Analysis of various dynamic load balancing strategies used in parallel systems

Atul Kumar Singh, Akhil Jain, Mohammad Haroon, Garima Singh

Department of Computer Science, College of Engineering, Teerthanker Mahaveer University, Moradabad (U.P.), India
Email: er.atul.kumar.singh@gmail.com, toakhiljain@gmail.com, haroonayme@gmail.com, garimasingh.0606@gmail.com

Abstract: In this paper the dynamic load balancing strategies are discussed to minimize the execution time of single applications running in parallel on multicomputer systems. Dynamic load balancing is important for the efficient use of parallel systems. Dynamic load balancing schemes are needed to solve non-uniform problems on multiprocessor systems. Distribution of the work load is known as Load Balancing. An appropriate distribution of workloads across the various processing elements is very important as disproportional workloads can eliminate the performance benefit of parallelizing the job. Load balancing on parallel systems is a critical and challenging activity. Load balancing policies may be categorized as static or dynamic. Static load balancing algorithms distribute the tasks to processing elements at compile time and are generally based on the information about average behavior of the system, while dynamic algorithms bind tasks to processing elements at run time and react to the actual current system state in making transfer decisions.
[Atul Kumar Singh, Akhil Jain, Mohammad Haroon, Garima Singh. **Analysis of various dynamic load balancing strategies used in parallel systems.** Journal of American Science 2011; 7(10):599-606]. (ISSN: 1545-1003). http://www.americanscience.org.

Keywords: Dynamic Load Balancing, Parallelism, Microprocessors.

Introduction:-

One of the most important issues is how to effectively utilized parallel computers that have become increasingly complex to improve the performance. Such systems are constructed by different processor connected with communication link to operate in parallel with relatively low cost known as multi processor system.

The parallel computer is one of the remarkable developments of methodology and technology in computer science in recent years. Due to multiprocessor structure of the computer architecture, this computer has a capability to execute multiple instructions or multiple data simultaneously. The parallel computer not only provides support for efficient computation of mathematical, economical, industrial, and ecological problems but also aims new computer architecture beyond the traditional von Neumann type.

And multiprocessor system be very efficient at solving problems that can be partitioned into tasks with uniform computation and communication patterns. However, there exists a large class of non uniform problems with uneven and unpredictable computation and communication requirements. Therefore Dynamic load balancing (DLB) schemes are needed to efficiently solve non-uniform problems on multiprocessor systems.

In the present project, efforts are concentrated on the design of a novel multiprocessor architecture and to schedule the arriving load on to it, in order to achieve higher performance. In addition to designing an appropriate network, the efficient management of parallelism on the network involves optimizing performance needed like the minimization of communication and scheduling overhead.
In addition to the simulation studies are carried out to compare the performance of proposed triangular network and these scheduling schemes with standard hypercube multiprocessor architecture.

Need For Parallelism:

- Need of parallelism arise from the need to build faster and faster machines and achieve higher computing speed.
- When applications require throughput rates that are not easily obtained with today's sequential machines, parallel processing offers a solution.
- Parallel processing is based on Multiprocessor processors working together to accomplish a task to gain high performance.
- Exploiting parallelism is now a necessity to improve the performance of computer systems.

That's why we need to develop a multiprocessor architecture with low cost and high performance.

Generally stated, parallel processing is based on several processors working together to accomplish a task. The basic idea is to break down, or partition, the computation into smaller units that are distributed among the processors. In this way, computation time is reduced by a maximum factor of p, where p is the number of processors present in the multiprocessor system.

Most parallel algorithms incur two basic cost components:

1. Computation delay—under which we subsume all related arithmetic/logic operations, and

2. Communication delay—which includes data movement.

In a realistic analysis, both factors should be considered.

Dynamic load Balancing

Dynamic load balancing (DLB) is essential for the efficient use of highly parallel systems when solving non-uniform problems with unpredictable load estimates. Dynamic load balancing schemes which seek to minimize total execution time of a single application running in parallel on a multicomputer system.

