Various Approaches for Dynamic Load Balancing for Multiprocessor Interconnection Network

Mukul Varshney
Assistant professor,
Computer Science and Engineering, Sharda University

Dr. Anand Sharma
HOD, Aligarh College of Engineering and Technology, Aligarh

Abhakiran Rajpoot
Assistant professor ,
Computer Science and Engineering, Sharda University

ABSTRACT

Multiprocessor interconnection network have become powerful parallel computing system for real-time applications. Now a days the many researchers doing research on the dynamic load scheduling in multiprocessor system. Load balancing is the method of dividing the total load among the processors of the distributed system to progress task's response time as well as resource utilization whereas ignoring a condition where few processors are overloaded or under loaded or moderately loaded. However, in dynamic load balancing algorithm presumes no priori information about behavior of tasks or the global state of the system. There are numerous issues while designing an efficient dynamic load balancing algorithm that involves utilization of system, amount of information transferred among processors, selection of tasks for migration, load evaluation, comparison of load levels and many more. This paper enlightens the performance analysis on dynamic load balancing strategy (DLBS) algorithm, used for hypercube network in multiprocessor system. Dynamic load scheduling (DLB) algorithm are required to efficiently solve this problems on multiprocessor systems.

In this paper our focus on study and evaluation of various dynamic load balancing strategies such as SID, RID,DEM ,GM HBM etc.

KEYWORD: Interconnection network, Parallel processing, Multiprocessor System, Load Balancing, Scheduling Algorithm, knowledge over head, threshold

I. Introduction

In computing, load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives[1]. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process.

When adaptive algorithms are used, after an interval of computation, the mesh may be refined (or coarsened) at some locations, usually based on an estimate of the discretization error. The refinement (or coarsening) process can generate widely varying numbers of mesh nodes on the processors. Subsequently, there is a need for dynamic load balancing. Load imbalance may also be caused by the use of local time stepping, local spatial approximation schemes of varying orders [2], or non-linear material properties.

Load balancing differs from channel bonding in that load balancing divides traffic between network interfaces on a network socket (OSI model layer 4) basis, while channel bonding implies a division of traffic between physical interfaces at a lower level, either per packet (OSI model Layer 3) or on a data link (OSI model Layer 2) basis with a protocol like shortest path bridging.

The tradeoff between knowledge and overhead is illustrated, by example, with five different DLS schemes. The schemes presented vary in the amount of processing and communication overhead and in the degree of knowledge used in making balancing decisions. 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[2,4]

Sender Initiated Diffusion (SID)' is a highly distributed local approach which makes use of near-neighbor load information to apportion extra load from heavily loaded processors to underloaded neighbors in the system. Receiver Initiated Difision (RID) is the converse of the SID strategy, where underloaded processors requisition load from heavily loaded neighbors. Hierarchical Balancing Method (HBM) 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. Gradient Model (GM) [6,15]employs a gradient map of the proximities of underloaded processors in the system to guide the migration of tasks between overloaded and underloaded processors. Dimension Exchange Method (DEM) [12,14], is a global, fully synchronous, approach. Load balancing is performed

II. Example of Dynamic Load Balancing

As a simple example, figure 1 shows a mesh of shape ``A'', partitioned into 8 subdomains. It has been refined in Figure 1 (b). Due to the mesh refinement subdomain 1 has more nodes than the other subdomains.

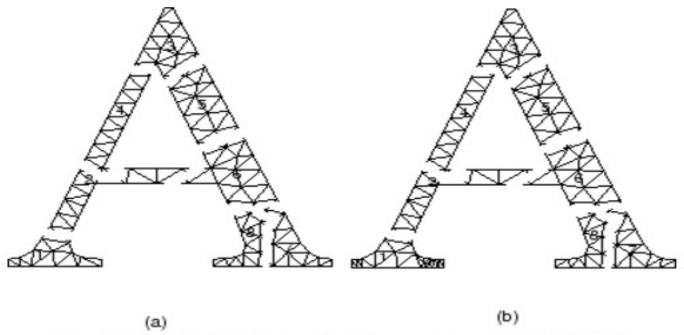

Figure 1 : (a) A mesh of ``A'' shape partitioned into 8 subdomains; (b) the mesh refined at subdomain 1

In general dynamic load balancing algorithms should satisfy the following objectives:

1.Re-balance the load of each processor with speed and scalability.

