Volume 4, Issue 6, June 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Multiple Task Allocation Considering Load Based On Genetic Algorithms

Surinder Kumar*
Assistant Professor
Department of Mathematics,
SGGS College, Sector-26, Chandigarh, India

Abstract: A distributed Computing System (DCS) comprised networked heterogeneous processing requires efficient to processor allocation to achieve minimum turnaround time and highest possible throughput. Task allocation in DCS remains an important and relevant problem attracting the attention of researchers in the discipline. A good number of task allocation algorithms have been proposed in the literature [3-9]. This algorithm considered allocation of the modules of a single task to various processing nodes and aim to minimize the turnaround time of the given task. But they did not consider execution of modules belonging to various different tasks (i.e. multiple tasks). In this work we have considered the number of modules that can be accepted by individual processing nodes along with their memory capacities and arrival of multiple disjoint tasks to the DCS from time to time. In this paper, a method based on genetic algorithm is developed which is memory efficient and give an optimal solution of the problem. The given simulation results also show signification achievement in this regard.

Keywords: Distributed Computing System, Multiple Task Allocation, Load, Inter Module Communication (IMC).

I. Introduction

A distributed computing system provides the user access to various resources so that the system maintains access to shared resources to allow computation speedup and improved data availability and reliability. A task to be run on DCS consists of a set of modules (possibly defined by some partitioning activity). Each of the modules comprised the task will execute on one processor and communicate with other modules of the task. The task allocation on a DCS consists of two major steps task partitioning and its allocation [6-10]. Both problems are NP-Hard and thus various heuristics have been proposed to solve these problems [6]. In a typical DCS, it is possible that some processors are assigned more tasks than other. Therefore, it is desirable for the workload in a DCS to be eventually distributed to maximize the CPU utilization and minimize the average response time. So, load balancing algorithms try to assign the task to processors in such a way that the load on each processor is approximately the same and the turnaround time is minimum.

Genetic algorithms have been used for various optimization problems in recent years. It has recently received much attention as stochastic search algorithms for various optimization problems. Here, task allocation problem can be easily encoded as string of bits and thus is the primary candidate to make suboptimal solution with the help of the genetic algorithm [11]. In our pervious paper [7] we have discussed a genetic algorithm based the task allocation problem for single task allocation. In this paper we are considering task allocation problem for multiple tasks. Actually a DCS facilitates execution of modules belonging to various unrelated tasks. The modules of any particular task, having inter-module communication (IMC), do cooperatively execute and do not depend on the modules of the other tasks. This leads to the situation where in a processing node; it may be assigned to modules belonging to different tasks. Our proposed algorithm will allocate the modules of the different tasks when the allocator is invoked considering its load constraints and sufficient memory. This paper has been arranged as follow. Section 2 considers the allocation problem with load, the next discussion the simulation results. Finally section 5 concludes the paper.

II. The Problem

The problem of the task allocation is to map the tasks, represented by task graphs, onto the processing nodes such that it takes optimal time to produce the result [7]. Here, we have considered a forest of five tasks, each partitioned into modules, and the processor graph (PG) consists of five processors.

Now, the problem is to allocate these modules of different tasks to the processors according to their memory constraints and load in each processor.

2.1 Load:

A processors load consists of the IMC and the execution cost of each module [4]. The task allocation problem must find mapping of the set of 'm' modules of tasks to *'n'* processors that will minimize the turnaround time of a task. Task mapping or assignment to processors, is given by a matrix *M*. The following equation gives the load on processor *'P'* [13].

$$Load = \sum_{i=1}^{m}\sum_{k=1}^{m1} X_{ikp} \cdot M_{ikp} + \sum_{i=1}^{m}\sum_{k=1}^{m1}\sum_{j=1,q=1}^{m,n} C_{ijk} \cdot M_{ikp} \cdot M_{jkq} \cdot CC_{pq} \quad (2.1)$$

$$where,\ CC_{pq} = Cf_i \cdot L^i_{pq}$$

X_{ikp} = execution cost of module *'i'* of task *'k'* on processor *'p'*
M_{jkp}= if module *'m_i'* of task *'k'* is assigned to processor '*p*'
'0'otherwise
C_{ijk}= communication cost between i^{th} and j^{th} module of task '*k*'
L^i_{pq}= indicate whether two processors are *'p'* and *'q'* directly connected
Cf_i= coefficient matrix which has *'n'* entries
M_{ikq}= if module *'m_i'* of task *'k'* is assigned to processor *'q'*
'0' otherwise

The first part of the equation is the execution cost of the modules allocated to '*p*'. The second part is the communication overhead on '*p*'. M_{ikp} and M_{jkq} indicate that modules '*i'* and '*j'* of k^{th} task are allocated to two different processors (*p* and *q*), L_{pq} indicate the connectivity of the processors '*p'* and *'q*'. To allocate the modules considering load balancing of nodes, we need to compute the load on each of the *'n'* processors. Here, we assume that the matrix *C* and *X* are for the task *'k'* and for every task these matrices will be different.

