are applied on those to form the next generation.

b. Fitness Function .A fitness function is used to measure the quality of the individuals in the population according to the given optimization objective. The fitness function can be different for different cases. In some cases the fitness function can be based on deadline, while in other cases it can be based on budget constraints.

c. Selection. We use the proportion selection operator to determine the probability of various individuals genetic to the next generation in population. The proportional selection operator means the probability which is selected and genetic to next generation groups is proportional to the size of the individual's fitness.

d. Crossover. We uses single-point crossover operator. Single-point crossover means only one intersection is set up in the individual code, at that point part of the pair of individual chromosomes is exchanged.

e. Mutation. Mutation means that the values of some gene locus in the chromosome coding series were replaced by the other gene values in order to generate a new individual. Mutation is that negates the value at the mutate points with regard to binary coded individuals.

3. THE PROPOSED SOLUTION

3.1 Application Model
A process or an application can be broken down into a set of tasks, and we represent these tasks in the form of a directed acyclic graph (DAG).
A parallel program with n tasks can be represented by a 4-tuple (T, E, D, [A_i]).

1) T = {t_l, t_2. . . t_n} is the set of tasks.
2) E the edges, represents the communication between tasks. An edge from t_i to t_j where t_i, t_j ∈ T represents the precedence relationship between these two tasks.

3) D is an n x n matrix, where the element d_{ij} of D is the data volume which ti should transmit into t_j. This cost is assumed to be zero if tasks i and j are scheduled on the same processor.

4) A_i, 1 <= I <= n, is a vector [e_{il}, e_{i2},. . . ,e_{in},], where e_{iu}, is the execution time of ti on P_u.

If (t_i, t_j) ∈ E then t_i is called the immediate predecessor (also referred to as the parent) of t_j and t_j is called the immediate successor (also referred to as the child) of ti. Given a task t_a, its set of immediate Predecessors denoted by ipred (t_a) is defined as

$$t_j \mid (t_j, t_{a)} \in E$$

t_a is a join task, if it has two or more immediate predecessors. A node with no immediate predecessors is called an entry node.
The immediate successors of ta is denoted by isucc (t_a) and is defined as

$$t_j \mid (t_{j,} t_a) \in E$$

A node with no immediate successor is called an exit node. If the DAG does not have an entry task or an exit task, a dummy entry or exit node with zero computation cost, along with a zero communication cost to the node for which it is connected to can be set up, therefore making our algorithms applicable for DAGs of any kind. We also assume non-preemptive scheduling. That is, when a task is scheduled to a processor, it cannot be interrupted during execution [9].

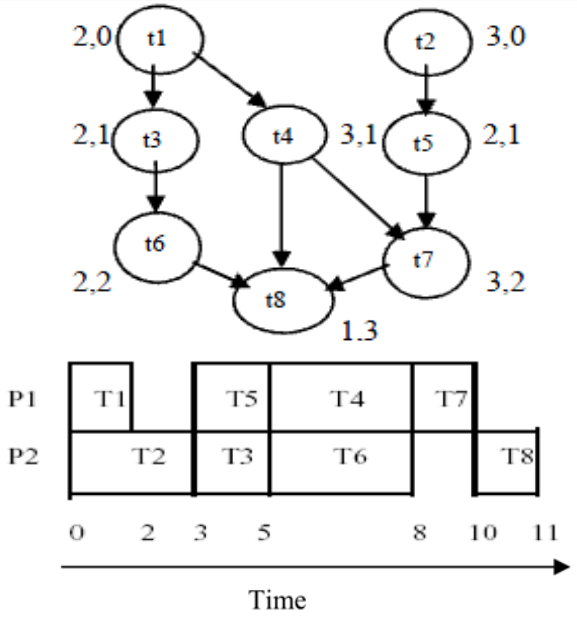

Figure 3: A Task Graph and Schedule for Two Processors Displayed as Gantt chart

3.2 Basic Assumptions

These are the assumptions that must be followed.

1. Any processor can execute the task and communicate with other machines at the same time.
2. Each processor can only execute one process at each moment.
3. Graph is fully connected.
4. Once a processor has started task execution it continues without interruption, and after completing the execution it immediately sends the output data to children tasks in parallel i.e. tasks are non-preemptive.
5. Intra-processor communication cost is negligible compared to the inter-processor communication cost.

3.3 Proposed Genetic Algorithm

3.3.1. Task selection phase

First we calculate the height of each node and store it in an array of size number of nodes in that DAG.