2.Minimize the edge-cut (or more precisely, the communication cost of the application after the re-balance)

In order to satisfy the first objective, the dynamic load balancing algorithm should not only identify what to migrate efficiently, the amount of data required to be migrated should also be kept to a minimum. Various metrics such as TotalV and MaxV [6] have been used to model and minimize the data migration cost.

One way to re-balance the load is to repartition the mesh using one of the partitioning algorithms . Indeed parallel algorithms such as JOSTLE or ParMETIS are able to partition large mesh very rapidly [7]. For example, ParMETIS was able to partition a mesh of the order of 1 million nodes in less than 2 seconds on 128 PEs of a Cray T3D [8]. However it is important, but difficult, to ensure that the new partitioning will be ``close'' to the original partitioning. Should the new partitioning deviate considerably from the old one then the cost of transferring large amounts of data will be incurred[8]. It has been found that repartitioning is more appropriate when there has been a substantial localized refinement on the mesh

An alternative strategy is to migrate the excessive nodes to neighboring processors, effectively shifting the boundaries to achieve a balanced load. This approach may potentially cause less movement of data than repartitioning, although the edge-cut after the migration could possibly be larger than that given by a global repartitioning. Therefore care must be taken to keep edge-cut down when choosing the nodes to be migrated. It has been found [12] that this strategy is more suitable when the load imbalances caused by the refinement are low, or when localized high imbalances occur throughout the mesh. This is because in such cases the optimal partition will be relatively close to the initial partition.

III Categories of Dynamic Load Balancing

A. Client-side random load balancing

Another approach to load balancing is to deliver a list of server IPs to the client, and then to have client randomly select the IP from the list on each connection. This essentially relies on all clients generating similar loads, and the Law of Large Numbers to achieve a reasonably flat load distribution across servers. It has been claimed that client-side random load balancing tends to provide better load distribution than round-robin DNS; this has been attributed to caching issues with round-robin DNS, that in case of large DNS caching servers, tend to skew the distribution for round-robin DNS, while client-side random selection remains unaffected regardless of DNS caching.

With this approach, the method of delivery of list of IPs to the client can vary, and may be implemented as a DNS list (delivered to all the clients without any round-robin), or via hard coding it to the list. If a "smart client" is used, detecting that randomly selected server is down and connecting randomly again, it also provides fault tolerance.

B. Server-side load balancers

For Internet services, server-side load balancer is usually a software program that is listening on the port where external clients connect to access services. The load balancer forwards requests to one of the "backend" servers, which usually replies to the load balancer. This allows the load balancer to reply to the client without the client ever knowing about the internal separation of functions. It also prevents clients from contacting back-end servers directly, which may have security benefits by hiding the structure of the internal network and preventing attacks on the kernel's network stack or unrelated services running on other ports.

Some load balancers provide a mechanism for doing something special in the event that all backend servers are unavailable. This might include forwarding to a backup load balancer, or displaying a message regarding the outage.

IV Types of Dynamic Load Balancing Algorithm

A. Diffusion algorithm

One of the most popular approaches to the flow calculation problem is to use diffusion based algorithms [6,14]. In a heat diffusion process, the initial uneven temperature distribution in space causes the movement of heat, and the system eventually reaches a steady-state temperature.

The diffusion algorithm, as described in [6], is given as follows. At each iteration k+1 of the algorithm, processor i will send an amount proportional to the difference between its load and its neighbor's load, $c_{ij}(l_i^{(k)} - l_j^{(k)})$, to its neighbor j Assume Cij=Cji, the new load $li^{(k+1)}$ of the processor i is given by the combination of its own load $li^{(k)}$ and contributions from/to its neighboring vertices, namely

$$l_i^{(k+1)} = l_i^{(k)} - \sum_{i \leftrightarrow j} c_{ij}(l_i^{(k)} - l_j^{(k)}), \quad i,\ j \in V,\ k = 1, 2, \ldots \tag{1}$$

