Two Round Scheduling (TRS) Scheme for Linearly Extensible Multiprocessor Systems

Abdus Samad
University Women's Polytechnic
Faculty of Engg. & Tech.
Aligarh Muslim University, Aligarh
India

M. Qasim Rafiq
Dept. of Computer Engineering
Faculty of Engg. & Tech.
Aligarh Muslim University, Aligarh
India

Omar Farooq
Dept. of Electronics Engineering
Faculty of Engg. & Tech.
Aligarh Muslim University, Aligarh
India

ABSTRACT
Balancing the computational load over multiprocessor networks is an important problem in massively parallel systems. The key advantage of such systems is to allow concurrent execution of workload characterized by computation units known as processes or tasks. The scheduling problem is to maintain a balanced execution of all the tasks among the various available processors (nodes) in a multiprocessor network. This paper studies the scheduling of tasks on a pool of identical nodes which are connected through some interconnection network. A novel dynamic scheduling scheme named as Two Round Scheduling (TRS) scheme has been proposed and implemented for scheduling the load on various multiprocessor interconnection networks. In particular, the performance of the proposed scheme is evaluated for linearly extensible multiprocessor systems, however, a comparison is also made with other standard existing multiprocessor systems. The TRS operates in two steps to make the network fully balanced. The performance of this scheme is evaluated in terms of the performance index called Load Imbalance Factor (LIF), which represents the deviation of load among processors and the balancing time for different types of loads. The comparative simulation study shows that the proposed TRS scheme gives better performance in terms of task scheduling on various linearly extensible multiprocessor networks for both uniform and non-uniform types of loads.

General Terms
Parallel and Distributed Systems, Scheduling & Load Balancing.

Keywords
Dynamic Scheduling, Multiprocessor, Interconnection Network, Tasks, Two Round Scheme.

1. INTRODUCTION
The efficient management of parallelism on an interconnection network involves optimizing conflicting performance indices, like the minimization of communication and scheduling overheads and uniform distribution of load among the nodes. In such a system more than one nodes process the various jobs concurrently. Each job may consist of various tasks that could be executed independently. The number of tasks allocated to each processor has to be controlled in such a way that a high speed execution of processes may occur while maintaining high processor utilization. In such a system, if some nodes remain idle while others are extremely busy, system performance will be degraded drastically. Therefore, scheduling of tasks becomes an important problem for multiprocessor system architectures and consequently it has a substantial effect on the system performance and utilization. It is required that all the processors should share the load evenly that would lead to complete the job in minimum possible time

Scheduling may be performed at the local level or global level based on the information they use to make load balancing decisions [1]. In the global schemes, the scheduling decision is made using global knowledge: i.e. all the processors take part in the synchronization and send their performance profiles to the scheduler. Scheduling algorithms can be classified as either static or dynamic. The static algorithm performs by a predetermined policy, whereas, the dynamic algorithm makes its decision at run time according to the status of the system [2], [3].

The important parameter when dynamic scheduling algorithms are implemented on a parallel system is the configuration of the interconnection network. The parallel system generally uses a regular point-to-point interconnection network, instead of a random network configuration. Over the years, many different interconnection networks have been used in commercially available concurrent systems and numerous research prototypes have been proposed and evaluated in the literature [4], [5], [6]. Prime examples are found in ring network, hypercube, debruijn network, Linearly Extensible Tree (LET) network, Linearly Extensible Cube (LEC) network, star graphs [7], [8], [9], [10]. The choice of the topology of the interconnection network is critical in the design of massively parallel computer systems. Interconnection networks may be categorized into two major groups on the basis of their complexity and scalability. The first category includes high complex networks because of their exponential expension and hence posses poor scalability [11], [12], [13]. Some examples are hypercube, Twisted hypercube, de_Bruijn networks etc. The second category of multiprocessor systems is of Linearly Extensible Networks, which are lesser complex. These networks are highly scalable networks i.e. the size of the system (e.g., the number of nodes) can be increased with minor or no change in the existing configuration. These include LET, LEC and Tree type networks. In this paper two linearly extensible multiprocessor interconnection networks having similar topological properties are considered for the purpose of simulation (Fig. 1 to Fig. 2). In addition the performance is also evaluated for standared hypercube architecture (Fig. 3) and a comparative study is made.The important properties [7], [8] of these interconnection networks are given in Table 1.

