A GA Approach to Static Task Scheduling in Grid based Systems

Arun Baruah
133/4, 4th Crs, Munekolala
Marathalli, Bangalore
India
arunbaruah@gmail.com

http://www.systemprogrammers.net/

Abstract

Static task scheduling in computational grids is very important because of the optimal usage of computing time for scheduling algorithms. Given a set of resources, a static scheduler computes the execution schedule before runtime. In static task scheduling resource information and performance parameters are assumed to be known depending on how a job can be divided, relevant research can be categorized into two different areas: divisible workload, scheduling where they can divided workload into arbitraty-sizes. Solving this problem dynamically needs more time. Therefore an attempt is made to solve it by meta-heuristic techniques. A new GA scheduler, GASAScheduler is presented whose run-time depends on the number of tasks in scheduling problem. The computation time to find sub-optimal function improved. The result shows the computation time of the proposed algorithms is better.

***Keywords*:** Meta-heuristic, Static Task Scheduling, Computational Grids, SA, GA

I. Introduction

Grids consists of thousands of inter-connecting nodes which are connected to each other using the networks (LAN, WAN) using Grid interfaces like Globus, Alchemi, BONIC to name a few. The task scheduling problem in Grid based systems is known to be NP-hard since, for allocating T task to M machines, the number of allocations will be $|M|^{|T|}$ and the number of states for running will be $|T|!$ One of job of scheduling is to determine assignments of tasks to computing nodes in order to optimize the completion time for the final task in the system. If the number of task is very high, finding the optimal solution or sub-optimal task scheduling would be time-consuming. So we must use meta-heuristic algorithms based on the problem instead of using common method such as dynamic programming. Meta-heuristic algorithms prevent common errors in their own operation while trying to find the optimal solution. Therefore, they appear to be appropriate for solving problems [1]. There are many heuristic methods available for solving the static task scheduling, some of which are as follows:

Opportunistic Load Balancing (OLB) is very simple heuristic and assigns the jobs to resources, in an arbitrary manner as soon as the resources are available. Main aim of OLB is to keep all machines as busy as possible. OLB generally results in very poor make spans[2].

Minimum Execution Time (MET) works in contrast to OLB and assigns each job in an arbitrary manner to the resource with the best expected execution time for that job, regardless of the availability of the resource. MET focus on assigning each job on the best resource for it. MET tries to find good resource pairings but because it does not consider the current load on a resource, it will often cause load imbalance between the processors [2].

Minimum Completion Time (MCT) combines the benefit of both OLB and MET by assigning each job to the resource with minimum expected completion time for that job. MCT tries to avoid the circumstances in which OLB and MET perform poorly [2].

Genetic Algorithms (GA) are evolutionary techniques that are used to search for optimal solution in a very large search space. GA's are inspired by human genetics and generally works by encoding the problem in the form of chromosomes. GA operators like crossover and mutations are applied and new generations are evolved. Fitness is computed after every generations and further exploration is stopped as soon as acceptable fitness value is achieved [3-6]. Simulated Annealing (SA) is an iterative technique that considers only one possible solution (mapping) for each job at a time. This solution uses the same representation as the chromosomes for the GA. SA use a procedure that probabilistically allow poorer solutions to be accepted to attempt to obtain a better search for the solution space. The probability is based on a system temperature that decreases for each iteration. As the system temperatures "cools" it is more difficult for proper solutions to be accepted [7].

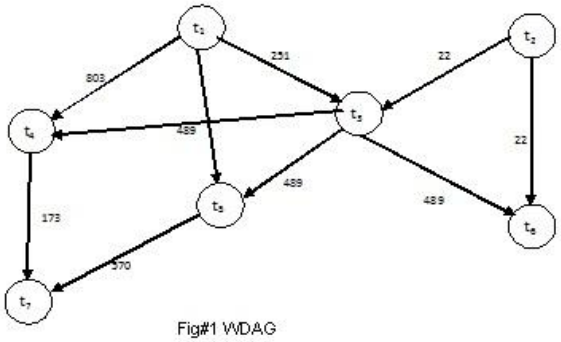
One of the best meta-heuristics methods is the Genetic Algorithm. There are many researches under the topic of solving the static task scheduling using GA's in the Grid based systems and also in distributed systems [1,3,4,9,11,12]. In this paper, a GA is presented which has a good ability to solve the above problem using the simulated annealing.

