Scheduling real-time indivisible loads with special resource allocation requirements on cluster computing

Abeer Hamdy
Electronics Research Institute
Cairo, Egypt
abeer@eri.sci.eg

ABSTRACT

The paper presents a heuristic algorithm to schedule real time indivisible loads represented as directed sequential task graph on a cluster computing. One of the cluster nodes has some special resources (denoted by special node) that may be needed by one of the indivisible loads' tasks (denoted by special task). Most previous scheduling algorithms assign the indivisible load as a single unit to one of the cluster nodes. Using this scheduling strategy may get the special node overloaded and the system may not be able to accommodate arriving loads although other nodes are unloaded. The proposed scheduler explores the task graph of the indivisible load and assigns the special task to the special node if there is enough workload to accommodate it. The other tasks are assigned to the other processing nodes subject to several predefined criteria which are satisfying the real time requirements, minimizing both of the communication cost and context switching overheads.

keywords: scheduling, allocation algorithm, cluster computing, multi processors, processing power reservations.

1. INTRODUCTION

A computer cluster is a group of loosely coupled computers connected through fast local area network and work together closely so that in many respects it can be viewed as a single computer. They have become a common alternative to large scale Symmetric Multiprocessors (SMP) because they are cost effective and scalable

Currently, high performance computing systems in general are used in several high performance applications that exhibit real-time characteristics like control systems, autonomous robots, banking systems. Scheduling a large number of real-time applications on a set of processors is a challenging problem. Real time applications are composed of one or more tasks that are dependent in most cases and are required to perform their functions under strict timing constraints. A task missing its deadline may cause other tasks to miss their deadlines resulting in a system failure. Consequently, the scheduler has to determine an assignment and execution order of these tasks on the set of processors in a way that allow the accommodation of as many applications as possible while satisfying their real time requirements.

Moreover an efficient utilization of the nodes' processing powers is also necessary to exploit the true potential of the cluster resources and efficiently utilize the processing nodes which leads to the accommodation of as many tasks as possible.

Giving these issues, this paper proposes a heuristic scheduling algorithm that uses adequate knowledge of the state of the cluster nodes and the scheduled tasks to schedule new arrived indivisible loads. Each indivisible load is represented by a directed sequential task graph. One of the tasks of the load has to be scheduled over certain node in the cluster that has special resources. These special resources may be hardware like extra memory or software like certain databases. The node with special resources is denoted by special node and the task that needs these special resources is denoted by special task. If the indivisible load is scheduled as a single unit over the special node, the special node will be overloaded and may not be able to accommodate more loads. Consequently, new arrived loads may be rejected although there are available workload over others cluster's nodes. Moreover, allocating a task workload over a processor can be guaranteed more than allocating an indivisible load with more workload requirements. So, the proposed scheduling algorithm explores the task graph of the indivisible load and assigns the special task to the special node if there is enough workload to accommodate it. The other tasks are assigned to the special node or the other processing nodes according to the workload requirements to satisfy their deadline along with other predefined criteria which are efficient usage of the processing nodes and minimizing both of the communication and context switching overheads.

The paper is organized as follows: Section 2 survey related research work. Section 3, explains the scheduling problem. Section 4 discusses the proposed scheduling algorithm. Section 5 presents an illustration example. The conclusion and future work are given in Section 6.

2. RELATED WORK

Scheduling a set of jobs for parallel execution on a set of processors is important and challenging task. This problem is known to be NP-complete even in its simplest forms [20]. Finding an optimum solution is infeasible unless some restrictions are imposed on the models representing the submitted loads and the parallel system. So, heuristic algorithms are suggested to find sub-optimal solutions. A large number of these algorithms, each of which works under different circumstances, have been proposed in literature [2-22]. Satish et al.[13] presented two scheduling algorithms based on a statistical optimization approach for scheduling a task dependence graph with variable task execution times onto a heterogeneous multiprocessor system. Jin et al. [20] presented a MILP mathematical programming formulation for static scheduling of dependent tasks onto homogeneous multiprocessor system of an arbitrary architecture with communication delays. El-Rewini [6] presented several scheduling algorithms to schedule different types of task graphs over multiprocessors environments. Oudshoornand et al. [10] presented an adaptive system to allocate tasks to the processing nodes based on the past usage statistics of each user. Ammar et al. [16] introduced an algorithm to schedule sequential task graph applications on a cluster. Some approaches allow the execution of the same task on multiple nodes if the output data is needed by multiple downstream tasks [22]. These approaches are trading additional computational power for lower communication overhead. Daviodovic et al. [21] proposed a large set of benchmark graphs for the Multiprocessor

Scheduling Problem with Communication Delays. The proposed benchmark problems have known optimal solutions and cover a broad spectrum of characteristics: multiprocessor architecture, number of processors, number of tasks, and density of inter-task communications links. They used these benchmark problem instances to analyze the performance of several constructive heuristics solutions.