Multiprocessor system be very efficient at solving problems that can be partitioned into tasks with uniform computation and communication patterns. However, there exists a large class of non uniform problems with uneven and unpredictable computation and communication requirements. Therefore Dynamic load balancing (DLB) schemes are needed to efficiently solve non-uniform problems on multiprocessor systems

To do so, an optimal tradeoff between the processing & communication overhead and the degree of knowledge used in the balancing process must be sought.

A GENERAL DYNAMIC LOAD BALANCING MODEL:-

We have developed a general model for dynamic load balancing.

This model is organized as a four phase process:

1) Processor load evaluation
2) Load balancing profitability Determination
3) Task migration strategy
4) Task selection strategy

Phase1: Processor Load Evaluation

- A load value is estimated for each processor in the system.
- These values are used as input to the load balancer to detect load imbalances and make load migration decisions**.**

Phase2: Load Balancing Profitability Determination:

- The imbalance factor quantifies the degree of load imbalance within a processor domain.
- It is used as an estimate of potential speedup obtainable through load balancing
- It is weighed against the load balancing overhead to determine whether or not load balancing is profitable at that time**.**

Phase 3: Task Migration Strategy:

Sources and destinations for task migration are determined. Sources are notified of the quantity and destination of tasks for load balancing**.**

Phase 4: Task Selection Strategy:

Source processors select the most suitable tasks for efficient and effective load balancing and send them to the appropriate destinations**.**

- The first and fourth phases of the model are application dependent and purely distributed. Both of these phases can be executed independently on each individual processor.
- Our focus is on the Profitability Determination and Task Migration phases, the second and third phases, of the load balancing process
- As the program execution evolves, the inaccuracy of the task requirement estimates leads to unbalanced load distributions.
- The imbalance must be detected and measured (Phase 2) and an appropriate migration strategy devised to correct the imbalance (Phase *3).*
- During the Profitability Determination Phase a decision is made as to whether or not to invoke the load balancer.
- The load *imbalance factor* $\Phi(t)$ is an estimate of the potential speedup obtainable through load balancing at time t .
- It is defined as the difference between the maximum processor loads before and after load balancing, L_{max} *and* L_{bal} , respectively.

$$\Phi(t) = L_{max} - L_{bal}$$

A decision on whether or not to load balance is made based on the value of $\Phi(t)$ relative to the balancing overhead, $L_{overhead}$, required to perform the load balancing. In general, load balancing is profitable if the savings is greater than the overhead, i.e.

$$\Phi(t) > L_{overhead}$$

The responsibility of invoking the balancer may either be authorized to all processors in the system or only to designated processors containing the necessary information. For highly parallel systems it is desirable to distribute the responsibility to multiple points in the system. This may be accomplished by Partitioning the system into independent groups of processors called ***balancing domains.*** The size of a balancing domain may range anywhere from a few processors to the entire system.

Load balancing decisions are based solely on information pertaining to those processors within each domain. The notion of balancing domains is a way of distributing the balancing process. Furthermore, by decreasing the number of processors being considered in the balancing process, balancing domains reduce the complexity of calculating the imbalance factor as well as the complexity of phase 3, the ***Load Migration Strategy.***

Potentially more accurate migration strategies are made possible by larger balancing domains. However, larger domains may increase the aging period of information and cause the load balancing overhead to be more unevenly distributed. These tradeoffs are illustrated by the different strategies to be discussed.

DYNAMIC LOAD BALANCING STRATEGIES:
The following five **DLB** strategies are designed to support highly parallel systems.

1. Sender Initiated Diffusion (SID)
2. Receiver Initiated Diffusion (RID)
3. Hierarchical Balancing Method (HBM)
4. Gradient Model (GM)
5. Dimension Exchange Method (DEM)

The schemes presented vary in the amount of processing and communication overhead and in the degree of knowledge used in making balancing decisions.