In section 2, modeling of task scheduling problem is presented. In section 3, the genetic algorithm and basic GA are introduced. In section 4, the proposed GA algorithm is introduced. In section 5, the simulation result and comparison between algorithms are presented and in section 6, Conclusion is written which concludes the findings.

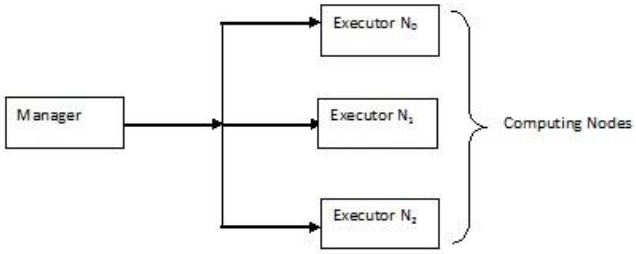
II. MODELING THE PROBLEM

A set of tasks can be modeled as Weighted Directed Acyclic Graph (WDAG) as mentioned below:

WDAG = (T, <, E, D) [11] where T = {t_i ; I = 1, … n} is a set of tasks, < is a partial order defined on T which specifies operational procedure constraints. That is, $t_i < t_j$ means that t_i must complete its task before t_j can start execution. E is a set of directed edges. A directed edges (i,j), between two tasks t_i and t_j specifies a partial order. D is an n * n matrix of communication data, where $D_{i,j}$ is the amount of data required to be transmitted from task t_i to task t_j. If grid consists of a set of m nodes which are connected to each other, then Estimated Completition Time (ECT) would be a n * n matrix, where $ECT_{i,j}$ shows the estimated completition time of the task t_i on the nodes m_j. A WDAG is shown in the figure (1) and the grid nodes consisting three nodes shown in figure (2)



Fig#1 WDAG



Fig#2. A Grid Computing of 3 nodes

R is m * n matrix which shows the data transfer rate between different nodes. If two tasks schedules on the sane node, the communication cost (ComCost) of transferring data will be zero; otherwise it is obtained based on equation (1)

$$\text{ComCost}(t_i, t_j) = \frac{Di,j}{R[N(i),N(j)]} \qquad \ldots\ldots\ldots (1)$$

$D_{i,j}$ is the amount of data required to be transmitted from task t_i to task t_j and R[N(i), N(j)] is the data transfer rate of two different nodes.

As per our mentioned concepts, the static task scheduling problem in the grid based system becomes a π : T→N mapping. This mapping allocates a set of tasks T to a set of nodes N, where the procedure constraints on the tasks is satisfied and the completion time of tasks on nodes is minimized. The problem's answer or scheduling length (SL) will be given in equation (2)

$$\text{Min}(SL = \max\{c_j \mid j=0, \ldots, m-1\}) \ldots\ldots\ldots\ldots (2)$$

C_j is the completion time of final scheduled task on nodes N_j including completion time, communication time and waiting time because of procedure constraints.

Two other parameters are defined for each nodes (tasks) in the graph known as botton-level and top-level. The bottom-level of a node is the length of the longest path from the node to a leaf node. If a node has no children, its bottom-level is equal to the average execution time of the task on the different computing machines. The top-level of a node is the length of the longest path from the node to a root node in the WDAG without considering the execution time of that task. In effect, the top-level determines the earliest beginning time of a task. Therefore, if a task has no parent its top-level will be zero.

III. THE GENETIC ALGORITHM

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some innovative flair of human search. The algorithm is as follows:

```
BEGIN
  Generate initial population;
  Compute fitness of each individual;
  REPEAT /*new generations */
      FOR  populationSize / 2 DO
          Select two parents from old generation;
          /*biased to the fittest one */
          Recombine parents for two offsprings;
          Compute fitness of offsprings;
          Insert Offspring as new generation.
      ENDFOR
   UNTIL population has converged.
END
```

These algorithms are started with a set of random solution called initial population. Each member of this population is called a chromosome. Each chromosome of this problem which consists of the string genes. The number of genes and their values in each chromosome depends on the population specification. In the algorithm of this paper, the number of genes of each chromosome is equal to the number of the nodes in the DGA and the gene values demonstrate the scheduling priority of the related task to the node, where the higher priority means that task must executed early.