Some of the artificial intelligence techniques like Genetic algorithms have drawn the attention of many researchers in parallel computing as they are known to be effective in solving such NP-hard problems. Jaewon [15] uses GA in scheduling real time tasks in multiprocessor environment. Their objectives are to minimize the number of processors required. Wu et al. [2] proposed a novel GA that uses an adaptive fitness function that gradually increases the difficulty of fitness values until a satisfactory solution is found. Moore [12] applies parallel GA to the scheduling problem and compares its accuracy with mathematically predicted expected value. More GA approaches can be found in [3,14,17,19].

However, most of these algorithms do not efficiently utilize processors' processing power. Moreover, they treat indivisible load as a single unit and do not exploit its structural features.

3. THE SCHEDULING PROBLEM

The scheduling problem consists of two models which are the cluster model, the indivisible load model. The next subsections discuss these two models in details.

3.1 The cluster model

In this paper the cluster system consists of a master node denoted by P_0 and N processing nodes denoted by $P_1,...,P_N$. All the nodes have the same computational speeds and are fully connected by homogeneous communication links. One of the processing nodes has special resources; it is denoted in this paper by "special processor". The master node doesn't participate into computation; it is responsible for admission only. It takes the decision of rejecting the load or accepting it and assigning its jobs to the different processing nodes.

3.2 The indivisible load model

In the cluster environment, a set of K real time indivisible loads {A1, A2, …, Ak}, compete for the cluster resources. Each submitted load is represented by a directed sequential task graph of n vertices. Each vertex "v" represents a real-time job (task), where v ∈ Aj = {Tj,1, Tj,2, …, Tj,n}. One of these tasks has to be allocated over the special processor; it is denoted by "special Task". Each task Tj,i of a load Aj is characterized by three parameters (Sj,i, PPj,i, Dj,i) where : Sj,i: The task's start time, PPj,i: its required processing power which indicates its estimated execution time, and Dj,i: the task's deadline. A weight is associated with each edge of the graph. This weight represents the amount of communication delay required for the result of a task Tj,i to reach its successor Tj,i+1 if both of them are allocated to different processors. The communication delay is assumed to be zero for the tasks allocated on the same processor.

It is assumed that loads randomly arrive and are submitted to the cluster once they arrive. Moreover, it is assumed that the attributes of the load's tasks are known a priori. The reason for such an assumption is to schedule all tasks of the load at once.

4. THE PROPOSED SCHEDULER

The objective of the scheduler in any distributed system is to provide the best allocation and execution order of the tasks on the system's processors according to pre-set objectives. The proposed scheduler consists of two main components, processing power reservation algorithm and the processor allocation algorithm.

The processor allocation algorithm is a heuristic search algorithm responsible for distributing the arrived tasks over the processing nodes depending on various criteria like decreasing the communication cost or balance the load over the cluster, etc. So an objective function is suggested to guide the search for the best allocation. The allocation algorithm is loaded over the master node only. On the other hand, each node in the cluster contains its own processing power reservation algorithm that manages the execution of the tasks assigned to the node. It should maximize the utilization of the available processing power of each processor to allow the allocation of as many tasks as possible, under the constraints that each task mustn't violate its required deadline. The next subsections discuss these two algorithms and the objective function in details.

4.1. Processing power reservation algorithm

The processing power reservation algorithm in this paper is a modified version to rialto operating system that was developed by Microsoft research [9]. Rialto system can schedule both real time and non-real time independent tasks. In Rialto system, the processing power reservations are made by the tasks to ensure minimum execution rate that satisfies time constraints. The request for reservation is of the form reserve x% processing power out of y % available processing power for a certain time (task's deadline). The available processing power of a processor ranges from 0 to 100%. According to this approach a task T_j is accepted if the processor can provide available processing power over T_j's deadline not less than the required. The processor maintains a data structure called a reservation table, such that all processing power reservations can be honored continuously. Each entry in the table contains the attributes of a task which are (PP_j, S_j, F_j), where, F_j is the task's finish time, $F_j = D_j - S_j$. Table (1) shows a snap shot of the reservation table of a processor between t = 115 and t = 211.