Initially the load for vertex $i \in V$ is $l_i^{(1)} = l_i$. In matrix form, the above equation can be rewritten as

$$l^{(k+1)} = (I - A\,W\,A^T)\,l^{(k)}, \tag{2}$$

where W is a diagonal matrix of the size $|E| \times |E|$, that consists of the coefficients Cij, and A is a matrix of size $|E| \times |V|$, to be defined in Section 4.3.4. For the choice of the coefficients, Boillat [6] suggested

$$c_{ij} = \frac{1}{\max\{deg(i), deg(j)\} + 1}, \quad i \leftrightarrow j, \quad i,\ j \in V.$$

The diffusion algorithm, being a stationary iterative method of the form (11), can converge quite slowly on graphs with small connectivity. Boillat [6] proved that the worst case happens when the graph is, say, a line, and in such a case the number of iterations needed to reach a given tolerance is O(p^2) with p, the number of vertices. There are other variations of the diffusion algorithm ([]).A special case of the following equation,

$$(I + A\,W\,A^T)\,l^{(k+1)} = l^{(k)},$$

is solved in []. The convergence of the diffusion algorithm can also be improved using the Chebyshev polynomial [9]. Many investigations of dynamic load balancing algorithms have used a diffusive approach, although the details vary. For example, in both the tiling algorithm and the iterative tree balancing algorithm [11], a processor selects amongst its neighbors the one with the highest load and posts a request. In the tiling algorithm the amount of load to be sent is decided by looking at the average of the loads in the neighborhood. In the iterative tree balancing algorithm the requests are viewed as a forest of trees. The flow along the branches of the tree is then calculated using a logarithmic time parallel scan operation.

There are two type of diffusion algorithm

1. Sender Initiated Diffusion (SID)

The SID strategy is a, local, near-neighbor *diffusion* approach which employs overlapping balancing domains to achieve global balancing. for an *N* processor system with a total system load *L*, a diffusion approach, such as the SID strategy, will cause each processor's load to converge to *L/N.* [1,7,8]

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

2. Receiver Initiated Diffusion (RID)

(1) First, the balancing process is initiated by any processor whose load drops below a pre specified threshold ***(*** L_{Low} ***) . [7,8]***

(2) 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

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

B. Dimension Exchange Algorithm

Cybenko suggested a dimension exchange algorithm, in which the edges of the graph are colored so that no two edges of the same color share a vertex. Pairs of processors having the same color were grouped and a processor pair (i, j) with load l_i and lj exchange their load, after which each has the load (li+lj)/2. The algorithm was proved to converge in d steps if the graph considered was a hypercube with dimension d. Xu and Lau [10] extended the dimension exchange algorithm so that after the exchange processor i has load li*a+lj*(1-a). If a=0.5 this is equivalent to Cybenko's algorithm. Based on an eigen value analysis of the underlining iterative matrices, they argued that for some graph a factor a other than 0.5 gives better convergence. On a graph with small connectivity, this algorithm suffers in convergence in the same way as the diffusion algorithm.

C. Multilevel Algorithm

To speedup the diffusion algorithm, Horton [] suggested a multilevel diffusion method. The processor graph was bisected and the load imbalance between the two subgraphs was determined and transferred. This process was repeated recursively until the subgraphs could not be bisected any more. The advantage of the algorithm is that it is guaranteed to converge in log(p) bisections, and the final load will be almost exactly balanced even if the loads are integers. However, because it is not always possible to bisect a connected graph into two connected subgraphs, it was not clear from the paper how to proceed for such a case. Connectivity can of course be restored by adding new edges to a disconnected subgraph. However this is equivalent to moving data between non-neighboring processors and should be avoided.

Linear programming based algorithms

A possibly better model [] of the communication cost in the migration process, as opposed to (3), is the maximum cost of load migration over all processors, that is

$$\text{cost} = \max_{i \in E} \left(t_0 + \alpha\, |x_i|\right).$$

Here t_0 is the communication latency and α is the subsequent cost of communication per word. The flow calculation problem then becomes

$$\begin{array}{ll} \text{Minimise} & \{\max_{i \in E} (t_0 + \alpha\, |x_i|)\} \;, \\ \text{subject to} & Ax = b, \end{array}$$

which is equivalent to

$$\begin{array}{ll} \text{Minimise} & c, \\ \text{subject to} & Ax = b,\; c \geq (t_0 + \alpha\, |x_i|),\; i \in E. \end{array} \tag{3}$$

However it is not clear that an efficient parallel algorithm exists for such a linear programming problem.