Set of chromosomes in each iteration of GA is called a generation, which are evaluated by their fitness functions. The new generation i.e., the offspring's are created by applying some operators on the current generation. These are called crossover which selects two chromosomes of the current population, combines them and generates a new child (offspring), and mutation which changes randomly some gene values of chromosomes and creates a new offspring. Then, the best offspring's are selected by evolutionary select operator according to their fitness values.

The GA presented by Dhodhi et.al[11] named here Dhodhi GA (DGA) has four steps as shown below algorithms:

Step 1: Read WDAG, ECT (estimated completion time) and R values from file and get Np, Ng, Xr and Mr from the user where
Np → (initial population size),
Ng →(the number of generations),
Xr →(crossover probability),
Mr→(mutation probability)

Step 2: Calculate the botton-level and the top-level of each task in the WDAG;
Generate initial population (pi);
Pcurrent ← Pi ;

Schedules ← Decodingheuristic(Pcurrent);
BestSchedule ← evaluate(schedules);
Step 3: while stop criterion not satisfied, do begin
P_{new} ← {}; /*empty new population*/
3 – 1: repeat for (N_p/2) times
Father ← select ($P_{current}$, sum_of_fitness);
Mother ← select ($P_{current}$, sum_of_fitness);

P_{new} ← P_{new} Ù crossover (father, mother, child1, child2, X_r);

End repeat;

3 -2: for each chromosomes € P_{new} do begin

Mutate(chromosomes, M_r);

Endfor

3-3:

P_{new} ← P_{new} Ù{four best chromosomes of $P_{current}$}

$P_{current}$ ← P_{new};

Schedules ← decodingheuristic ($P_{current}$);

Best_schedule ← evaluate (schedules);

Endwhile

Step 4: Repeat the best schedule

IV. GASASCHEDULER – the proposed algorithm

The discussed algorithm (DGA) by Dhodhi et. Al [11] has used a fixed number of generations (N_g) for each WDAG with any number of tasks. This is a problem in the above algorithm because if the number of tasks is small, there is no need to tolerate high computation time of the algorithm, and if the number of tasks in WDAG is too large, it is possible that the number of generations and the number of iterations are not enough to find an optimal or sub-optimal solution. For tackling this problem, we can use a new algorithm which its running time depend on the number of tasks.

In this new idea, two main parameters of this algorithm i.e., N_p and N_g are defined as factors of task numbers. These two factors are called N_P factor and N_g factor. It is known that if the number of tasks in a WDAG is small, the computation time will be decreased because of lessening two above parameters. However, if the number of tasks is large the computation time of the algorithm will be increased. For decreasing the computing time, the simulated annealing (SA)[13] is used here. The SA algorithm is shown below:

```
BEGIN
  Initialize initial solution X, highest temperature T_h and coolest temperature T_c
  T = T_h when the temperature is higher than T_c
While not in equilibrium
Search for new solution X'
 Accept OR reject X' according to metropolis criterion
END
  Decrease the temperature T
END
```

Simulated annealing is a probabilistic search algorithm. The term simulates annealing derives from the process of heating and cooling a substance slowly to finally arrive at the solved state[13]. When the temperature is high, atoms can occasionally move to states with higher energy, but then, as temperature drops the probability

of such moves is reduced. In the task scheduling algorithm, the energy of the state corresponds to its completion time, and the temperature becomes a control parameter which is reduced during the execution of the algorithm.

For decreasing the temperature and applying the geometric cooling schedule in a proper time, a new method is used. Therefore a new algorithm GASAScheduler is introduced.

In GASAScheduler, if the convergence of results is recognized after some iteration then, some parameters of the algorithm will be changes intentionally. This happens by decreasing the number of current population size (N_p) and X_r by multiplying them to a value which is less than one and also increasing M_r by multiplying its value to a number more than one. The completion time of the new algorithm is decreased by lessening current population and crossover probability and there is an attempt to prevent to trap with local minima by increasing the mutation probability. The new algorithm is as follows:

BEGIN

Step 1: Read WDAG, ECT and R from the file and get Crossover-factor, mutation-factor, Np , Ng , Np-factor Ng-factor, slidingwindow, Xr , Mr , population-factor and Comparison-factor from the user;

Np ← Number of task * Np-factor;

Ng ← Number of task * Ng-factor;

(Fig- 3a – the modification in first step of algorithm shown in fig 2)

3-4: calculate value of CD

If(CD >= Comparision_base)

/*so, annealing happens */

Np ← Np * population-factor;

Xr ← Xr * crossover-factor;

Mr ← Mr * mutation-factor;

Reset slidingwindow to zero;

Endif

(Fig 3b – the fourth sub-step of the step three is added to the algorithm shown in fig 2)

GASAScheduler algorithm is similar to DGA however, the first step is modified as shown in fig(3a) and the fourth sub-step of the step three is added to the algorithm as shown in fig(3b).