(1) Knowledge- The accuracy of each balancing decision

(2) Overhead- The amount of added processing and communication incurred by the balancing process.

The load balancing overhead includes the communication costs of acquiring load information and of informing processors of load migration decisions, and the processing costs of evaluating load information to determine task transfers.

1. Sender Initiated Diffusion (SID)

The SID strategy is a, local, near-neighbor *diffusion* approach which employs overlapping balancing domains to achieve global balancing. A similar strategy, called Neighborhood averaging, is proposed in. The scheme is purely distributed and asynchronous. Each processor acts independently, apportioning excess load to deficient neighbors.

It has been shown in, that for an *N* processor system with a total system load *L* unevenly distributed across the system, a diffusion approach, such as the SID strategy, will eventually cause each processor's load to converge to *L/N.*

Balancing is performed by each processor whenever it receives a load update message from a neighbor indicating that the neighbors load, 1_i<Ideal Load , where *Ideal Load* is a preset threshold. Each processor is limited to load information from within its own domain, which consists of itself and its immediate neighbors. All processors inform their neighbors of their load levels and update this information throughout program execution. The profitability of load balancing is determined by first computing the average load in the domain, *Lp,*

$$L_p = \frac{1}{K+1}\left(l_p + \sum_{k=1}^{K} l_k\right)$$

2. Receiver Initiated Diffusion (RlD):

The RID strategy can be thought of as the converse of the SID strategy in that it is a receiver initiated approach as opposed to a sender initiated approach. However, besides the fact that in the RID strategy under loaded processors request load from overloaded neighbors, certain subtle differences exist between the strategies. First, the balancing process is initiated by any processor whose load drops below a pre specified threshold (L_{Low}).

Second, upon receipt of a load request, a processor will fulfill the request only up to an amount equal to half of its current load (this reduces the effect of the aging of the data upon which the request was based). Finally, in the receiver initiated approach the under loaded processors in the system take on the majority of the load balancing overhead, which can be significant when the task granularity is fine.

As with the SID strategy, each processor is limited to load information from within its own domain, which consists of itself and its immediate neighbors. All processors inform their near-neighbors of their load levels and update this information throughout program execution.

The RID strategy differs from its counterpart **SID** in the task migration phase. Here, an under loaded processor first sends out requests for load and then receives acknowledgment for each request.

3. The Gradient Model (GM)

- The gradient model is a demand driven approach.
- The basic concept is that under loaded processors inform other processors in the system of their state, and overloaded processors respond by sending a portion of their load to the nearest lightly loaded processor in the system.
- This model employs a gradient map of the proximities of under loaded processors in the system to guide the migration of tasks between overloaded and under loaded processors.

The resulting effect is a form of relaxation where tasks migrating through the system are guided by the proximity gradient and gravitate
towards under loaded points. The scheme is based on two threshold parameters: the ***Low-Water-Mark*** *(LWM)* and the ***High- Water-Mark (HWM).*** **A** processor's state is considered light if its load is below the **LWM,** heavy if above the **HWM,** and moderate otherwise. **A** node's ***proximity*** is defined as the shortest distance from itself to the nearest lightly loaded node in the system. All nodes are initialized with a proximity of w_{max}, a constant equal to the diameter of the system. The proximity of a node is set to 0 if its state becomes light. All other nodes *p* with near-neighbors ***n2*** compute their proximity as

$$\text{proximity}(p) = \min(\text{proximity}(n_i)) + 1.$$

A node's proximity may not exceed $\mathbf{w_{max}}$**. A** system **is** saturated, and does not require load balancing if all nodes report a proximity of $\mathbf{w_{max}}$.If the proximity of a node changes it must notify its near-neighbors. Hence, the balancing process is initiated by lightly loaded processors reporting a proximity of 0. In order for load balancing to take place, there must be at least one overloaded processor and one under loaded processor in the system. No measure of the degree of imbalance is made, only that one exists. This criterion is characterized by the simplified version of the load balancing profitability determination phase**,** where, given an overloaded processor ***p*** and an under loaded processor *q,*