T_j	S_j	F_j	PP_j
T_1	115	135	0.2
T_2	124	156	0.1
T_3	143	172	0.3
T_4	167	211	0.4

Table1: Example for Reservation table

In the modified approach which is used in this paper, a task T_j is accepted if the processor can provide available workload during T_j's deadline not less than the required, where the required workload WL_j for T_j can be calculated by Eq.(1) :

$$WL_j = PP_j * D_j \qquad \text{………(1)}$$

A variable processing power is assigned to the allocated task T_j to satisfy its deadline instead of rejecting it if its required processing power PP_j cannot be guaranteed. Consequently, the processor may accept more tasks and produces a higher throughput. The details of this approach is presented in [1].

4.2. The objective function

Scheduling real-time indivisible load represented as sequential task graph requires allocating the tasks to the different processors of the cluster subject to the following objectives:

1. The task run on a certain processor has to achieve its deadline and to efficiently utilize the processor processing power in a way that allow the accommodation of other tasks to simultaneously share the same processor.
2. Reduce the communication cost by grouping as many tasks as possible in one bundle and allocate this bundle to a processor.
3. Reduce the context switching by trying to allocate the tasks to un-heavily loaded processors.

These objectives may cause conflicting requirements when trying to produce optimal schedules. Therefore scheduling problem is known to be NP-complete in its general form. The suggested multi objectives function consists of three weighted added terms denoted fragmentation term, context switching term and grouping term. Minimizing this function leads to a pareto optimal allocation of the indivisible load's tasks on the processing nodes that satisfies the previous requirements.

$$\psi_{i,G_{n1,n2}} = \alpha F_{i,n2-n1} + \beta \frac{1}{G_{n1,n2}} + \gamma C_{i,n2-n1} \qquad \text{….. (2)}$$

Where:

$\psi_{i,G_{n1,n2}}$: Is the objective function when allocating a group of tasks starting from T_{n1} to T_{n2} on processor P_i.

α, β, γ : are constants we set the value of each of them to 1

$F_{i,n2-n1}$: Fragmentation term

$C_{i,n2-n1}$: Context switching term

$G_{n1,n2}$: Grouping term

The fragmentation term represents the best utilization of the processor. The task (or group of tasks) is allocated on the processor with minimum available workload which is enough to accommodate the task/group and satisfy the real time requirements.

$$F_{i,n2-n1} = \sum_{k=n_1}^{n_2} \frac{(Wl_{av,i}(T_k) - Wl_{req}(T_k))}{d_k}$$

The context switching term is measured approximately by the number of active tasks within the course of deadline of the task/group. It is normalized by dividing its value by the number of active indivisible loads within the task/group's deadline.

$$C_{i,n2-n1} = \frac{\sum_{k=n_1}^{n_2} avg.\ nu.of\ active\ tasks\ within\ T_k\ deadLine}{nu.of\ active\ indivisible\ loads}$$

The grouping term is simply represented by the group size. As the group size increases the communication cost reduces, assuming equal communication delay between the tasks.

$$G_{n_1,n_2} = n_2 - n_1 + 1$$

The three terms are normalized such that the maximum value of each of the three terms equal to the maximum number of tasks in any indivisible load.

4.3. The allocation algorithm

The proposed heuristic allocation algorithm proceeds as follows:

1. Calculate the required work load WL_{sp} for the special task T_{sp} using Eq. (1)

2. The allocation algorithm provokes the processing power reservation algorithm of the special node P_{sp} to get the available workload $WL_{sp,i}$ on this node during the course of T_{sp}'s deadline.

3. *If* $WL_{sp,i} < WL_{sp}$ the special task cannot be allocated on the special processor and consequently the whole load is rejected
else the allocation algorithm proceeds to check the acceptance of the load.

4. Calculate the required work load WL_j for each task T_j (excluding T_{sp}) of the indivisible load using Eq. (1)

5. The allocation algorithm provokes the processing power reservation algorithm of all the nodes such that each node provides its reservation table to the allocation algorithm. Consequently, the available workload over each processor during the course of the deadline of each task T_j can be determined.