In step one the data i.e., WDAG, ECT and R from the file and other parameters are taken as input from the user. Population- factor, crossover-factor and mutation-factor are the cooling schedules of the algorithm. The crossover and mutation operators are applied on the current generation to produce a new generation and for copying the four best chromosomes of the current generation without any changes to the new one. All the chromosomes of the new one, all of the chromosomes of new generation are decoded and the best optimal solution is stored in the Best-optimalsolution array. Therefore the stored fitness value of each element is better than the previous ones, so the Best-optimalsolution is a descending array.

For applying SA method, in the fourth sub-step, the convergence of the Best-optimalsolution element is tested by a new idea. This idea uses a sliding window which its length depends on the number of tasks in WDAG. The sliding window length (SWL) is given by the equation (3) as shown below:

SWL = number_of_task * sliding_window …….(3)

Sliding_window is a coefficient of the sliding window which its suitable value is determined latter. If the beginning of the sliding window is the index i of the Best-optimalsolution, then the convergence degree (CD) is calculated by the equation (4):

$$CD = \sum_{k=i}^{swl+1-1} \quad Best\_Optimalsolution \cdots\cdots\cdots\cdots(4)$$

swl *Best_optimalsolution

Considering the decending array Best_optimalsolution, the CD is always equal or less than one. If the value of the CD is near to one, the values of the elements are more convergent, then applying the cooling schedules will be more appropriate. The value of the CD is compared to a comparison-base; if the CD is equal or more than that, the SA will happen and the sliding window counter will be set to zero; otherwise the next iteration of the algorithm will be done. This minimum distance between two SA events is equal to the SWL. Determining the optimal parameters of the GASAScheduler: The algorithm is written in java. A set of 30 graph was taken. All elements of the matrix R is set to one. To find out the proper value of each parameter i.e., crossover-factor, mutation-factor, population-factor, comparison-base, sliding_window, the different values are assigned to each parameters, while the values of the other parameters remain fixed. Then the computation time and the overall completion time for each WDAG are calculated and accordingly, the best value for each parameter is determined as follows:

Crossover-factor =0.6, population-factor=0.8, mutation-factor = 1.1, comparison-base =0.95, sliding-window =0.25

V. Simulation and results

A set of simulation is done using the Gridsim (grid simulator) to compare GASAScheduler with the WDAG scheduling algorithm. A set of 15 random WDAG graph consists of about 100 to 200 tasks are generated randomly as shown in table (3). The matrix ECT is generated by random and all elements of the matrix R are set to one. Xr =0.6, Mr =0.05 are set for two algorithms and Np =500, Ng = 1000 are defined for DGA.

To run the simulations of two algorithms in the nearly same condition, the initial values for Np-factor and Ng-factor are set to 3 to 6 respectively, so that by multiplying their values to the number of tasks (between 100 and 200), the Np and Ng are attained at DGA range value. The other GASAScheduler parameters are equal to the same values which are obtained in section 4.

Graph number	Number of tasks	Number of nodes	Number of dependencies
1	102	5	546
2	115	10	56
3	117	7	1559
4	140	9	3785
5	142	8	9300
6	151	6	7786
7	157	5	6966
8	155	8	5846
9	158	7	3383
10	160	6	5475
11	179	6	675
12	180	6	4800
13	189	6	6517
14	199	5	5646
15	199	3	14202

Table –I WDAG used for comparison of two algorithms.