III. Genetic Algorithm

A genetic algorithm emulates biological evolutionary theories to solve optimization problems. A genetic algorithm comprises a set of individual element (the population) and a set of biological inspired operators defined over the population itself. According to the evolutionary theories, only the most suited elements in a population are likely to survive and generate offspring, thus transmitting their biological heredity to new generations [14]. In computing terms, a genetic algorithm map a problem onto a set of (typically binary) strings i.e. 0s and 1s.

Each solution is associated with fitness value that reflects how good it is. The survival if the individual depends on its fitness value. The higher the fitness value, the more is the chance to survive. Genetic operation such as crossover and mutation and used to create the subsequent generation from the strings of the current population. The process is repeated until the algorithm converges [12].

To solve any problem with genetic algorithm, the problem should easily map to a population binary strings.

The fitness function in a genetic algorithm is the objective function that is to be optimized. It is used to evaluate the search nodes, thus it controls the genetic algorithm [7]. As the GA is based on the notion of the survival of the fittest, the better the fitness value the greater is the chance to survive.

The simplest form of genetic algorithm involves three types of operators: selection, crossover and mutation.

Selection: the operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more time it is likely to be selected to reproduce.

Crossover: It is generally used to exchange portions between strings. The operator randomly chooses a locus and exchanges the subsequence before and after that locus between strings. Crossover is not always affected. The invocation of the crossover depends on the probability of the crossover.

Mutation: It is used to flip the bits of the strings i.e. 1 exchanged to 0 and vice versa. Because of this, the process is also called inversion. Mutation can occur in each bit position in a string with some very small probability. Thus the structure of genetic algorithm is:

```
GA ( )
{
Initialize population
Evaluate population
Until algorithm converges do
        {
        Perform crossover and mutation
        Evaluate population
        }
}
```

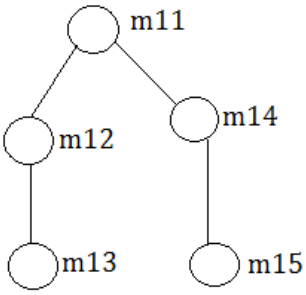
3.1. The GA for multiple task allocation:

For our problem with multiple task allocation we defined the following assumptions:

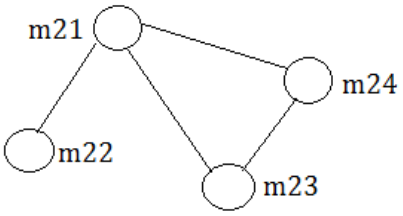
***Definition* 1**: A chromosome consists of 'n' digits, where 'n' is the total no. of modules of all the tasks to be allocated. In figure 1, the total no. of modules is 28 (m_{11}, m_{12}, $m_{13,\ldots}$, m_{58}) of the five tasks.

***Definition* 2:** A digit can take any one of the 'm' values, where 'm' is the no. of processing nodes available. If the digit is a '0', it means the corresponding module is yet to be allocated.

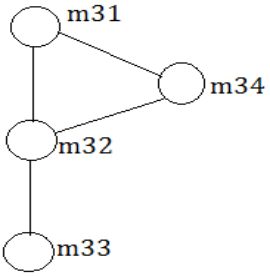
***Definition* 3**: A data structure STATUS associated with every processing node, which has two fields showing the maximum no. of the modules that can be allocated to this processor and the memory capacity of the processing node. Whenever a module is chosen for allocation onto a processing node, the STATUS is checked and it is ascertained whether the processing node can accommodate the module at hand, if not another processing node is chosen if available.