The rest of the paper is organized into five sections. Section 1 is the introduction. Section 2 is an overview of the given scheduling problem. This is followed by the design and implementation of the proposed Two Round Scheduling scheme in section 3. The simulation results in section 4, provides the comparative study that shows the applicability of the proposed scheme. Section 5 concludes the paper.

Table 1: Summary of some Interconnection network characteristic

Type	Size (N) (Nodes)	Degree (d)	Diameter (D)	Bisection Width (b)	Extensibility
Hypercube	$N = 2^n$	N	n	2^{n-1}	Exponential
LET	$\sum_{k=1}^{n} k$	4	√N	$2\log_2(n+2)$	Linear
LEC	$N=2*n$	4	O ([N])	N	Linear

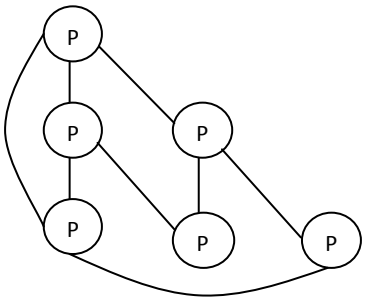

Fig 1: A six processor LET network

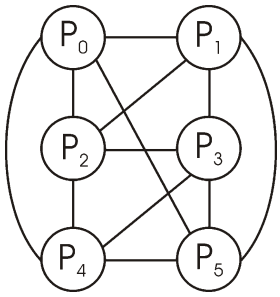

Fig 2: A six processor LEC network

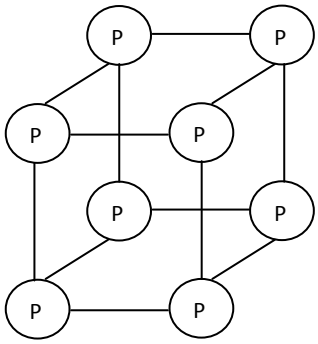

Fig 3: An eight processor hypercube network

2. DYNAMIC TASK SCHEDULING PROBLEM

The performance of a multiprocessor system can be characterized by communication delay, distribution of load among the processors and scheduling overhead [12], [13], [14], [15]. There are many schemes which are based on the principle of minimum distance feature [7] [16]. Minimum distance is the property which assures the minimization of the communication in distributing subtasks and collecting partial results. A scheduling scheme operates with this property such as Minimum Distance Scheduling (MDS) minimizes overhead and ensures the maximum possible speedup, however, at the cost of idle unconnected node(s) [7], [9]. In this scheme, the adjacency matrix of the network is used to satisfy the minimum distance property. A 'one' in the matrix indicates a link between two nodes whereas a 'zero' indicates there is no link between nodes. For load balancing, the MDS algorithm determines the value of Ideal Load (IL) at various stages of the load (task generation). IL is calculated by summing the load of each node in the network divided by the total number of nodes available in the network. The processors having a load value greater than the IL are considered as overloaded processors. Similarly, processors having lesser load than the value of IL are termed as underloaded processors. In other words the overloaded (donors) and underloaded (acceptors) processors are identified based on a threshold value known as IL. Each donor processor, during balancing, selects tasks for migration to the various connected and underloaded processors (i.e. the processors having a 'one' in the adjacency matrix) and thus maintaining minimum distance. Mostly any load balancing algorithm considers the overall load on the network. However, in this algorithm the load is mapped through various stages of the task structure. Each stage represents a particular state of the task structure which consists of finite number of tasks

2.1 Dynamic Load Model

For the purpose of simulation we assume a simple problem characterization in which the load is partitioned into a number of tasks. Each task can be an independent program or partitioned modules of a single program. However, all the tasks are independent and may be executed on any processor in any sequence. The scheduling performance of the strategy has been tested on the three different networks by simulating artificial dynamic load. In order to simulate the load on the given networks, it is characterized into two groups of task structures i.e. uniform and non-uniform load. For a meaningful simulation, tree structures that forms a representative sample of programs are needed which are to be executed on the network. The tree is considered as a test problem on which the schemes are to be applied. In case of uniform load, tasks are generated in a deterministic manner in the form of a regular tree. Each node of the tree represents a task, and executed in parallel in breadth-first manner starting from the root task which is assigned to some given nodes of the network. The total number of nodes in the task tree at a level represents a particular stage of the load.