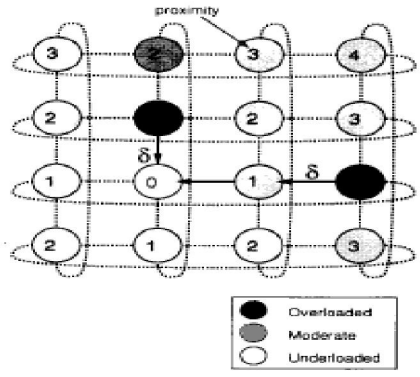
$$Lp - Lq > \text{HWM} - \text{LWM}.$$

The proximity map is used to perform the migration phase. If a processor's state is heavy and any of its near-neighbors report a proximity less than $\mathbf{w_{max}}$, then it sends a unit of its load to the neighbor of lowest proximity. Tasks are routed through the system in the direction of the nearest under loaded processors. A task continues to migrate until it reaches an under loaded processor or it reaches a node for which no neighboring nodes report a lower proximity. The scheme is illustrated in Figure below. In this example, there are two overloaded nodes in the system and one under loaded node. The overloaded nodes are at different proximities from the under loaded node, but both send a fraction of load, 6, in the direction of the under loaded processor. The value of *5* can be determined as either a percentage of the initial load, or as a fixed number of tasks. The scheme may perform inefficiently when either too much or too little work is sent to an under loaded processor.

Given *N* processors interconnected using a hypercube topology, in the worst case, an update of the gradient map, to recognize the presence of a new under loaded processor, would require,

$$C_{tOt}(\text{update}) = N \log \text{Nmessages}.$$

The worst case occurs when there are no other under loaded processors in the system.
The migration of tasks from overloaded to under loaded processors incurs added overhead due to the asynchronous nature of the algorithm.



At the other extreme, an overloaded processor, in transferring a preset portion of load, may not send enough to solve the imbalance. Hence, the degree of information used in the balancing process may lead to inefficient migration decisions.

4. Hierarchical Balancing Method (HBM)

- It is an asynchronous global, approach which organizes the system into a hierarchy of subsystems.
- Load balancing is initiated at the lowest levels in the hierarchy with small subsets of processors and ascends to the highest level which encompasses the entire system.
- Specific processors are designated to control the balancing operations at different levels of the hierarchy.

