



Comparison of Dynamic Load Scheduling Algorithm for Multiprocessor Interconnection System

Mukul Varshney*, Aparajita Nailwal, Shivani Garg
CSE, Sharda University, Greater Noida,
Uttar Pradesh, India

Abstract— *In this era, the use and access of fast processors and also multi-processors has widely increased. Achieving Parallelism is now a necessity to improve the performance of computer system. One of the main Issues is how to effectively utilized parallel computer that have become increasingly complex. It is estimated that many modern supercomputers and parallel processor deliver only 10 percent or less of their peak performance potential in a variety of applications. Yet high performance degradation are many. Thus, in multi-processor systems, scheduling on a number of processors is of great interest for such purposes [1,2]. Parallel computers perform their computation by executing different computational tasks on a number of processors concurrently. During the execution of the parallel code, The processors within a parallel computer generally exchange information. This transfer of information occurs either in the form of messages sent by one processor to another or different parallel processors sharing a specified common memory resource within the parallel computer. The main idea of load balancing is to use multiple nodes to replace only one node, thereby it can enhance the computing power and reliability of the node at a low cost and time. Each individual node in load balancing system, which can provide services independently without any external support of other hosts, has equivalent status in the system. When one or a few nodes in the system fail, rather than being interrupted, the system will adjust scheduling scheme rapidly and assign a new service request to a normal working node, meanwhile, shift the unaccomplished task of the failure host to other hosts[7]. 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 [6]. Multiprocessor system must 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 scheduling (DLB) schemes are needed to efficiently solve non-uniform problems on multiprocessor systems. Some of the major benefits of parallel computing systems are information sharing among distributed users, resource sharing, better price/performance ratio, shorter response time, higher throughput, higher reliability, extensibility and incremental growth. Load balancing on multi computers is a challenge task due to the knowledge overhead and the inter processor communication overhead incurred in the collection of state information, communication delays, knowledge overhead, redistribution of load etc. Parallel and distributed computing environment is inherently best choice for solving/running distributed and parallel program applications. Dynamic load scheduling (DLB) algorithm are required to efficiently solve this problems on multiprocessor systems. [2,6]. In this paper our effort is concentrated on study and evaluation of various dynamic load balancing strategies such as SID, RID, DEM, GM HBM etc[7,15]*

Keywords— *Multiprocessor system, load balancing strategies, Sender initiated diffusion, knowledge over head, threshold, RID, HBM, GM*

I. INTRODUCTION

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.[8]

The trade off 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 Diffusion (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) [6,11] 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) [13,15], 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.

II. DESCRIPTION OF LOAD SCHEDULING PROBLEM

A. Why the Load Balancing Problem Occur In multiprocessor systems

Some of the nodes may be heavily loaded while others are lightly loaded. This suggests that it is possible to improve the overall performance of a multiprocessor system by transferring jobs from the heavily loaded nodes to the lightly loaded nodes. This type of processing power sharing is called load balancing, which can be classified into two categories: static and dynamic policies. A static policy uses only the information about the average behavior of the system to make job transfer decisions. A policy takes into account the current system state and react accordingly is called dynamic or adaptive policy. The multiprocessor system is a new form of parallel processing system. The paper targets at achieving reasonable task allocation in the system, making it operate at higher performance levels.

B. Basic Concept of Load Balancing

Dynamic load balancing algorithms can be classified into three categories: Sender Initiated (SI), Receiver Initiated (RI), and Periodically Exchanged (PE). [2,4] In the SI algorithms, a heavily loaded node initiates the load balancing by requesting the load information of other nodes and sending out its jobs to the lightly loaded nodes. In the RI algorithms, a lightly loaded node initiates The load balancing by sending job request messages to other nodes and waiting for remote jobs. In periodical exchange sender and receiver both initiating load balancing in a fix time interval so that load of the over all system will be balanced.

III. A GENERAL DYNAMIC LOAD SCHEDULING MODEL

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. 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.