The above two algorithm executed for each graph thrice. After scheduling, the computation time and the total completion time (in seconds) of each algorithm is shown table (2) below. Also the average computation time of

GASAScheduler selected as a base and the ratio between the average computation time of DGA and the average computation time of GASAScheduler is calculated from each graph. Then by adding the obtained ratios and dividing the results by the number of graph, the average ratio is achieved. The above mentioned ratios are calculated for average and overall completion time as shown in the table (2) below:

Graph number	DGA		GASAScheduler	
	Average computation time (sec)	Overall completion time (sec)	Average computation time (sec)	Overall completion time (sec)
1	222	22514	66	22516
2	1096	1899	536	1888
3	302	33732	320	34266
4	748	58334	372	61197
5	963	230869	477	330876
6	1296	256129	493	156129
7	550	219969	477	210148
8	897	196919	460	197809
9	821	40890	508	51040
10	998	125393	499	99050
11	1242	13589	606	13808
12	1039	84444	683	94132
13	1234	123199	891	111508
14	1279	159318	1081	151198
15	1799	69789	1756	103637
Sum of ratios	26.89311465	15.10411209	15	15
Average ratios	1.79287431	1.006940806	1	1

As the results shows, the computation time of GASAScheduler is decreased by about 80% compared to DGA. However, the average total completion time is decreased.

VI. Conclusions

The task scheduling problem in the grid based system is NP-hard. Therefore the meta-heuristic algorithms which obtain near optimal solution in an acceptable interval time as preferred to the dynamic programming. The

genetic algorithm is one of the meta-heuristic algorithms which have high capacity to solve the complicated problems like the task scheduling.

In this paper, a new GA named GASAScheduler is presented which its population size and the number of generations depends on the number of tasks. The computation time of this algorithm is decreased by using SA. There is a tradeoff between the computation time and the total completion time. But with proper utilizing of SA, the computation time of the algorithm decreases more, although the overall completion time is not increased. The GASAScheduler proved highly influential in task scheduling problem.

VII. References

[1] Haupt, R.L., Haupt, S.E., Practical genetic algorithms, John willy & Sons, 2004.

[2] Armstrong, R., Hensgen, D., and Kidd, T., "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions", 7th IEEE Heterogeneous Computing Workshop (HCW '98), 1998, pp. 79-87.

[3] Ali, S., Braun, T. D., Siegel, H. J., and Maciejewski, A. A., Heterogeneous computing, in Encyclopedia of Distributed Computing, Kluwer Academic, Norwell, MA, 2001.

[4] Braun, T. D., Siegel, H. J. and Beck, N., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed systems", Journal of Parallel and Distributed Computing Vol. 61, 2001, pp. 810-837.

[5] Zafarani Moattar E., Rahmani A.M., Feizi Derakhshi M.R., "Job Scheduling in Multi Processor Architecture Using Genetic Algorithm", 4th IEEE International conference on Innovations in Information Technology, dubai, 2007, pp. 248-251.

[6] Shenassa, M. H., Mahmoodi, M., "A novel intelligent method for task scheduling in multiprocessor systems using genetic algorithm", journal of Franklin Institute, Elsevier, 2006, pp. 1-11.

[7] Pourhaji Kazem A. A., Rahmani A. M. and Habibi Aghdam H., , "A Modified Simulated Annealing Algorithm for Static Scheduling in Grid Computing", International Conference on Computer Science and Information Technology 2008 (ICCSIT 2008), Singapore August 29 – September, 2008, pp. 623-627.

[8] Rahmani A. M., Vahedi M. A., "A Novel Task Scheduling in Multiprocessor Systems with Genetic Algorithm by Using Elitism Stepping Method", INFOCOMP – Journal of Computer Science, Vol. 7(2), 2008, pp.58-64.

[9] Rahmani A. M., Moztaba Rezvani, "A Novel Genetic Algorithm for Static Task Scheduling in Distributed Systems International Journal of Computer Theory and Engineeing, Vol. 1(1), 2009, pp.1793 -8201.

[10] Abdeyazdan M. and Rahmani A. M., "Multiprocessor TaskScheduling using a new Prioritizing Genetic Algorithm based on number of Task Children", Book chapter of Distributed and Parallel Systems in Focus: Desktop Grid Computing, Springer Verlag, 2008, pp. 105-114.

[11] Lee, Y.H., Chen, C., "A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems", the 9th workshop on compiler techniques for high-performance computing, 2003.

[12] Dhodhi, M. K., Ahmad, I., Yatama, A. and Ahmad, I., "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems", Journal of Parallel and Distributed Computing, Vol. 62, 2002, pp. 1338–1361.

[13] X. Yao, "A New Simulated Annealing Algorithm", International Journal of Computer Mathematics, 56: 1995, pp 161 – 168.