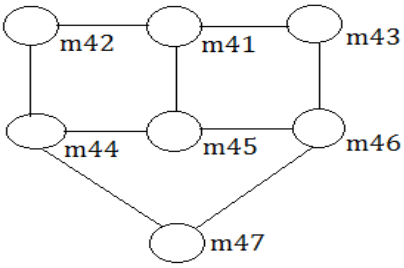
(a) Task T_1



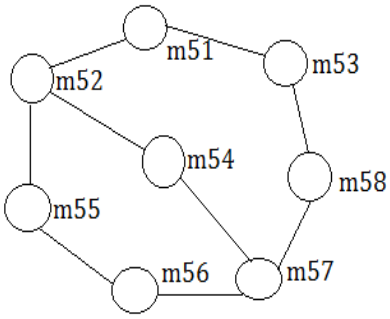
(b) Task T_2



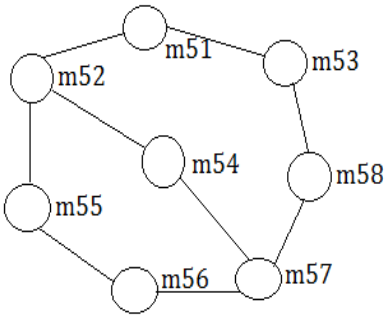
(c) Task T_3



(d) Task T_4



(e) Task T_5, Task Graphs (TG)



Processor Graph (PG)

Fig.1. Task Graphs and Processor Graph Fig. 1. (Continued)

***Definition* 4:** The fitness of a chromosome is the inverse of the cost of allocation. In our case, the inverse of the load described in equation (2.1) is the fitness function.

3.2. The algorithm for multiple task allocation:

1) Randomly generate five chromosomes, verify STATUS and take the one with maximum fitness value
   /* this fitness value is our threshold limit. Any chromosome below the threshold will be rejected and not included the population */
2) Generate an initial population of 50 chromosomes above the threshold limit.
3) SELECT: probability of selection of parents is linear dependent on the fitness value.
   /* i.e. ax+b; where 'x' is the fitness value and 'a' & 'b' are arbitrary */
4) Perform crossover and mutation with probability 'P_c' and 'P_m' at a random chosen point.
5) If total no. of chromosome (generated) < 100
   Go to
   SELECT
6) Pick up the chromosomes randomly, using the probability of selection as in SELECT.
   Take out the one with maximum fitness
   /* this represents the allocation */

3.2.1. Description of SELECT:

To effectuate the probability of selection, one method would be to store 'P_i' copies if i^{th} chromosomes, where 'P' is proportional to probability of selection of i^{th} chromosome. Thus the total number of chromosomes (copies include) will be

Then we would generate a random number between '1' and 'T' for example 'r' and select the chromosome corresponding to 'r'.

However, this method would require memory for each copy of every chromosome. To save memory we would instead attach a field with each new chromosome generated. In this field we store an integer number directly proportional to the fitness value of chromosome. Thus the chromosome represents 'X' copies of the chromosome, where 'X' is the number in its field.

When a chromosome is to be randomly selected, we generate a random number from '1' to (X_1, X_2, …,X_n), where 'X_i' is the number in the field associated with the i^{th} chromosome. Let us say, the number generated is 'Y' and $X_1+ X_2+ …+X_k<Y<X_1+X_2+…. +X_{k+1}$. Thus the chromosome selected is X_{k+1}^{th} chromosome.

Example:

Chromosome 1: fitness value=10
Chromosome 2: fitness value=12
Chromosome 3: fitness value=15
Chromosome 4: fitness value=8

Let the probability of selection be a fitness value (a^*(fitness value) +b), where 'a' and 'b' are arbitrary. i.e. X_1=10 and 'a' is assumed to be 10 and 'b' to be 0.

Let us generate 100 copies of chromosome 1 (X_1=100), 120 copies of chromosome 2 (i.e. X_2=120) … and so on. Thus the total number of chromosomes ($X_1 + X_2 + X_3 …$) = 100 + 120 + 150 + 80 = 450.

We generate a random number = 230.

Now,

$$100 + 120 < 230 < 100 + 120 + 150$$
$$X_1 \quad X_2 \qquad X_1 \quad X_2 \quad X_3$$

Hence, the 230^{th} chromosome will be a copy of chromosome 3. It is easy to see that the probability of selection in this case is proportional to the fitness value.

IV. Simulation Results

The genetic algorithm discussed in the previous section was implemented and tested on random task graphs. We have tested the algorithm on Pentium processor (100 MHz) obtained the following results of the example given below.

