Task Scheduling in Grid Computing Environment Using Compact Genetic Algorithm

Prateek kumar Singh, Neelu Sahu

Abstract— **Grid computing is a high performance computing environment to solve larger scale computational demands. Grid computing contains resource management, task scheduling, security problems, information management and soon. Task scheduling is a fundamental issue in achieving high performance in grid computing systems. However, it is a big challenge for efficient scheduling algorithm design and implementation. In this paper, a heuristic approach based on compact genetic algorithm is adopted to solving task scheduling problem in grid environment. Each individuals is represented a possible solution. This approach aims to generate an optimal schedule so as to get the minimum completion time while completing the tasks.**

***Index Terms*— Grid computing, Task scheduling, Compact Genetic Algorithm.**

I. Introduction

A computational grid is a large scale, heterogeneous collection of autonomous systems, geographically distributed and interconnected by heterogeneous networks. Job sharing (computational burden) is one of the major difficult tasks in a computational grid environment. Grid resource manager provides the functionality for discovery and publishing of resources as well as scheduling, submission and monitoring of jobs. However, computing resources are geographically distributed under different ownerships each having their own access policy, cost and various constraints. With the development of the network technology, grid computing used to solve larger scale complex problems becomes a focus technology. Task scheduling is a challenging problem in grid computing environment [8]. If large numbers of tasks are computed on the geographically distributed resources, a reasonable scheduling algorithm must be adopted in order to get the minimum completion time. So task scheduling which is one of NP-Complete. Grid computing is a term referring to the combination of computer resources from multiple administrative domains to reach a common goal. The grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. Grid computing is a high performance computing environment to solve larger scale computational demands [9]. Grid computing contains resource management, job scheduling, security problems, information management and so on. Job scheduling is a fundamental issue in achieving high performance in grid computing systems [10].

Problems become a focus by many of scholars in grid computing area. Because of, the classical algorithms are not dynamic, they cannot achieve the optimal scheduling for all situations, and therefore these algorithms cannot adapt themselves with all situations. Compact genetic algorithm has been widely used to solve this problem.

Manuscript received Dec, 2013.

***Prateek kumar Singh**, MTech (CSE), Department of Computer Science and Engineering, RGPV/LNCT Jabalpur,India*

***Neelu Sahu**,,M.E (C.T.A)Department of Computer Science and Engineering, SSCET/ CSVTU, Bhilai, India*

II. Problem Identification in Grid System

The resource in a computational grid can be anything which can be used to solve the given problem. For example a set of printers which are used for printing a set a documents. The overall objective of task scheduling is to minimize the completion time and to utilize the resources effectively and usually it is easy to get the information about the ability to process data of the available resource.[1][13].

The scheduling problem arises in a situation where there are more tasks than the available resources. Consider a scenario wherein there are x={1,2,3,4….X} tasks to be done and there are y={ 1,2,3,4…..Y} resources available. With the condition that the task is not allowed to migrate between resources. In such a situation if we have y> x then there is no reason for developing new algorithms for task scheduling because then resources can be allocated to the tasks on first come first serve basis, but if y<x then we need to develop new algorithms for task scheduling because now inefficient resource allocation can greatly hamper the efficiency and throughput of the scheduler.

To formulate the problem, define Ta ={1,2,3,….X} as x independent tasks permutation and Rb={1,2,3,Y} as y computing resources. For example a set of printers which are used for printing a set a documents. The overall objective of task scheduling is to minimize the completion time and to utilize the resources effectively and usually it is easy to get the information about the ability to process data of the available

resource [4].

Suppose that the processing time Pa,b for task 'a' computing on 'b' resource is known. The completion time P(x) represent the total cost time of completion. The objective is to find a permutation matrix y= (Ya, b), such that:

$Y_{a,b}$=1 if resource 'b' performs task 'a'.

Else

$Y_{a,b}$=0 that means number of resource are not perform number of task. Which minimize total cost:

$$\boxed{P(X) = \Sigma\Sigma\, P_{a,b} * Y_{a,b}}$$

Subject to:

$\Sigma Y_{a,b}=1;\ \forall b \in T$

$Y_{a,b} \in \{0, 1\},\ \forall a \in R;\ \forall b \in T$

The minimal *P*(*x*) represents the length of schedule whole tasks working on available resources. The scheduling constraints guarantee that each task is assigned to exactly one resource. We will discuss that a new optimal schedule is able to find the minimal completion time. To solve the task scheduling problem we have used the Compact genetic algorithm (cGA). We set an initial population by selecting a random starting sequence from the set of x! Sequences; where x is the total number of tasks. After getting the initial solution we calculate fitness value of each solution, according to equation. After that we calculate best among the entire solution and set it as an initial global best.GA update equation is used to update old population and generate new sequences and then their resources are calculated. These sequences, along with their resources are then used to find the fitness value of each individual of each solution of the Population [5]. After this if crossover criteria is satisfied, then crossover operation performed over two randomly selected parents and as a result a new sequence is generated. Then the resource of this offspring is calculated. Using the sequence and its resources the fitness value of the offspring is calculated. Based on the fitness value, if the offspring is better than its worst parent then this solution replaces that parent [12].