Similar linear programming based flow calculation were proposed in [], where the problem

$$\text{Minimise} \sum_{i \leftrightarrow j} \delta_{ij}$$

$$\text{subject to} \sum_{i \leftrightarrow j} (\delta_{ij} - \delta_{ji}) = l_j - \bar{l}, \quad j \in V$$

$$0 \leq \delta_{ij} \leq \alpha_{ij}, \quad i, j \in V; \ i \leftrightarrow j$$

was solved. Here α_{ij} is the number of vertices on subdomain i that may be moved to sub domain j, using a node selection strategy based on layering (see the next section). This linear programming problem was solved using the simplex method to give the flow. The problem has 2|E| variables and |V|+|E| constraints. A multilevel approach was used to group subdomains into super-partitions, thereby breaking the linear programming problem into smaller ones to be solved by subsets of processors. This reduced the overall complexity of solving the linear programming problem.

D. Hierarchical Balancing Method (HBM)

It is an asynchronous global, approach which organizes the system into a hierarchy of subsystems. [1,7] • 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.

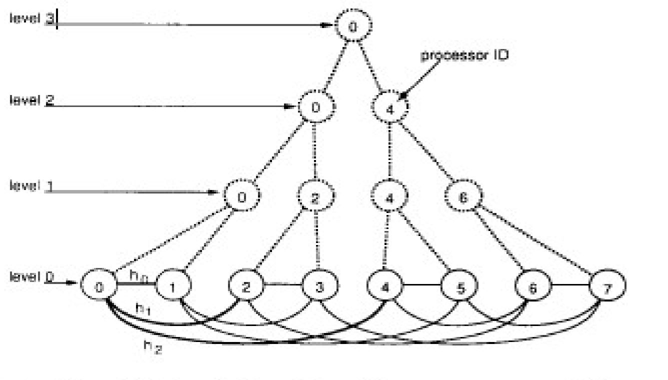

Fig. 3 Hierarchical organization of an eight-processor system with hypercube interconnections, where h_k is the connection to the neighbor at the kth level.The processor IDs at intermediate nodes in the tree represent those processors delegated to manage the balancing of corresponding lower-level domains.

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

E. The Gradient Model (GM)

The gradient model [5,13] is a demand driven approach .The basic concept is that underloaded 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 underloaded processors in the system to guide the migration of tasks between overloaded and underloaded processors.

The resulting effect is a form of relaxation where tasks migrating through the system are guided by the proximity gradient and gravitate towards underloaded 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.

F. Central Queue Algorithm:

This algorithm stores new activity and unfulfilled requests in a cyclic FIFO queue. Each new activity is inserted in the queue. Then, whenever a request for an activity is received the first activity is removed from the queue. If there is not any requested activity in the queue then the request is buffered until a new activity is available. This is a centralized initiated algorithm and need high communication among nodes.

G. Local Queue Algorithm:

This algorithm supports inter process migration. This idea is static allocation of all new process with process migration initiated by the host when its load falls under the predefined minimum number of ready processes. When the host gets under load it request for the activities from the remote hosts. The remote hosts than look up its local list for ready activities and some of the activities are passed on to the requestor host and get the acknowledgement from the host. This is a distributed co-operative algorithms requires inter process communication but lesser as compared to central queue algorithm.

H. Least Connection Algorithm : This algorithm decides the load distribution on the basis of connections present on a node. The load balancer maintains the log of numbers of connections on each node. The number increases when a new connection is established and decreases when connection finishes or time out. The nodes with least number of connections are selected first.

V Features of Load Balancer

Hardware and software load balancers may have a variety of special features. The fundamental feature of a load balancer is to be able to distribute incoming requests over a number of backend servers in the cluster according to a scheduling algorithm. Most of the following features are vendor specific:

Asymmetric load: A ratio can be manually assigned to cause some backend servers to get a greater share of the workload than others. This is sometimes used as a crude way to account for some servers having more capacity than others and may not always work as desired.