Height of a node can be calculated as

$$Height(t_i)=\{0,\ if PARENT(t_i)=\emptyset,\ Where\ Parent(t_i)\ is\ the\ set\ of\ parent\ tasks\ of\ t_i \quad 1+\max_{t_k \in PARENT(t_i)} height(t_k),\ Otherwise \quad (1)$$

Formula 3.1: Calculate Height of task ti in DAG task graph

Now arrange the tasks in non-decreasing order of their heights, and then this order of task is called the priority of tasks. If two tasks have same priority then chooses the task which comes first. This height function indirectly conveys precedence relations between the tasks. If the task Ti is an ancestor of task Tj, then height (Ti) < height (Tj). If there is no path between the two tasks, then there is no precedence relation between them and order of their execution can be arbitrary.

3.3.2. Task's start time and Finish time

Let T (t_i, P_u) be the time when ti receives all the required data from its parent tasks and completes its execution on Pu, Then,

$$T(t_i,p_u)=\max\{PAT(t_i,p_u),\ max\{\ t_{k\in PATENT(t_i)}\ DAT(t_i,t_k,p_u)\}+ET(t_i,p_u) \quad (2)$$

Formula 3.2: Formula for Starting and finish time calculation of a task

Where, PAT (t_i, P_u) is the (processor available) time when Pu completes the execution of all the tasks which are previously assigned to Pu, before the execution of t_i,

DAT (t_i, t_k, P_u) is the Data Available time when tk completes its execution and transmits all data which are needed to execute t_i on P_u

ET (t_i, P_u) is the execution time of task t_i on P_u.

3.3.3. Schedule Encoding (Chromosome)

A string is a candidate solution for a problem. Therefore, a string should represent a complete schedule. A string consists of several lists or 1D array. Each list is associated with a processor. The tasks in an array are linked according to their execution order on its associated processor. Each node in the list has s_i, and f_i which represent the time when task ti starts and completes its execution, respectively.

the Chromosome can be represented with a matrix of size [No. of Task x No. of Processors]

For multiprocessor scheduling problem, a legal search node (a schedule) is one that satisfies following conditions:

1. The precedence relations among the tasks are satisfied
2. Every task is present and appears only once in the schedule.

A schedule can be represented as several lists of computational tasks (fig4). Each list corresponds to computational tasks executed on a processor and order of tasks in the list indicates the order of execution.

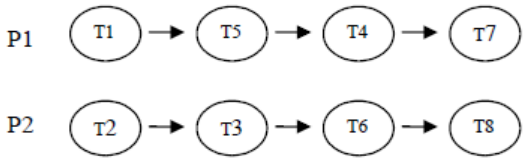

Figure 4: List Representation of a Schedule

3.3.4. Initial Population

The next step in algorithm is the creation of population. For genetic algorithm, a randomly generated initial population of search nodes is

required. We impose the following height ordering condition on schedules generated:
"The list of tasks within each processor of schedule is ordered in an ascending order of their height". For example, consider task graph in the figure1. Task T5 (height=1) is an ancestor of T8 (height=3). If they are assigned to same processor, then T5 will precede T8 according to the height ordering and this would guarantee that T5 would be executed before T8 in that processor. Similarly, T6 and T5 are not related and hence they can be executed in any order.

1. [Initialize] Compute height for every task in TG

2. Sort the tasks according to their heights in ascending order.

3. Sort the tasks with the same height according to their bottom-level in descending order.

4. Repeat step 4 and step 5 until finish of all the tasks.

5. Generate a permutation of processors.

6. Assign tasks to processors in order.

7. The above steps are repeated for the number of population size.

Figure 5: Algorithm to generate initial population: (Algorithm – Generate a schedule)

3.3.5. Fitness Function
A fitness function attaches a value to each chromosome in the population, which indicates the quality of the schedule. It comes from the evolutionary principle of "survival of the fittest", where the organisms with the best characteristics for their environment have a better chance of surviving to the next generation than weaker organisms, which are less adapted to their environment.

In the task scheduling problem, a fitness function returns the time when all tasks in a DAG complete their executions. A fitness function F(x) of a string x is defined as follows:

$$F(x) = \max_{1 \le u \le m} \{ completion\_time(P_u) \} \quad (3)$$

Formula 3.3: Fitness Calculation formula

Where completion time $(P_u) = \min_{ti \in I} T(t_i, p_u)$

Where I is the set of tasks assigned to P_u.

3.3.6. Roulette-Wheel Selection
Fitness proportionate selection, also known as roulette-wheel selection, is a genetic operator used in genetic algorithms for selecting potentially useful solutions for recombination. In fitness proportionate selection, as in all selection methods, the fitness function assigns fitness to possible solutions or chromosomes. This fitness level is used to associate a probability of selection with each individual chromosome. If fi is the fitness of individual i in the population, its probability of being selected is

$$P_i = f_i / \sum_{j=1}^{N} f_j \quad (4)$$

Formula 3.4: Formula for Roulette wheel selection

Where N is the number of individuals in the population.