In order to characterize non-uniform load (non-deterministic load), the total problem is conceived to be an arbitrary tree which unwind itself level by level. A task scheduled on a processor spawns an arbitrary or random number of subtasks, which are part of the whole problem tree. Thus the load on each processor is varying at run time creating unbalance, and balancer/scheduler has to be invoked after each stage.

Using the above pattern of task structure (load), the performance of the networks has been tested for various scheduling schemes as well as with a new scheduling scheme. The performance is measured in terms of Load Imbalance Factor (LIF) i.e. the load imbalance left after a balancing action at each stage of the load. The above simulation has been performed on various similar multiprocessor networks using IBM server X series 226 having Intel Xeon 3.0 GHz processor.

3. TWO ROUND SCHEDULING (TRS) SCHEME

A new scheme has been proposed for solving load balancing problem with unpredictable load estimates. The proposed algorithm works as an extension of MDS and named as Two Round Scheduling scheme. It is dynamic in the sense that no priori knowledge of the load is assumed. TRS scheme takes into consideration those acceptor nodes which are not connected directly to donor node. There may be more than one path between the donor and acceptor processors which require multi-hop. However, large number of hopes gives minimum load imbalance and hence, LIF is smaller (i.e. less than the standard range of 40%). The proposed TRS algorithm has a constraint in the scheduling to consider only one processor as intermediate node between donor and acceptor nodes. To perform the load balancing, the algorithm calculates ideal load value for each stage of the task structure, which is used as a threshold to detect load imbalances and make load migration decisions. The load imbalance factor for k_{th} stage, denoted as LIF_k, is defined as:

$$LIF_k = [\max \{load_k (P_i)\} - (ideal\_load)_k] / (ideal\_load)_k \quad (1)$$

where,

$$(ideal\_load)_k = [loadk(P_0) + loadk(P_1) + \ldots + load_k (P_{N-1})] / N, \quad (2)$$

and max ($load_k(P_i)$) denotes the maximum load pertaining to stage k on a processor P_i ,$0 \leq i \leq N-1$, and $load_k(P_i)$ stands for the load on processor P_i due to k^{th} stage. Each stage of the task structure (load) represents a finite number of tasks. Based on the IL value, the donor (overloaded) processors and acceptors (underloaded) processors are identified. Migration of task can take place between donor and acceptor processors only. The whole algorithm is implemented in 'C' language. A pseudo code of the algorithm is shown in Table 2.

Table 2: The trs algorithm

```
trs( )
{
/* Generate task at 0th processor, tgs indicates task generation at a particular stage*/
/* Consider LMAX is the maximum load on a processor at a particular load stage */
tgs[0] = 1;
while (it_count1 < LMAX)
{
/* calculate IL and RIL */
IL = Calculate_IL (tgs);
RIL = ceil (IL);
printf (tgs);
/* For all processors check whether the load on a particular processor is exceeding the RIL (Rounded IL). If so then migrate the load*/
/* Let the total number of processors are equal to PMAX */
for (it_count2 = 0; it_count2 < PMAX; ++ it_count2)
{
if (tgs [it_count2]  > RIL)
{
/* Migrate till load at processors become equal to or less then RIL */
while (true)
{
migrate (it_count2)
if (tgs [it_count2] < = RIL ) break;
} } }
printf (trs)
/* calculate LIF */
LIF = (max(tgs) – IL) / IL;
/* Enter into the next level of the task generation  (ts indicates task structure)*/
tgs = ts * tgs;
it_ count ++;
} }
/* Functions used by the algorithm */
Calculate_IL (X[ ])
{
sum = 0; /* x[i] indicates load at ith processor */
for (i = 0; i < PMAX; ++i )
sum = sum + x[i];
return (sum / PMAX);
}
/* Perform migrations */
migrate (p_number)
{
/* Get the set of connected processors to the processor for which migration is being called i.e. p_number */
for (i =0; i < PMAX; ++i )
{
if (connect ed (i, p_number, level))
temp [k++] = i ;
k--;
}
/ * Get the small loaded processor number */
small = temp [0];
for (i = 0 ; i < PMAX; ++i)
if (tgs [temp[i] ] < tgs [small] )
small = temp [j];
/* Transfer the load from p_number to the smallest loaded and connected processors */
```

```
while (tgs[p_mumber] != IL || tgs[small] != IL)
{
 tgs [p_number] --;
tgs [small] + =1; }
 }
/* Check the under loaded processors which are not
connected. If any repeat the above procedure for  the next
level of connectivity */
}
/* Function used to find the maximum load on a processor
*/
max (X [ ] )
{
max = x [0];
for (i =0; i < PMAX; ++ i)
if (x [i] > max ) max = a [i] ;
return (max);
}
/* Function to check the connectivity of processor i with
processor j. Assume the level of connectivity is given (1 or
2)*/
int connected (int i, int j, int level) /* returns true if
processors i, j are connected */
{
  /* printf("\n node %d is connected to %d: %d", i, j, adj
[i][j]); */
  if (level = = 1)
  return adj [i][j];
  for(int k = 0; k < PMAX; k++)
  {
    if (k = = i || k = = j) continue;
    if (connected (i ,k , 1) && connected (k, j, 1 ))
    {
          /* printf("\n node %d is connected to %d through
%d", i, j, k); */
          return 1;
       }  }
  return 0;       }
   end of procedure
```