Priority activation: When the number of available servers drops below a certain number, or load gets too high, standby servers can be brought online.

SSL Offload and Acceleration: Depending on the workload, processing the encryption and authentication requirements of an SSL request can become a major part of the demand on the Web Server's CPU; as the demand increases, users will see slower response times, as the SSL overhead is distributed among Web servers. To remove this demand on Web servers, a balancer can terminate SSL connections, passing HTTPS requests as HTTP requests to the Web servers. If the balancer itself is not overloaded, this does not noticeably degrade the performance perceived by end users. The downside of this approach is that all of the SSL processing is concentrated on a single device (the balancer) which can become a new bottleneck. Some load balancer appliances include specialized hardware to process SSL. Instead of upgrading the load balancer, which is quite expensive dedicated hardware, it may be cheaper to forgo SSL offload and add a few Web servers. Also, some server vendors such as Oracle/Sun now incorporate cryptographic acceleration hardware into their CPUs such as the T2000. F5 Networks incorporates a dedicated SSL acceleration hardware card in their local traffic manager (LTM) which is used for encrypting and decrypting SSL traffic. One clear benefit to SSL offloading in the balancer is that it enables it to do balancing or content switching based on data in the HTTPS request.

Distributed Denial of Service (DDoS) attack protection: load balancers can provide features such as SYN cookies and delayed-binding (the back-end servers don't see the client until it finishes its TCP handshake) to mitigate SYN flood attacks and generally offload work from the servers to a more efficient platform.

HTTP compression: reduces amount of data to be transferred for HTTP objects by utilizing gzip compression available in all modern web browsers. The larger the response and the further away the client is, the more this feature can improve response times. The tradeoff is that this feature puts additional CPU demand on the load balancer and could be done by web servers instead.

TCP offload: different vendors use different terms for this, but the idea is that normally each HTTP request from each client is a different TCP connection. This feature utilizes HTTP/1.1 to consolidate multiple HTTP requests from multiple clients into a single TCP socket to the back-end servers.

TCP buffering: the load balancer can buffer responses from the server and spoon-feed the data out to slow clients, allowing the web server to free a thread for other tasks faster than it would if it had to send the entire request to the client directly.

Direct Server Return: an option for asymmetrical load distribution, where request and reply have different network paths.

Health checking: the balancer polls servers for application layer health and removes failed servers from the pool.

HTTP caching: the balancer stores static content so that some requests can be handled without contacting the servers.

Content filtering: some balancers can arbitrarily modify traffic on the way through.

HTTP security: some balancers can hide HTTP error pages, remove server identification headers from HTTP responses, and encrypt cookies so that end users cannot manipulate them.

Priority queuing: also known as rate shaping, the ability to give different priority to different traffic.

Content-aware switching: most load balancers can send requests to different servers based on the URL being requested, assuming the request is not encrypted (HTTP) or if it is encrypted (via HTTPS) that the HTTPS request is terminated (decrypted) at the load balancer.

Client authentication: authenticate users against a variety of authentication sources before allowing them access to a website.

Programmatic traffic manipulation: at least one balancer allows the use of a scripting language to allow custom balancing methods, arbitrary traffic manipulations, and more.

Firewall: direct connections to backend servers are prevented, for network security reasons Firewall is a set of rules that decide whether the traffic may pass through an interface or not.

Intrusion prevention system: offer application layer security in addition to network/transport layer offered by firewall security.

VI. MODEL FOR DYNAMIC LOAD SCHEDULING APPROACH

We have developed a general model for dynamic load balancing.

This model is organized as a four phase process[6,13]