3.3.7. Crossover
A crossover is a simple operation that a pair of strings randomly swaps their substrings with each other. The presence of this operator in GA is more significant. It implements the principle of evolution. New chromosome is generated with this operator by combination of two randomly selected parental chromosomes by inheriting ancestor genetic material. The crossover rate gives the probability that a pair of parents will undergo crossover. But this newly generated offspring chromosome vector may have a fitness value less than or greater than their parental chromosome. The present research paper employed one point crossover in the proposed algorithm.

The working of one point crossover is as follows:
[Step 1] first select a random processor p1 from the processor list.
[Step 2] Then select a random task t1 scheduled on that processor.
[Step 3] Select another random processor p2 form the processor list and make sure that the selected processor is not the same processor selected in step1.
[Step 4] Now find the task t2 on processor p2 which comes before t1 in priority queue.
[Step 5] Now swap the entire task which comes after t1 (including t1) on p1 with the all the tasks which comes after t2 in p2.
[Step 6] Now save the current arrangement.

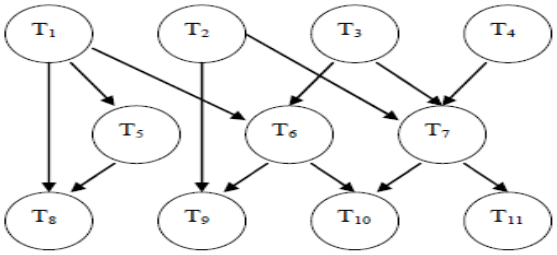

Figure 6: A DAG with 11 different tasks

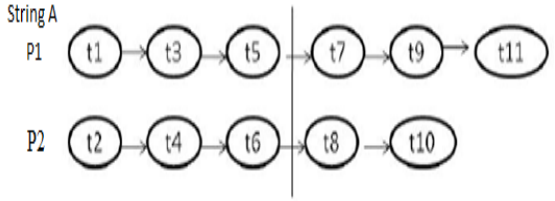

Figure 7: A String for Single point Crossover operation

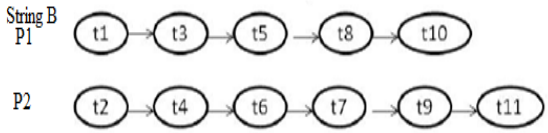

Figure 8: A String after Single point Crossover operation

3.3.8. Mutation

A mutation operation is designed to reduce the idle time of a processor waiting for the data from other processors. Let the data dominating parent (DDP) task of task ti be the task which transmits the largest volume of data to ti. That is,

$$DDP(T_i) = \{ T_k \mid \max_{tk \in PARENT(ti)} (d_{ki}) \} \quad (5)$$

Formula 3.4: Formula to find Data dominating parent (DDP) Task

Details of a mutation operation are as follows.
In a string X,
[STEP 1] Select the list of a processor say p_u, which has the latest finish time.
[STEP 2] In the selected list, find the longest idle period, and also the task t_i right
after the end of the idle period.
[STEP 3] Extract-task(X, P_u, $\{t_i\}$)
[STEP 4] Insert-task (X, P_V, $\{t_i\}$) where P_V is the processor to which DDP (t_i) is assigned.

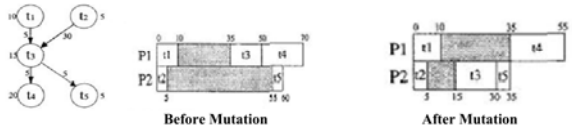

Figure 9: Example of Mutation, when n=5 and m=2

3.3.9. Termination Criterion

The GA will evolve the population until one or more stopping conditions are met. The individual with the lowest makespan is selected after each generation.
Condition 1: If we find an Individual which has Makespan less than the specified minimum, then GA stops evolving.
Condition 2: Variable gen stores the number, how many generations the GA should run. User input the variable every time he runs the program. When the generation count crosses the gen, the GA stops evolving.

3.4 Pseudo Code for Proposed Algorithm

Begin
initialize P(k); {create an initial population of size POP_SIZE}
evaluate P(k); {evaluates the fitness of all the chromosomes}
Repeat
For i=1 to POPSIZE do
Select a chromosome a as parent from population;
Child 1 <= Crossover(parent);
Child 2 <= Mutation (Child 1);
Add (new temporary population, Child 1, Child 2);
End For;
Make (new population, new temporary population, old population);
Population = new population;
While (not termination condition);
Select Best chromosome in population as solution and return it;
End

4. RESULTS AND DESCRIPTION

Default values for parameters are: No of Processors= 5, No. of Generation=100, No of Task= 26.