6. The algorithm checks the acceptance of each task (excluding T_{sp}) of the load on each processor to determine for each task the candidate set of processors that can accept this task ζ_j. A processor Pi is considered candidate for task T_j if it has available workload enough to execute that task within its required deadline.

i.e. $Pi \in \zeta_j$ if $WL_{av,i} \geq WL_j$

Since the tasks are represented by a sequential task graph, allocating a task on a processor will not affect the acceptance of other tasks as they reserve different time slots. If each task of the indivisible load has a nonempty available processor set (i.e. $\zeta_j \neq \phi$), then the indivisible load is accepted. Otherwise it is rejected. If the indivisible load is accepted, the allocation algorithm starts looking for the best allocation of the tasks over the processing nodes that minimizes the objective function.

7. The heuristic algorithm starts with the first task in the indivisible load A_i and considers groups of sizes (l=1, 2,…, n_i) where n_i is the number of sequential tasks of the load A_i. For each group ($G_{1,1}, G_{1,2}$, …, G_{1,n_i}) the algorithm finds the set of candidate processors to which the group can be assigned $\zeta_{G_{1,L}}$. This set is the intersection of the available processor sets of the tasks constituting the group. Note that:

a) If one of the sets is empty this means that we can't group these tasks into one bundle and assign this bundle to a single processor.

i.e. if $\zeta_{G_{1,L}} = \phi$ this grouping is not feasible.

b) For any group includes the special task its set of candidate processors includes only the special processor.

8. For each group $G_{1,L}$ the algorithm computes the value of the objective function $\psi_{i,G_{1,L}}$ when assigning this group to each candidate processor $Pi \in \zeta_{G_{1,L}}$ using Eq.(2). The algorithm considers the processor which results in minimum objective function to be the only candidate for this group. Among all possible groups ($G_{1,1}, G_{1,2}$, …, G_{1,n_i}) the algorithm picks up the group with the minimum value of the objective function on its candidate processor. This group and its processor constitute part of the required scheduling scheme. For example if the algorithm picked up $G_{1,3}$ and its candidate processor is P_5, this means that the algorithm will assign $\{T_1, T_2, T_3\}$ to P_5.

9. The allocation algorithm repeats steps 7&8 considering groups that start at task T_4 with sizes (l=1, 2, …, n_i-4+1). Next, the algorithm proceeds until all the tasks of the indivisible loads are allocated on the cluster processors.

The computational complexity of this allocation algorithm is $O(N * K * n_i^2)$, where, n_i the number of tasks of the indivisible load, N is the number of processing nodes, and K is the number of indivisible loads.

Algorithm 1: summarizes the steps of the allocation algorithm

---

Algorithm 1: indivisible load allocation algorithm

Input: a set of randomly arrived indivisible loads, one of the tasks of each load is a special task

Output: Scheduling scheme, acceptance rate

acceptance_counter = 0

While (the loads arrival queue is not empty)
begin
  Pick up a load A_i
  // Check the acceptance of the special task T_{sp} on the special node P_{sp}
  If $WL_{av,sp} < WL_{sp}$, T_{sp} is rejected & A_i is rejected
  Else {
   // check the acceptance of A_i
   For each task T_j, where j = {1,2,…. n_i}
    Check the acceptance of T_j on all processors and determine its available processor set ζ_j
    if $\zeta_j \neq \phi$ $\forall$ j= {1,2,… n_i} then
     A_i is accepted
   }
   If (A_i is accepted)
    Increment acceptance_counter
    S = 1
   While (S < n_i**) {**
    Form all possible groups of tasks that start with task T_S and ends with task T_L, where L= {S, S+1,…. n_i}
    Find the candidate processors $\zeta_{G_{S,L}}$ for each $G_{S,l}$
    $\zeta_{G_{S,L}} = \zeta_S \cap \zeta_{S+1} \cap ..... \cap \zeta_l$
    Compute $\psi_{i,G_{S,L}}$ for each group $G_{S,L}$ on each candidate processor $Pi \in \zeta_{G_{S,L}}$
    Pick up the group with the minimum objective function $G_{S,k}$, S≤k ≤L
    Add this group and its assigned processor to the scheduling scheme
    S=S+ size of $G_{S,k}$
   }
  Allocate the groups on the assigned processors
End

---

5. ILLUSTRATION EXAMPLE

This example shows how our algorithm is used to schedule an indivisible load consists of a set of 5 sequential jobs. Assume that the indivisible load attributes is given by Table 2 and T_4 is the special task. Assume that the cluster have eight processors $P_0 \ldots P_7$; where, P_0 is the master and is not included in the scheduling. Assume P_5 is the special processor. The values of the available workloads on the processors over the course of tasks' deadlines are assumed for illustration purpose only and are given in Table 3.