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

A. 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**.**

B. 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**.**

C. Task Migration Strategy:

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

D. 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.

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

VII Conclusions

In this paper, dynamic load balancing strategies designed to support highly parallel systems have been presented and compared. The different strategies exemplify some of the main issues and tradeoffs that exist in dynamic load balancing, specifically in reference to highly parallel systems. Two major issues, that of load balancing overhead and the degree of knowledge used in balancing decisions were discussed. Also considered were, the concept of balancing domains, the aging of information, and the form of balancing initiation. Of the five strategies proposed, the DEM strategy tended to outperform the rest for all granularities. The efficiency of the DEM and the HBM strategies, depends heavily on the system interconnection topology. The hypercube topology is ideally suited to match these two strategies communication dependencies. Furthermore, the system sizes tested were very small in the context of highly parallel systems. The overhead of synchronization costs [scale as O(NlogN)] for the DEM approach and the aging period and non uniform overhead distributions of the HBM approach may deteriorate their performance when the number of processors is large (1000 processors). The RID strategy, on the other hand, is easily ported to simpler topologies, and can scale gracefully for larger systems. Finally, for a wider variety of applications, exhibiting local communication dependencies between tasks, the RID scheme is able to maintain task locality. Therefore, since its performance was shown to be comparable to those of the DEM and HBM approaches, the RID strategy may be best suited for a broader range of systems supporting a large variety of applications

References

[1] U. Karthick Kumar, "A Dynamic Load Balancing Algorithm in Computational Grid Using Fair Scheduling", International Journal of Computer Science Issues, Volume 8 ,Issue 5, September 2011.

[2] Bin Lu, Hongbin Zhang, "Grid Load Balancing Scheduling Algorithm Based on Statistics Thinking",9th International Conference for Young Computer Scientists,IEEE,2008.

[3] V.P Narkhede, S.T. Khandare, "Fair Scheduling Algorithm with Dynamic Load Balancing Using in Grid Computing", International Journal of Engineering and Science, Volume 2, Issue 10, April 2013.

[4] Fahd Alharbi, "Simple Scheduling Algorithm with Load Balancing for Grid Computing", Asian Transactions on Computers, Volume 2, Issue 2, May 2012.

[5] Junwei Cao, Daniel P. Spooner, Stephen A Jarvis, Graham R. Nudd, "Grid Load Balancing UsingIntelligent Agents", ACM, Volume 21, Issue 1, January 2005.

[6] Mohammad Haroon, Mohammad Husain, "Analysis of a Dynamic Load Balancing in Multiprocessor System", International Journal of Computer Science engineering and Information Technology Research, Volume 3, Issue 1, March 2013.

[7] Abhijit A. Rajguru, S.S. Apte, "A Comparative Performance Analysis of Load Balancing Algorithms in Distributed System using Qualitative Parameters", International Journal of recent Technology and Engineering, Volume 1, Issue 3, August 2012.

[8] Urjashree Patil, Rajashree Shedge, "Improved Hybrid Dynamic Load Balancing Algorithm for Distributed Environment", International Journal of Scientific and Research Publications, Volume 3, Issue 3, March 2013.

[9] M.A. Alam, K. Varshney, “A comparative study of interconnection network” Int. J. Comput. Appl., 127 (4) (2015), pp. 37-43

[10] M. Alam, A.K. Varshney A new approach of dynamic load balancing scheduling algorithm for homogeneous multiprocessor systemInt. J. Appl. Evol. Comput., 7 (2) (2016), pp. 61-75

[11] M. Alam, A. Khan, A.K. Varshney A review on dynamic scheduling algorithms for homogeneous and heterogeneous system Proceedings of the Springer International Conference of Computer Society of

India (CSI) of Transactions on ICT (2016)(in press)

[12] M. Dobber, R.V.D. Mei, G. Koole Dynamic load balancing and job replication in a global-scale grid environment: a comparisonIEEE Trans. Parallel Distrib. Syst., 20 (2) (2009), pp. 207-218

[13] D. Jain, S.C. Jain Load balancing real-time periodic task scheduling algorithm for multiprocessor environment",In Circuit, Power and Computing Technologies (ICCPCT), International Conference on IEEE (2015), pp. 1-5

[14] Z.A. Khan, J. Siddiqui, A. Samad A novel multiprocessor architecture for massively parallel System" Parallel, Distributed and Grid Computing (PDGC), International Conference on IEEE (2014), pp. 466-471

[14] W.M.H. LeMair, A.P. Reeves Strategies for dynamic load balancing on highly parallel computers IEEE Trans. Parallel Distrib. Syst., 4 (9) (1993), pp. 979-992

[15] K. Singh, M. Alam, S.K. Sharma A survey of static scheduling algorithm for distributed computing system"' Int. J. Comput. Appl., 129 (2) (2015), pp. 25-30