4. SIMULATION AND ANALYSIS OF RESULTS

To draw general conclusion about the effectiveness of the proposed TRS scheme, it has been implemented on various multiprocessor networks under the same environment. The simulation run consists of generating various types of load and mapping them on the six processors Linearly Extensible Tree (LET), six processors LEC and eight processors hypercube networks. The estimation of LIF is obtained for various stages of the tasks structures and the curves are plotted as the average percent LIF against the load for different stages shown in Figure 4.

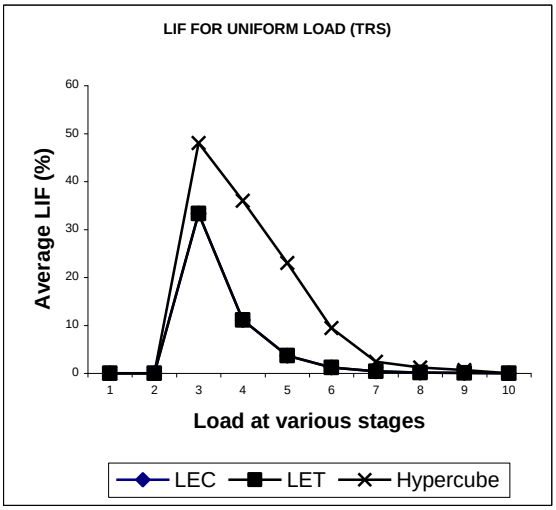

Fig 4: **TRS scheme on various multiprocessor networks**

The trends of curves obtained in Figure 4 indicate the behaviors of the load imbalance factor with respect to the load at various stages on various multiprocessors networks for uniform task structure. It is clear from the curve that initially the value of LIF starts from zero and reaches to its maximum value for all the multiprocessor networks. This high value of LIF is due to lesser numbers of tasks generated and balancing on the networks. When sufficient numbers of tasks are available a lesser value of LIF is obtained. Therefore, a good balancing is achieved when more numbers of tasks are available.

To study the effect of proposed Two Round Scheduling scheme, the simulation results are evaluated for linearly extensible multiprocessors i.e. LET and LEC networks and are compared with other standard networks such as Hypercube network. The comparative study indicates similar behavior on all the multiprocessor networks for uniform load. The value of LIF starts reducing when sufficient numbers of tasks are available and finally approaches to zero in all the multiprocessors networks. However, the value of LIF is similar in case of LET and LEC networks. This indicates that the TRS algorithm is performing equally well on both the LET and LEC networks for uniform load. The reason for better performance in LET and LEC networks for uniform load is due to the lesser number of processors in the networks.

To check further that the TRS is performing better on LEC or LET, the load balancing time (i.e. time taken to get a zero value of LIF) of TRS scheme is evaluated on both the networks (i.e. LEC and LET). The balancing time at various load stages for uniform task structure are evaluated as another performance parameter, for all the multiprocessor networks mentioned above and curves are plotted as Time verses Load at different stages, shown in Figure 5.