Given a set of five tasks with their corresponding modules

$$T_1(m_{11}, m_{12}, m_{13}, m_{14}, m_{15})$$
$$T_2(m_{21}, m_{22}, m_{23}, m_{24})$$
$$T_3(m_{31}, m_{32}, m_{33}, m_{34})$$
$$T_4(m_{41}, m_{42}, m_{43}, m_{44}, m_{45}, m_{46}, m_{47})$$
$$T_5(m_{51}, m_{52}, m_{53}, m_{54}, m_{55}, m_{56}, m_{57}, m_{58})$$

and a set of five processors (p_1 , p_2 , p_3 , p_4 , p_5) . Further, the following inputs are provided.

Adjacency coefficient for direct link = 5
Adjacency coefficient for single indirect link =10
Adjacency coefficient for double indirect link =20
STATUS [1] = {10, 30}
STATUS [2] = {7, 35}
STATUS [3] = {5, 30}
STATUS [4] = {8, 40}
STATUS [5] = {9, 35}

Table-(1)
Execution Cost of Modules of Task (T_1) on different processors.

	p_1	p_2	p_3	p_4	p_5
m_{11}	10	20	05	25	05
m_{12}	35	10	15	15	10
m_{13}	10	15	25	10	20
m_{14}	20	35	20	05	25
m_{15}	10	05	10	05	10

Table-(2)
Execution Cost of Modules of Task (T_2) on different processors.

	p_1	p_2	p_3	p_4	p_5
m_{21}	20	05	35	10	05
m_{22}	10	10	10	10	10
m_{23}	15	10	20	15	15
m_{24}	10	15	20	15	30

Table-(3)
Execution Cost of Modules of Task (T_3) on different processors.

	p_1	p_2	p_3	p_4	p_5
m_{31}	15	25	15	10	10
m_{32}	30	10	25	20	05
m_{33}	20	05	10	15	10
m_{34}	10	05	05	15	20

Table-(4)
Execution Cost of Modules of Task (T_4) on different processors.

	p_1	p_2	p_3	p_4	p_5
m_{41}	05	10	25	20	30
m_{42}	10	25	05	05	05
m_{43}	25	10	05	10	25
m_{44}	05	10	15	25	25
m_{45}	10	15	20	25	30
m_{46}	05	10	10	10	10
m_{47}	05	10	10	20	20

Table-(5)
Execution Cost of Modules of Task (T_5) on different processors.

	p_1	p_2	p_3	p_4	p_5
m_{51}	05	10	06	03	02
m_{52}	07	08	10	03	01
m_{53}	06	05	15	10	20
m_{54}	08	10	12	14	16
m_{55}	11	10	12	05	06
m_{56}	05	10	12	08	06
m_{57}	06	08	10	11	12
m_{58}	08	09	02	03	01

Table-(6)
IMC Cost of Modules of Task (T_1)

	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}
m_{11}	00	10	20	20	05
m_{12}	10	00	10	00	20
m_{13}	00	10	00	00	10
m_{14}	20	00	00	00	20
m_{15}	05	20	10	20	00

Table-(7)
IMC Cost of Modules of Task (T_2)

	m_{21}	m_{22}	m_{23}	m_{24}
m_{21}	00	05	10	10
m_{22}	05	00	00	00
m_{23}	10	00	00	05
m_{24}	10	00	05	00

Table-(8)
IMC Cost of Modules of Task (T_3)

	m_{31}	m_{32}	m_{33}	m_{34}
m_{31}	00	05	15	10
m_{32}	05	00	10	05
m_{33}	00	10	00	00
m_{34}	10	05	00	00

Table-(9)
IMC Cost of Modules of Task (T_4)

	m_{41}	m_{42}	m_{43}	m_{44}	m_{45}	m_{46}	m_{47}
m_{41}	00	05	10	15	15	15	20
m_{42}	05	00	00	10	00	00	15
m_{43}	10	00	00	00	00	05	10
m_{44}	00	10	00	00	10	15	05
m_{45}	15	00	00	10	00	05	05
m_{46}	00	00	05	00	05	00	05
m_{47}	00	00	00	05	05	05	00

Table-(10)
IMC Cost of Modules of Task (T_5)

	m_{51}	m_{52}	m_{53}	m_{54}	m_{55}	m_{56}	m_{57}	m_{58}
m_{51}	00	05	10	10	15	20	40	45
m_{52}	05	00	00	05	10	15	35	40
m_{53}	10	00	00	00	00	00	00	10
m_{54}	00	05	00	00	00	00	10	15
m_{55}	00	10	00	00	00	05	25	30
m_{56}	00	15	00	00	05	00	20	25
m_{57}	00	00	00	10	25	20	00	05
m_{58}	00	00	00	00	00	00	05	00