	S	F	D	PP	WL
T_1	100	150	50	0.7	**35**
T_2	150	270	120	0.5	**60**
T_3	270	350	80	0.3	**24**
T_4	350	410	60	0.4	**24**
T_5	410	500	90	0.6	**54**

Table 2: Attributes of the tasks of the indivisible load

Ava. WL	T_1	T_2	T_3	T_4	T_5
P_1	40	55	35	30	24
P_2	25	15	22	44	28
P_3	60	73	46	17	33
P_4	55	40	53	29	57
P_5	70	120	33	27	60
P_6	23	100	12	19	65
P_7	45	37	10	18	32

Table 3: Available workload on the Processors of the cluster over the course of deadline of each task

	T_1	T_2	T_3	T_4	T_5
P_1	●		●		
P_2					
P_3	●	●	●		
P_4	●		●		●
P_5	●	●	●	●	●
P_6		●			●
P_7	●				

Table 4: Available processor set for each task

In Table 4, we determined the available processor set for each task by comparing its required workload (Table 2) to the available workload on each processor (Table 3) during the course of its deadline. Table 5, lists all groups that start at task T_1 and determines the candidate processor list of each group using the intersection of the available processor sets (Table 4) of the tasks constituting this group, taking into consideration that any group includes the special task T_4 has to be scheduled over the special processor P_5. In Table 6, we computed the values of the objective function for all possible groups on their candidate processor. As can be seen in the table, the minimum value of the objective function is achieved when we assign $G_{1,3}$ (T_1, T_2, T_3) to P_3.

	$G_{1,1}$	$G_{1,2}$	$G_{1,3}$	$G_{1,4}$	$G_{1,5}$
P_1	●				
P_2					
P_3	●	●	●		
P_4	●				
P_5	●	●	●	●	●
P_6					
P_7	●				

Table 5: Candidate processors for each group starts at T_1

Obj. fn	$G_{1,1}$	$G_{1,2}$	**$G_{1,3}$**	$G_{1,4}$	$G_{1,5}$
P_1	41.67				
P_2					
P_3	120	152.11	**30.31**		
P_4	70.86				
P_5	85.93	128.97	46.23	67.78	80.2
P_6					
P_7	41.67				

Table 6: Values of objective functions when assigning the groups to their candidate processors

Scheduler proceeds to allocate the tasks starting from the task number T4. We don't need to determine the candidate processors for the groups starting from T4 because any group has to be assigned to p5 so we compute the objective function of the possible groups starting from T4 that can be assigned to p5 as shown in table 7. The results shown in the table suggests that, $G_{4,5}$ should be assigned to P_5.

Obj. fn	$G_{4,4}$	**$G_{4,5}$**
P_1		
P_2		
P_3		
P_4		
P_5	65.45	**28.25**
P_6		
P_7		

Table 7: Values of objective functions when assigning the groups start at the special task T_4 on the special processor P_5

The scheduling solution (in Tables 6 and 7) decides that (T_1, T_2, T_3) should be assigned to P_3 and the special task T_4 and task T_5 should be assigned to the special processor P_5 as shown in figure 1.

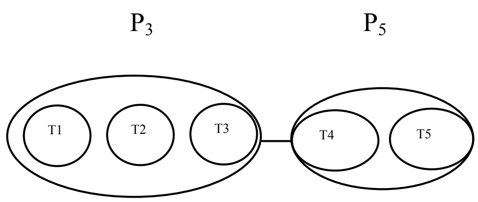

Figure 1: Feasible solution

6. CONCLUSIONS

The paper has introduced a heuristic algorithm to schedule real time indivisible loads represented as sequential task graphs on a cluster computing. One of the processing nodes has special resources that may be needed by one of the jobs of each load. The objectives of our scheduler are to achieve loads deadlines, decrease overheads by decreasing both of the communication cost and context switching, and increase the system's throughput. Simulation experiments have been conducted to test the performance of the proposed algorithm compared to single unit scheduling strategy and it is found that the proposed strategy is superior to the single unit scheduling strategy in terms of the acceptance rate. These results are not included here due to paper limitations.

Currently, another set of simulation experiments are being conducted considering some variations in both of the cluster and load models. Assuming that more than one node in the cluster have different special resources, which may be needed by more than one task in the load. Moreover, a cluster with heterogeneous processing power nodes and unlimited connectivity will be considered.