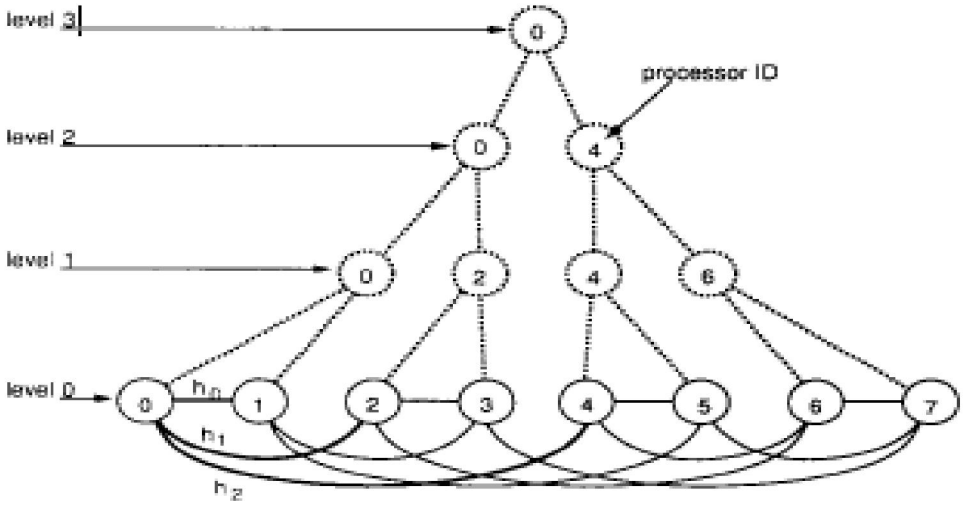


Fig. Hierarchical organization of an eight-processor system with hypercube interconnections, where h_k is the connection to the neighbor at the kth level. The processor IDs at intermediate nodes in the tree represent those processors delegated to manage the balancing of corresponding lower-level domains.

- The hierarchical balancing scheme functions asynchronously.
- The balancing process is triggered at different levels in the hierarchy by the receipt of load update messages indicating an imbalance between lower level domains.
- All load levels are initialized with each processor sending its load information up the tree.

5. Dimension Exchange Method (DEM)

- It is a global, fully synchronous approach.
- Load balancing is performed in an iterative fashion by "folding" an ***N*** processor system into log ***N*** dimensions and balancing one dimension at a time.

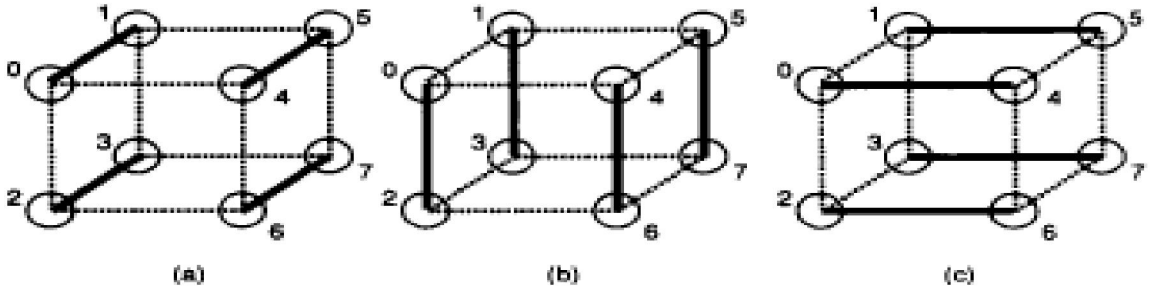


Fig. DEM strategy. All processors balance in order, in each dimension. In the example shown, a three-dimensional cube of eight processors, balancing is performed between neighbors in each dimension (a), (b), and (c). Due to synchronization requirements, the balancing between each pair of nodes must follow the order shown.

- In this scheme small domains are balanced first and these then combine to form larger domains until ultimately the entire system is balanced.
- In this scheme Balancing is initiated by any under loaded processor which has a load that drops below a preset threshold ,

$$L_p < L_{Threshold}$$

- This processor broadcasts a load balancing request to all other processors in the system.

COMPARISON ANALYSIS

Factor	SID	RID	GM	HBM	DEM
Migration Decisions	Designted processor	Designated processor	Underloaded processors	Designated processor	Designated processor
Balancing Domains	Overlapping Domain	Overlapping Domain	vary	vary	vary
Degree of Knowledge	Small	Small	High	Small	Small
Aging of Information	Length of updating interval & no. of processor	Length of updating interval & no. of processor	Diameter of the System	No. of level & no. of processor	Constant

Comparison analysis:

The differences between the five schemes are categorized into the following areas: sender or receiver initiation of the balancing, type of balancing domain, degree of knowledge used in the decision process, aging of information in the decision process, and overhead distribution and complexity. This comparison is summarized in Table

A. Balancing Initiation:

Our results indicate that the receiver initiated diffusion approach (RID) outperforms the sender initiated approach (SID) over the entire range of task granularities tested.

In theory, both approaches should yield similar results. Practical implementation issues, however, distinguish these approaches from one another. The strategy, receives load update messages from neighbors as in the 'ID strategy, but Operations are triggered by changes in the processor's own load (i.e., when it drops below a preset threshold).