III. GENETIC ALGORITHM AND SCHEDULING

There are many examples in the literature of artificial intelligence techniques being applied to task scheduling [1], [8]. Meta-heuristic search techniques such as Gas, tabu and ant colony search are most applicable to the task scheduling problem because we wish to quickly search for a near optimal schedule out of all possible schedules. Good results have resulted from the use of GAs in task scheduling algorithms [3].

A GA is a meta-heuristic search technique which allows for large solution spaces to be partially searched in polynomial time, by applying evolutionary techniques from nature[7]. GAs use historical information to exploit the best solutions from previous searches, known as generations, along with random mutations to explore new regions of the solution space. In general a GA repeats three steps (selection, crossover, and random mutations) as shown by the pseudo code in Fig. 1.

Selection according to fitness (efficiency in our case) is a source of exploitation, and crossover and random mutations promote exploration. A generation of a GA contains a population of individuals, each of which correspond to a possible solution from the search space [11]. Each individual in the population is evaluated with a fitness function to produce a value which indicates the goodness of a solution. Selection takes a certain number of individuals in the population and brings them forward to the next generation. Crossover takes pairs of individuals and uses parts of each to produce new individuals. Random mutations swap parts of an individual to prevent the GA from getting caught in a local minimum.

```
initialise population

do{

crossover

random mutation

selection

}while(stopping conditions not met)

return best individual
```

Figure1: Pseudo code for genetic algorithm

IV. PROPOSED METHODOLOGY

In this paper we have proposed a solution for grid scheduling using Compact genetic algorithm with linear Crossover operator. For solving any optimization problem we have to first formulate the problem according to optimization problem.

Population Representation

To solve the problem, representation of the population and fitness value is required, so we have to first represent the grid scheduling problem in terms of CGA with linear Crossover operator. In grid scheduling we have a set of tasks and a set of resources as input and a sequence, which informs that which task is to be operated on which resource and in which order as output. CGA with Linear Crossover is based on population concept and each individual in population represents a solution,

in case of grid scheduling problem, solution is a sequence of tasks which are to be performed. So we have to first formulate each individual of CGA with Linear Crossover [5].

We represent task set as X= {t1,t2,t3….tn}and set of resources R={r1,r2,r3….rm}, Task id and resource id is given to each task and each resource so that they can be easily differentiated from one other. Such as t1 is the task id of first task and r1 is the resource id of first resource. Here x represents the total number of tasks. For e.g. we have 5 tasks which are to be performed on 3 available resources, then we have dimension value as 5.

Assuming that we have a dimension value= {4,0,3,1,2,}. Here

4 represent value for first dimension of an individual which indicates 5th task.

0 represent value for second dimension of an individual which

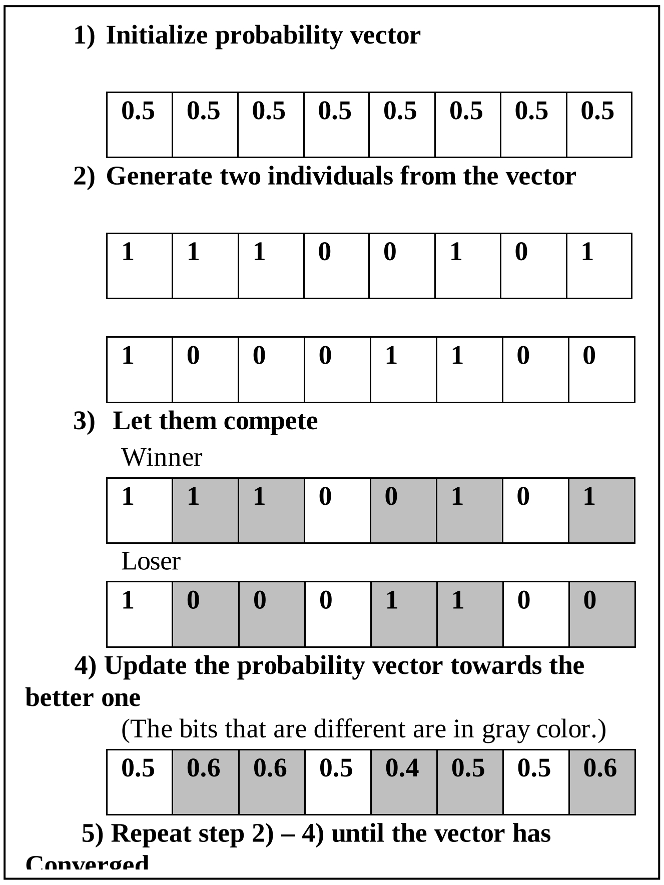
indicates 1st task.
3 represent value for third dimension of an individual which indicates 4th task.
1 represents value for fourth dimension of an individual which
indicates 2nd task.
2 represent value for fifth dimension of an individual which indicates 3rd task.
R = X mod M i.e. value of Task set mod Total resources
This formula is used to determine the associated resources for the calculated tasks in the sequence. We can calculate the resource set as {0, 2, 1, 2, 0}. From this set, we can interpret that
Task 4 is operated by resource 0,
Task 0 is operated by resource 2,
Task 3 is operated by resource 1,
Task 1 is operated by resource 2,
Task 2 is operated by resource 0,