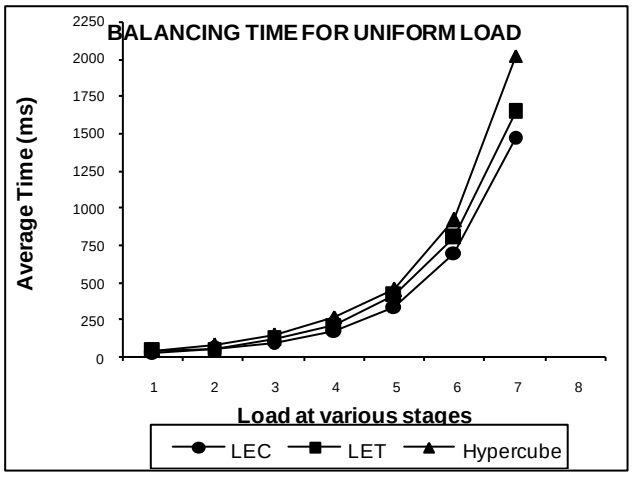

Fig 5: Performance of various multiprocessor networks

The performance results shown in Figure 5 indicate that in all the multiprocessor networks, the balancing time increases as the number of tasks increases. However, the balancing time in LEC network is always lesser as compared to other networks. Therefore, albeit the LEC and the LET networks produce similar results in terms of LIF when TRS is implemented for uniform load, LET gives an inferior performance when balancing time is taken into consideration. Thus, it may be concluded that the LEC network is outperforming in comparison to Hypercube network and gives comparable performance with LET network.

In case of non-uniform load (Figure 6), the effect of variation of load is clearly indicated in the curves. The LIF starts from zero and reaches to its high value which stays high for several stages of the task generation on all the multiprocessor networks. This high value of LIF is due to the lesser number of tasks running on the networks during these stages. The smaller number of tasks could not be balanced among all the processors of the networks which indicate an inefficient balancing in the networks. However, in all the cases the value of LIF starts decreasing towards minimum as the numbers of tasks are sufficient. In general the value of LIF is lesser in case of LEC network except the earlier stages of the load generation when sufficiently tasks are not available. Therefore, it may be concluded that the LEC network is performing better with TRS algorithm for non-uniform load also.

The performance of LEC is also evaluated in terms of load balancing time for non-uniform load. The TRS scheme is implemented on various multiprocessor networks and balancing time on each of the network is evaluated for various load stages. For comparison purpose the curves are plotted as Time verses Load (non-uniform) shown in Figure 7.