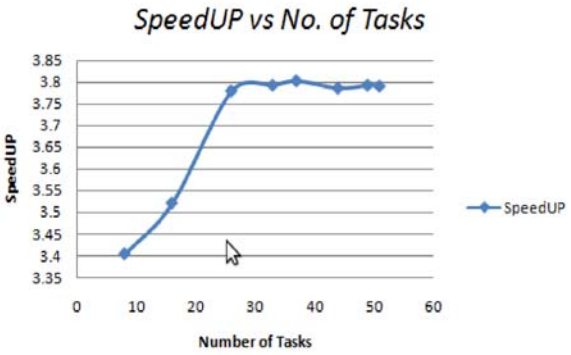

Figure 10: Speedup vs. No. of Tasks

Initially when number of tasks were less(like 10), they were unevenly distributed on 5 processors, so there might have possibility that one task scheduled on 1st processor and 2-3 tasks scheduled on 3rd processor, so there were much more differences in processors finish time. But when number of tasks increased, tasks were evenly distributed and processors finish time is approximately, equal. So speedup (serial/parallel) also stabilizes.

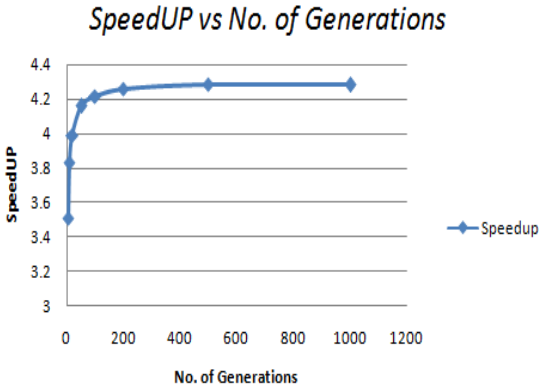

Figure 11: Speedup vs. No. of Generations

As the number of Generations increases, time taken by algorithm will also increases. After a fixed number of Generations, if we increase Generations then makespan changes very less or does not change. So we can take that fixed number to maximum generation count.

5. CONCLUSION

In this research paper, GA for the DAG-Scheduling have been presented for heterogeneous computing system. A string (Chromosome) was designed to represent a complete schedule and has the information needed for genetic operations. The complete genetic algorithm for DAG scheduling was implemented and tested on the various input task graphs in a simulated heterogeneous system. Various graphs have been generated for speedup by changing the number of tasks and number of generation cycles. There are many aspects could be considered as topics of research to introduce more accurate and improved algorithms rather that those introduced here such as the arriving rate of the tasks, cost of the task execution on each of the resource, cost of the communication, … etc. Also, applying these proposed algorithms on actual desired interested distributed system environment for practical evaluation can be other open problem in scheduling area.

REFERENCES

[1] ANDREW S. TANENBAUM, Department of Mathematics and Computer Science, Vrije Universities, Amsterdam, the Netherlands

[2] Journal of Theoretical and Applied Information Technology. (2011, April 9). [Online]. Available: http://www.jatit.org/distributed-computing/grid-vs-distributed.htm.

[3] L. Chunlin, and L. Layuan, "QoS based resource scheduling by computational economy in computational grid," Journal of Information Processing Letters, Vol. 98, pp. 119-126, 2006.

[4] Wai-Yip Chan and Chi-Kwong Li, “Scheduling Tasks in DAG to Heterogeneous Processor System”, Proceeding IEEE 1998.

[5] Pardeep Kumar*, Amandeep Verma, “Independent Task Scheduling in Cloud Computing by Improved Genetic Algorithm ", IJAR CSSE Volume 2, Issue 5, May 2012 ISSN: 2277 128X.

[6] O. M. Elzeki, M. Z. Reshad, M. A. Elsoud, “Improved Max-Min Algorithm in Cloud Computing ", International Journal of Computer Applications (0975 – 8887) Volume 50 – No.12, July 2012.

[7] Sung-Ho Woo, Sung-Bong Yang, Shin-Dug Kim, and Tack-Don Han," Task Scheduling in Distributed Computing Systems with a Genetic Algorithm", 0-8186-7901-8/97 1997 IEEE.

[8] Chris Ward, and Ben Macey “Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues”, IEEE Transactions on Parallel and Distributed Systems, Vol. 10,No. 8, pp. 795-812, Aug. 1999.

[9] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar. Users’ Guide to NetSolve V1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.