Table-(11)
Adjacency Matrix of Processors (L^1_{pq})

	p_1	p_2	p_3	p_4	p_5
p_1	0	1	1	1	1
p_2	1	0	0	1	0
p_3	1	0	0	1	0
p_4	1	1	1	0	1
p_5	1	0	0	1	0

Table-(12)
Memory Requirement of Modules in Units

m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	m_{21}	m_{22}	m_{23}	m_{24}	m_{31}	m_{32}	m_{33}	m_{34}	m_{41}
6	3	5	2	4	1	6	3	5	2	4	1	4	5
m_{42}	m_{43}	m_{44}	m_{45}	m_{46}	m_{47}	m_{51}	m_{52}	m_{53}	m_{54}	m_{55}	m_{56}	m_{57}	m_{58}
6	3	2	1	2	3	4	2	3	1	2	4	3	1

V. Results

a) On running the above example, we obtained the following results: The selected chromosome is 3544555514425131511111144 /* i.e. m_{11} is allocated to p_3, m_{12} to p_5, m_{13} to p_4 and so on */ time required by the program is 3 seconds.

b) In another example, considering 6 (six) processors and 8 (eight) tasks (T_1=4 modules, T_2=5, T_3=6, T_4=4, T_5=5, T_6=6, T_7=4, T_8=5) and their corresponding matrices, we obtained the following results:

Selected chromosome is 5422565533432333631452661121111536662 4 time required by program is 3 seconds.

The distributed computing systems are now expected to consist of large numbers of processing nodes with much greater number of tasks consisting of very many modules. Accordingly as evident from the above results, the time required for the GA based algorithm is almost the same for different number of tasks. The GA based algorithm will eventually promise better results for large systems with greater number of tasks. The algorithm has the complexity $O(nm^2_{max})$, where 'n' is the no. of tasks and m_{max} is the no of modules.

VI. Concluding Remarks

In this paper, we have considered the allocation of multiple tasks on heterogeneous DCS. A search method based on the genetic algorithm is proposed for this purpose. The execution and communication costs have been taken into account, which play a vital role for a task allocator in a distributed operating system. The algorithm can be easily and safely incorporated in the operating system of a distributed computing system.

References

1. C.C. Shen and W.H. Tshi, "A graph matching approach to optimal task assignment in distributed computing system using a minimal criterion," IEEE Trans. Computer 34 (1985) 197-203.
2. N. J. Nilson, Problem solving method in artificial intelligence (McGraw-Hill, New York, 1971).
3. M. Kafi and I. Ahemed, "Optimal task assignment in heterogeneous distributed computing system," IEEE Concurrency (1998) 42-51.
4. W.W.Chu and L. T. Lan, "Task allocation and precedence relations for distributed real time systems," IEEE Trans. Computers 36 (1987) 42-51.
5. P. Y.Richard Ma, E. Y. S. Lee and M. Tusuchiya, "A task allocation model for distributed computing system," IEEE Trans. Computers C-31 (1982) 41-47.
6. D.P.Yadyarthi and A. K. Tripahti,"Precedence constrained task allocation in distributed computing system," International Journal High Speed Computing 8 (1996) 47-55.
7. A.K.Tripathi, D.P.Yadyarthi and A.N.Mantri, "A genetic task allocation algorithm for distributed computing system incorporating problem specific knowledge," International Journal High Speed Computing 8 (1996) 363-370.
8. S.M.Shatz, J.P. Wang and M.Goto, "Task allocation for maximizing reliability of distributed computing system," IEEE Trans. Computer 41 (1992)11-16.
9. K. Efe, "Heuristic model of task assignment scheduling in distributed systems," IEEE Computer, June 1982, pp. 50-56.
10. C.Siva Ram Murthy, K.N.Balsubramaniyama murthy and A. Sreenivas, "Scheduling of precedence constrained parallel program task on multiprocessors," Microprocessing snd Microprogramming 36 (1992/93).
11. E.S.H.Hou, N.Ansari and H. Ren, "A genetic algorithm for multiprocessor scheduling," IEEE Trans. Parallel and Distributed Systems 5 (1994) 113.
12. M.Sriniwas and L.M.Patnaik, "Genetic algorithm: A survey," IEEE Computer (June 1994), pp.44-57.
13. J.L.R.Filho, P.C. Trelcaven and C.Alippi, "Genetic algorithm programming environments,"IEEE Computer (June 1994), pp. 29-42.