Both the 'ID and the strategies make load decisions based on the load status of their near neighbors. This load information suffers from the aging process.

In minimizing execution time it is beneficial to spare overloaded processors the burden of load balancing responsibilities. The extent of the overhead is dependent on the task granularity, and may become significant if tasks are small.

The study concludes that the Receiver Initiated policy is preferable to the Sender Initiated policy at high system loads when the transfers of tasks under the two strategies are comparable. This is understandable since in a heavily loaded system there will be fewer under loaded" nodes that are hard to find and a Receiver initiated approach would be more effective.

The GM and DEM strategies are also receiver initiated, but migration decisions are not the sole responsibility of the receiver. In the DEM scheme, once the balancer is invoked, migration decisions are made synchronously by designated processors. The GM scheme is slightly more difficult to characterize since under loaded processors (receivers) alert the system of their presence, but no explicit request is made to any particular overloaded processor (sender). Senders simply release tasks into the system in the presence of an under loaded node. Finally, in the HBM scheme migration decisions are made by designated processors in the system.

B**. Balancing Domains:**

The use of balancing domains is a means of decentralizing the balancing process and reducing its complexity. Two types of domains exist; overlapping domains which achieve global balancing through the process of diffusion and variable domains which change shape and/or members in subsequent balancing iterations. It has been shown in , where they refer to the balancing domains as buddy sets, that for a hypercube system using overlapping domains, there exists a maximum size domain beyond which the balancing process no longer benefits by using larger domains. The SID and RID approaches employ overlapping domains while all three other approaches use variable domains. The balancing domains in the HBM strategy vary to include a larger subset of processors at higher levels in the hierarchy. The same is true for the DEM strategy where each dimension is balanced in turn. Finally, the GM domains vary according to the location of the nearest under loaded processor.

C. **Degree of Knowledge:**

The degree of global knowledge, also referred to as information dependency, used in the balancing process is critical to the accuracy of balancing decisions. The more knowledge available in the decision process the more effectively the balancer can correct imbalances in the global load distribution. The SID and RID strategies only make use of a small degree of knowledge (load levels of h-neighbors) in each balancing decision. Both the HBM and DEM strategies use only a small degree of knowledge in each balancing step, but some additional knowledge is implicitly known. The HBM strategy is structured in such a way that, while the technique is asynchronous, lower level domains will balance themselves before upper level domains when imbalances exist.

D. **Aging of Information:**

The accuracy of the information used by the load balancer is vital to its effectiveness. Three of the four strategies described make use of a periodic update strategy. This update strategy is critical to the accuracy of load information in terms of the aging period. The aging of information specifically refers to the length of the delay from the time of load information determination to the time it is used in making balancing decisions. This delay is particularly critical when the load levels are changing at a rapid rate and the load information is only valid for a short period of time. Aside from the update interval, u(t), on the delay depends both the system communication latency as well as on the amount
of information being acquired. For the SID, RID, and HBM strategies the aging period depends primarily on the length of the update interval, u(t).

For the RID and SID strategies the aging period is also affected by the number of processors per domain, O(K).The HBM strategy aging period depends on the hierarchical organization, including the number of levels in the hierarchy as well as on the number of processors per branch (e.g.. O(log N)for a binary-tree organization). The aging period of the DEM strategy, because it operates synchronously, is constant, while that of the GM scheme is O(diameter(N)), where diameter(N) is the maximum number of hops between any two processors in the system.

E. **Overhead Distribution and Complexity**:

It is desirable to both minimize the load balancing overhead as well as to distribute it evenly across all processors in the system. This eliminates any bottlenecks in the balancing process and increments

in the overhead will not severely impact system performance.