REFERENCES

[1] A. hamdy, A. Hussien and R. Amar, “An efficient workload allocation to improve scheduling real-time tasks”, ”, IEEE Symposium on Computers and Communications ISCC 06, Pula-Cagliari, Sardinia, Italy June 2006.

[2] A. Wu, H. Yu, S. Jin, K. Lin, G. Schiavone , “An incremental genetic algorithm approach to multiprocessor scheduling”, IEEE Trans Parallel Distrib Syst 15(9):824–834, 2004.

[3] A. Auyeung , I Gondra , H Dai , “Evolutionary computing and optimization: Multi-heuristic list scheduling genetic algorithm for task scheduling”, Proceedings of the ACM symposium on applied computing, pp 721–724, 2003.

[4] D. Liong, J. Deogun, S. Goddard, “Feedback scheduling of real-time divisible loads in clusters”, special issue on the 14th IEEE real-time and embedded technology and indivisible loads symposium (RTAS’08), july 2008.

[5] D. Yu, T. G. Robertazzi, “Divisible load scheduling for grid computing”, in PDCS’2003, 15th Int’l Conf. Parallel and Distributed Computing and Systems, 2003.

[6] H. El-Rewini and M. Abd-El-Barr, Advanced Computer Architecture and Parallel Processing, John Wiley and Sons, 2005.

[7] H. Topcuoglu, S. Hariri, and M.Y. Wu, “Performance effective and low complexity task scheduling for heterogeneous computing”, IEEE transactions on parallel and distributed systems, vol. 13, pp.260-74, 2002.

[8] K. Li and Y. Pan, “Probabilistic analysis of scheduling precedence constrained parallel tasks on multi computers with contiguous processor allocation”, IEEE transactions on computers, vol.49, pp.1021-30, 2000.

[9] M. B. Jones, J. Regehr, and S. Saroiu, “Two Case Studies in Predictable application Scheduling Using Rialto/NT”, In Proceedings of the 7th Real-Time Technology and Indivisible loads Symposium (RTAS 2001), Taipei, Taiwan, May 30-June 1, 2001.

[10] M. J. Oudshoornand, H. Lin, “Evolving toward an optimal scheduling solution through adaptive” Journal of parallel and distributed computing, vol.62, issue,7, pp.1203-1222, july 2002.

[11] M. Albert, Real-Time Systems : Scheduling, Analysis, and Verification. Wiley-Interscience, August 2002.

[12] M. Moore, “An accurate parallel genetic algorithm to schedule tasks on a cluster”, Parallel Computing 30:567–583, 2004.

[13] N. R. Satish , K. Ravindran, K. Keutzer , “Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors”, Proceedings of the 7th ACM international conference on Embedded software table of contents Atlanta, GA, USA , 2008.

[14] O. Ceyda , M. Ercan, “A genetic algorithm for multilayer multiprocessor task scheduling”, In: TENCON 2004, IEEE region 10 conference, vol 2, pp 68–170, 2004.

[15] Oh. Jaewon and Wu. Chisu, “Genetic algorithm based real time task scheduling with multiple goals”, Journal of systems and software, vol. 71, issue 3, pp. 245-258, 2004.

[16] R. Ammar, A. Alhamdan, and A. El-Dessouky, “Real-Time scheduling of tandem tasks preceding graphs on grid computing”, In IEEE ISSPIT, Cairo, Egypt, 2001.

[17] S. Cheng , Y. Huang , “Scheduling multi-processor tasks with resource and timing constraints using genetic algorithm”, In 2003 IEEE international symposium on computational intelligence in robotics and automation, vol 2, pp 624–629,2003.

[18] S.K. Baruah and J. Goossens, “Rate-monotonic scheduling on uniform multiprocessors”, IEEE transactions on computers, vol.52, pp.966-70, 2003.

[19] S. T. Cheng and S.-I Hwang, “Optimal real-time scheduling with minimal rejections and minimal finishing time”, Real-Time systems, vol. 20, pp. 229-53, 2001.

[20] S. Jin, G. Schiavone, D. Turgut, “A performance study of multiprocessor task scheduling algorithms”, The Journal of Supercomputing , Volume 43 , Issue 1 ,January 2008.

[21]T. Davidović , T. G. Crainic, “Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems”, Computers and Operations Research, v.33 n.8, p.2155-2177, August 2006.

[22]W. Yao , J. You , B. Li , “Main sequences genetic scheduling for multiprocessor systems using task duplication”, Microprocess Microsyst 28:85–94, 2004.