Fitness Function

After representation of each individual we have to calculate fitness value of each individual. In case of grid scheduling problem optimal solution is the minimization the value of equation (ΣMab = 1; ∀b ϵT). Our main objective is to minimize the fitness value, an individual who have the minimum fitness value is considered as the optimal solution [3].

V. ALGORITHM: COMPACT GENETIC ALGORITHM

The Compact Genetic Algorithm (CGA) proposed by Harik, Lobo and Goldberg represents the population as a probability distribution over the set of solutions; thus, the whole population needs not to be stored. At each generation, CGA samples individuals according to the probabilities specified in the probability vector. The individuals are evaluated and the probability vector is updated towards the better individual. The CGA mimics the order-one behavior of Simple Genetic Algorithm (SGA) with uniform crossover using a small amount of memory, and achieves comparable quality with approximately the same number of fitness evaluations as the SGA. The process of the CGA is shown in figure 1. In the first step, the probability vector is initialized with 0.5. Each dimension in the vector represents the probability of each bit happened to be one [5]. Two candidate solutions are sampled from this vector. After evaluating, the winner and loser are specified. From figure 2, the winner is 11100101 and the loser is 10001100. The probability vector is updated according to the winner. The different bit between the winner and loser guides the probability to come closer to the better solution. Therefore, each dimension in the probability vector is updated toward the better solution by adding or subtracting the probability with an updating step size (1!). In a different bit, we add probability when the winner is 1, and subtract the probability when the winner is 0. E.g. updating step size is 0.1, the probability vector becomes as in step 4. The process of the CGA is repeated until the probability vector has converged. The concept of the CGA is simple and it has been proved that it performs like the SGA with population!, when the updating step size in the CGA is 1 ! [7]. the CGA reduces the size and power requirements of the system by representing the population as a probability vector rather than a collection of bit strings.



1) Initialize probability vector

0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

2) Generate two individuals from the vector

1	1	1	0	0	1	0	1

1	0	0	0	1	1	0	0

3) Let them compete

Winner

1	1	1	0	0	1	0	1

Loser

1	0	0	0	1	1	0	0

4) Update the probability vector towards the better one

(The bits that are different are in gray color.)

0.5	0.6	0.6	0.5	0.4	0.5	0.5	0.6

5) Repeat step 2) – 4) until the vector has Converged

Figure 2 the procedure of the CGA

Compact genetic algorithm with Linear Crossover

This algorithm initializes a Probability Vector (PV), where each component of the PV initialize with the parameter of 0.5, and then two solutions are randomly generated by using this PV. The generated solutions are ranked based on their fitness values. Then, the PV is updated based on these solutions. This process of adaptation continues until the PV converges. The cGA represents the population as a PV over a set of solutions with the uniform crossover [12].
At each iteration the cGA manages its population as a PV, *p* (i) = (*p*1 (i)... *pn*(i)), where *n* is the number of genes, thereby it mimics the order-one behavior of the SGA with the linear crossover. The value of *pi* (*k*) ϵ [0, 1], *i* = 1, ... , *n* , measures the proportion of the allele ”1” in the *i*th locus of the simulated population. Figure 3 describes the pseudocode of