Furthermore, the balancing overhead should be scalable to support large systems. Both the SID and RID strategies achieve a uniform overhead distribution that is independent of AV.but increases instead as O(K),the number of neighbors. The RID strategy, however, requires two more messages per task transfer. The HBM scheme also distributes the load balancing overhead, but some processors incur a larger portion than others. For a binary tree organization, the disparity in the overhead distribution is O(N/logN),or 1 : 3 given a broadcast mechanism . Nonetheless, the average overhead per processor increases as O(1ogN).For the DEM strategy some synchronization mechanism is required once the load balancer is invoked.

The overhead of the GM scheme is difficult to measure. In setting up the gradient map each processor in the GM scheme may need to update its proximity O(L\T) times. Furthermore, in the GM scheme, the processors in the path of migration incur additional overhead in forwarding tasks to their destinations.

Since these destinations are not fixed, unless a limit is put on the number of hops a task is permitted to travel, tasks may continue to migrate through the system indefinitely.

References:

1. Ardhendu Mandal and Subhas Chandra Pal "An Empirical Study and Analysis of the Dynamic Load BalancingTechniques Used in Parallel Computing Systems" Nov, 2010.
2. **A.Samad, M.Q. Rafiq and Omar Farooq** "A novel algorithim for fast retrival of information from a multiprocessor server" Proc. Intl. on parallel and distributed system, U.K. Feb 20 to feb 22, 2008
3. Marc H. Willebeek-LeMair, ***Member, IEEE,*** and Anthony P. Reeves **"**Strategies for Dynamic Load Balancing on Highly Parallel Computers", ***Senior Member,*** *IEEE,* IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. **4,** NO. 9, SEPTEMBER 1993
4. Z. Zeng and B.Veeravalli "Design and Performance of Queue and Rate Adjustment Dynamic load Balancing Polices for Distributed Network". IEEE trans. On computer, vol 55 ,no. 11, pp. 1410-1422,nov 2006.
5. Abdel A E and Khaled d," The Hyperstar interconnection Network"journal of Parallel and distributed computing no. 48,pp 175-199,1998
6. Aebi A., Sarbazi Azad, H., Shamaei, A., Meraji, S., XMulator: An Object Oriented XML-Based Simulator, accessible at http://www.XMulator.org, 2006.
7. Razi-Azad H., "Constraint-based performance comparison of multi-dimensional interconnection networks with deterministic and adaptive routing strategies", *Journal of Computers and Electrical Engineering*, vol. 30, pp.167-82,2004.
8. Meraji S., Sarbazi-Azad H., Nayebi A., "Message routing and performance issues in necklace hypercubes", Technical Report, School of Computer Science, IPM, Tehran,Iran, 2006.
9. **N**izadeh M., and Sarbazi-Azad, H., The necklace hypercube: a well scalable hypercube-based interconnection network for multiprocessors, *ACM SAC 2005*, pp.729-733, 2005.
10. K. G. Shin and Y.-C. Chang, "Load sharing in distributed realtime systems with state-change broadcasts," ***IEEE Trans. Comput.,*** pp. 1124-1142, Aug. 1989.
11. V. **A.** Saletore, **"A** distrubuted and adaptive dynamic load balancing scheme for parallel processing of medium-grain tasks," in ***Proc. Fifth Distributed Memory Comput. Conf,*** Apr. 1990, pp. 995-999.
12. **W.** Shu and L. **V.** Kale, "A dynamic scheduling strategy for the Charekernel system," in ***Proc. ACM Supercomput.*** *Con$,* 1989, pp. 389-398.
13. M. Willebeek-LeMair and A. P. Reeves, **"A** general dynamic load balancing model for parallel computers," Tech. Rep. EE-CEG-89- 1,Cornell School of Electrical Engineering, 1989.
14. D. P. Bertsekas and J. N. Tsitsiklis, ***Parallel and Distributed Computation:Numerical Methods.*** Englewood Cliffs, NJ: Prentice-Hall, 1989.K. M. Dragon and J. L. Gustafson,

10/24/2011