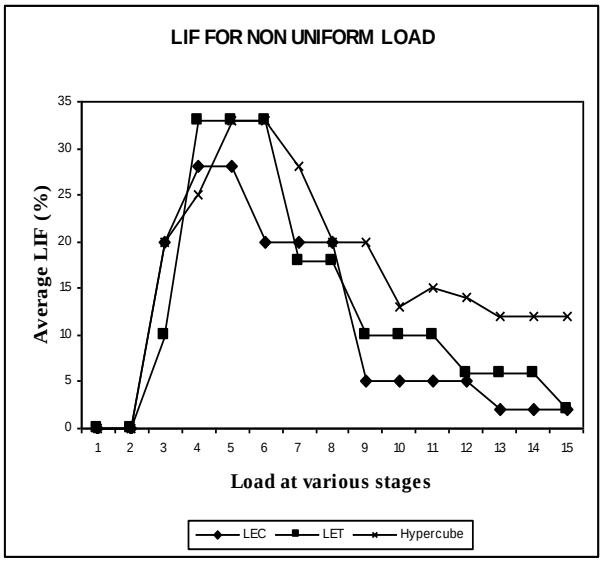

Fig 6: TRS scheme on various multiprocessor networks

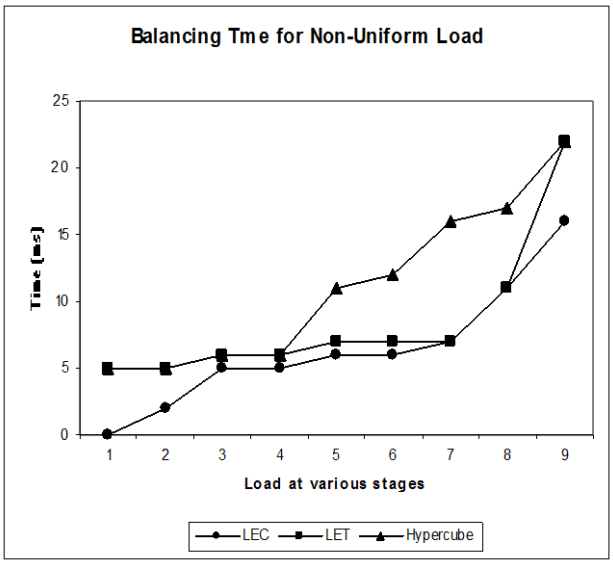

Fig 7: Performance of various multiprocessor networks

The performance results shown in Figure 7 indicate the behavior of balancing time when TRS scheme is implemented on various multiprocessor networks for non-uniform load. It is observed from the curves that there is no regular pattern in the balancing time with load. In case of non-uniform load the behavior of the tasks is unpredictable; therefore, the balancing time varies on each of the multiprocessor network. In general, the Hypercube network shows different behavior in the time. On the other hand LEC and LET indicate similar effect on the change in balancing time for various load stages. This is due to the fact that both the LEC and LET networks have smaller number of processors. It might be possible that the tasks available are lesser and consequently these lesser tasks are efficiently balanced on smaller number of processors. However, LEC shows the lesser balancing time.

To authenticate the performance of the proposed TRS scheme with the LEC network, it is desirable to implement other standard dynamic scheduling schemes on LEC. In the present work, in addition to the proposed TRS scheme, some reported dynamic scheduling schemes have also been implemented on the LEC network under the same environment [14], [15], [16], [17],[18]. In particular, the following two scheduling schemes were considered and have been implemented on LEC network after appropriate modification. These schemes have given optimal performance on the particular multiprocessor networks for which they are designed.

- Minimum Distance Scheduling (MDS)
- Hierarchical Balancing Method (HBM)

In case of MDS, migration from donor processor is done to the directly connected acceptors. Tasks are not allowed to migrate acceptors which are not connected directly. The HBM scheme is an asynchronous and decentralized approach. It classifies the multiprocessor system into a hierarchy of balancing domain. For the purpose of simulation the LEC network is divided into three level of hierarchy namely level0, level1 and level2 and the imbalance are evaluated. The estimation of average percentage of LIF is obtained and the curves are plotted against the various load stages for uniform load, shown in Figure 8.

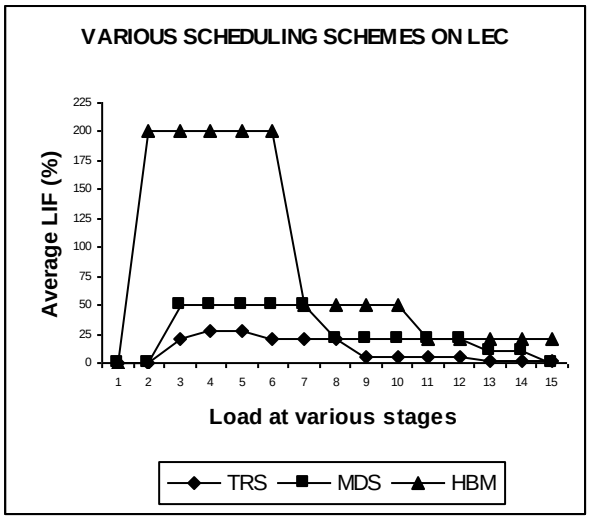

Fig 8: Comparison of TRS with other scheduling schemes on LEC

The TRS scheme shows the lesser imbalance with negligible average value of LIF as we tend to higher load stages. The value of LIF becomes zero, when the network receives good amount of load and thus, it is immediately balanced. Therefore, it can be said that the TRS scheme has a high degree of balancing, which makes the LEC network as effectively utilized in comparison to other scheduling schemes, when implemented on LEC. The other scheduling schemes are unable to achieve high degree of balancing for uniform and non-uniform types of load structure. TRS scheme indicates the superior values, i.e. lesser values at every point as well as lesser values of LIF at peaks with respect to other scheduling schemes.

5. CONCLUSIONS

The overall performance of the multiprocessor system is affected by a number of factors, such as communication delays, imbalance of load among the processor and scheduling overheads. Scheduling plays a vital role to improve the performance of the system and hence a Two Round scheduling algorithm has been proposed and implemented on various similar multiprocessor systems. The performance evaluated in terms of load imbalance and the balancing time. The performance of the TRS algorithm is highly dependent on the connectivity of the various nodes available in the network. However, the algorithm allocates the tasks to the available processors in the network whether they are connected directly or indirectly. From the comparison made on the graphs based on various simulation results, it may be concluded that TRS scheme is performing well on linearly extensible multiprocessor type systems in general and on LEC network in particular while considering the factor of LIF and its balancing time. The proposed TRS scheduling scheme is performing better, degree of balancing is higher and the network utilization is efficient. Therefore, it can be concluded that proposed TRS scheme is ideally suited for linearly extensible multiprocessor networks. The proposed TRS scheme may be applied to other similar multiprocessor network for better network utilization.