the cGA. For *i* = 1... , *n*, *pi* (0) is initialized with 0.5 to represent a randomly generated population. In each generation (i.e. iteration), two competing solutions are generated on the basis of the current PV and then the PV is updated to favour the better solution (i.e. winner). The probability *pi*(*k*) is increased (decreased) by the learning step, *α*, when the *ith* locus of the winner has an allele of "1" (resp. "0") and the *i*th locus of the loser has an allele of "0" (resp. "1"). If both the winner and the loser have the same allele in the *i*th locus, then the probability *pi*(*k*) remains the same. Now we use compact genetic algorithm and each iteration CGA manage its population as a probability vector is PV, Probability vector is initialized with parameter 0.5 to represent a randomly generated population. In each generation (i.e. iteration), generate the individuals from the probability vector and find out the best one and then the position vector is updated to favour the better chromosome (i.e. winner). Let the best individuals are 'a' and 'b' then compete both individuals, if both individuals fitness value is same then we assign 'a' is winner and update the probability vector along the way. Clearly the best individual wins all the competition. The CGA terminates when all the probabilities converge to zero or one.

```
Compact GA (n, N, fitness)

P= allocate vector of n real number;
       For i: =1 to n
      do p [i]:= 0.5;
      t=0;
Generate two solutions from probability
vector
     a:= generate p[i]; b := generate p[i];

Compete both solutions
if (fitness (a)> fitness (b)) then
         W = a;
Else
        L = b;
       t= t+1;
(Where W is winner and L is loser)

Update the probability vector
     d=1/n;
     For i: = 1 to n do
  If( W [i] >L[i] )then
       p[i]:=p[i] +d;
else
      p[i]:= p[i] -d;

Check if the probability vector has converged.
Go to Step2, if it is not satisfied.
```

Figure 3: CGA with Linear crossover

VI. CONCLUSION

In this paper we present compact genetic algorithm, an algorithm that mimics the order one behavior of simple genetic algorithm with a given population size and selection rate, but that reduce its memory requirement. I am showing the result of each no. of task is assign by no. of resources and calculates fitness value. I will show my next paper which task is minimum cost that will directly show the value of task and resources.

ACKNOWLEDGMENT

I would like to thank my project guide for guidance and thank H.O.D sir for using Computer lab and thank Director sir for using resources and thank colleague and friends for supporting and thank the anonymous referees for their helpful comments and suggestion that have improved the quality of this manuscript.

REFERENCES

[1] Abraham, A., Buyya, R., Nath, "Natures heuristics for scheduling jobs on computational grids". In: The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000). pp. 45{52. Citeseer (2000)
[2] Edwin S.H. Hou, Nirwan Ansari, Hong Ren, "A Genetic Algorithm for Multiprocessor Scheduling", IEEE Transaction On Parallel And Distributed System, 1994, Vol.5, No.2, pp.113-120.
[3] Reza Rastegar, Arash Hariri, "A Step Forward in Studying the Compact Genetic Algorithm", 2006 by the Massachusetts Institute of Technology, Vol.14, No.3, pp.277-289.
[4] Javier Carretero, Fatos Xhafa, "Genetic Algorithm Based Scheduling for Grid Computing Systems", International Journal of Innovative Computing, Information and Control, Volume 3, Number 6, 2007.
[5] Chatchawit Aporntewan, Prabhas Chongstitvatana,—
"A Hardware Implementation of the Compact Genetic Algorithm",IEEE congress of evolutionary computation Seoul Korea, may 2001.
[6] R.Deepa, T.Srinivasan, "An Efficient Task Scheduling Technique in Heterogeneous Systems using Genetic Algorithm".
[7] Lee Wang, Howard Jay Siegel, Vwani P. Roychowdhury, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach", Journal of Parallel and Distributed Computing,1997 ,pp.8-22. [8] Shijue Zheng, Wanneng Shu, and Shangping Dai, "Task Scheduling Model Design Using Hybrid Genetic Algorithm", First International Conference on Innovative Computing, Information and Control (ICICIC'06),2006 IEEE.
[9] Vahe Aghazarian, Arash Ghorbannia, Nima Ghazanfari Motlagh, Mohsen Khajeh Naeini, "RQSG-I: An Optimized Real time Scheduling Algorithm for Tasks Allocation in Grid Environments", 2011 IEEE.
[10]Yi-Hsuan Lee and Cheng Chen, "A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems" 2003.
[11] Ahn, C. W. and Ramakrishna, "Elitism-based compact genetic algorithms", IEEE Trans. Evolutionary Computation, 7(4):367–385.
[12]George, Goldberg and lobo,The "Compact genetic algorithm", 1998IEEE.
[13]Lei, chen, jing and bo yang, "Task scheduling algorithm based on pso for grid computing", International Journal of Computational Intelligence Research. Vol.4, No.1 (2008), pp. 37–43.

Neelu Sahu received the B.E degree in computer science & engineering from the Institute of Technology, Guru Ghasidas Vishwavidyalaya, Bilaspur, India, in 2010. And M.E from the Shri Shankaracharya Group of Institutions, Bhilai, India.
Prateek kumar Singh received the B.E degree in information Technology from R.I.T.E.E college, Raipur, India, in 2012. And pursuing M.Tech from the LNCT, Jabalpur, India.