6. REFERENCES

[1] M. J. Zaki, Wei Li, and S. Parthasarathy, "Customized Dynamic Load Balancing for a Network of Workstations", Journal of Parallel and distributed Computing, no. 43, 1997, 156-162.

[2] Z. Zeng, B. Veeravalli, "Design and Performance Evaluation of Queue-and-Rate-Adjustment Dynamic Load Balancing Policies for Distributed Networks", IEEE Trans. on Computers, vol. 55, no. 11, 2006, 1410-1422.

[3] S. Salleh, N. A. B. Aziz, N. A Azmee and N. H. Mohammed, "Dynamic Multiprocessor Scheduling for the Reconfigurable mesh Computing Network", Journal of Technology, University of Technology, Malaysia, vol. 37, 2002, 55-66.

[4] B. Parhami., "Challenges in interconnection network design in the era of multiprocessor and massively parallel Microchips" In Proceedings of International Conference on communication in Computing, 241-246.

[5] S. Kim and A. V Veidenbaum, "Interconnection network organization and its impact on performance and cost in shared memory multiprocessors" Journals of parallel computing, vol. 25, 1999, 283-309.

[6] A. Patel, A. Kasalik, and C. McCrosky, "Area Efficient VLSI Layout for binary Hypercube", IEEE Transaction on Computers, vol. 49, no. 2, 2006, 160-169.

[7] M. Q. Rafiq, P. Kumar and J. P. Gupta., "A Novel Tree-Structured Multiprocessor Network", In Proceedings of International Conference of on Robotics Vision and Parallel Processing for Automation, Malaysia, vol. 2, 1995, 576-585.

[8] A. Samad, M. Q. Rafiq and O. Farooq, "A Novel Algorithm for Fast Retrieval of Information from A Multiprocessor Server", In Proceedings of 7th WSEAS International Conference on Software Engineering,

Parallel and Distributed Systems (SEPADS '08), University of Cambridge, UK, 2008, 68-73.

[9] A. Samad and M. Q. Rafiq, "A Novel Server Architecture for Networking", In Proceedings of Int'l Conference on Robotics, Vision Information and Signal Processing, Malaysia, 2005, 1029-1032.

[10] B. Towles and W. Dally. Principles and Practices of Interconnection Network. Morgan Kaufmann Press, san Francisco.

[11] A. Ishfaq and A. Ghafoor, "Semi-Dostributed Load Balancing For Massively Parallel Multicomputer Systems", IEEE Transaction on Software Engineering, vol. 17, no. 10, 1991, 987-1004.

[12] W. M. H. LeMair and A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers", IEEE Transaction on Parallel and Distributed Systems, vol. 4, no. 9, 1993, 979-92.

[13] H. Attiya, "Two phase Algorithm for Load Balancing in Heterogeneous Distributed Systems", In Proceedings of 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euro-PDP'04), 2004, 434-439.

[14] M. Bertogna., M. Cirinei, and G. Lipari, "Schedulability analysis of Global scheduling algorithm on multiprocessor platforms", IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 4, 2009, 553-566.

[15] M. Dobber, R. V. D. Mei and G.Koole, "Dynamic Load Balancing and Job Replication in a Global-Scale Grid Environment: A Comparison" IEEE Transaction on Parallel and Distributed Systems, vol. 20, no. 2, 2009, 207-218.

[16] Yiqiu Fang, Fei Wang, Junwei Ge, "A Task Scheduling Algorithm Based on Load Balancing in Cloud Computing" Lecture Notes in Computer Science, Issue: 6318, Publisher: Springer-Verlag, 2010, 271-277.

[17] Bertogna, M., Cirinei, M., and Lipari, G., "Schedulability analysis of Global scheduling algorithm on multiprocessor platforms", IEEE Transaction on Parallel and Distributed Systems, volume 20, number 4, 2009, 553-566.

[18] D.I. George Amalarethinam and G.J. Joyce Mary, "A new DAG based Dynamic Task Scheduling Algorithm (DYTAS) for Multiprocessor Systems". International Journal of Computer Applications (0975 – 8887) Vol. 19, No. 8, 2011, 24-28.