B. 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.

C. 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.

D. 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.

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

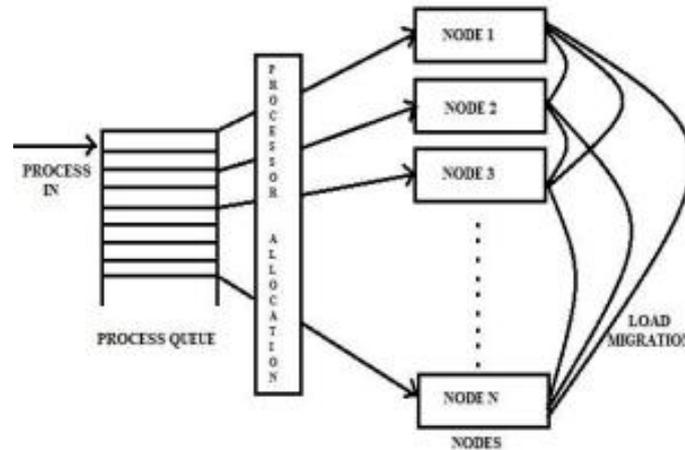


Fig. 1 Dynamic Load balancing to avoid on heavily loaded node

IV. DYNAMIC LOAD SCHEDULING ALGORITHMS

The following five **DLS** strategies [4,14] 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.

A. 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, $l_1 < \text{Ideal Load}$, 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

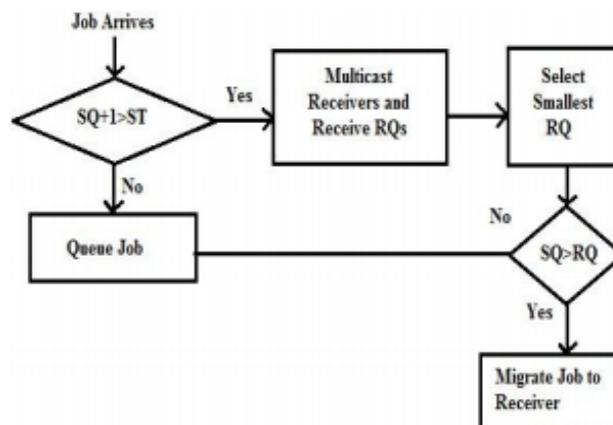


Fig. 2 Flowchart of Sender –Initiated Algorithm

SQ- Queue length of sender

RQ- Queue length of receiver

B. Receiver Initiated Diffusion (RID)

- (1) First, the balancing process is initiated by any processor whose load drops below a pre specified threshold (L_{Lo}).
- (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

C. 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.

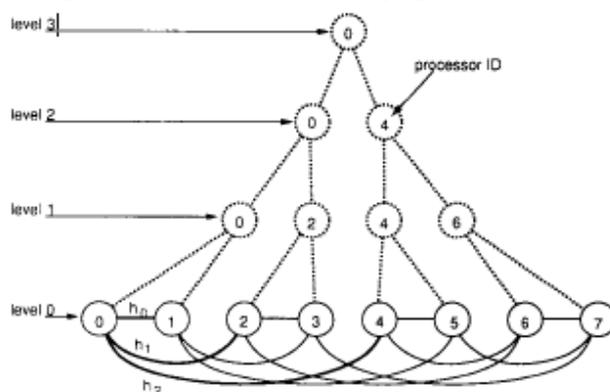


Fig. 3 Hierarchical organization of an eight-processor system with hypercube interconnections, where h_k is the connection to the neighbor at the k th level. The processor IDs at intermediate nodes in the tree represent those processors delegated to manage the balancing of corresponding lower-level domains.

- 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.

D. 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.

E. The Dimension Exchange Method (DEM)

The DEM strategy [15] is similar to the HBM scheme in that small domains are balanced first and these then combine to form larger domains until ultimately the entire system is balanced. This differs from the HBM scheme in that it is a synchronized approach. The DEM strategy was conceptually designed for a hypercube system but may be applied to other topologies with some modifications. In the case of an N processor hypercube configuration, balancing is performed iteratively in each of the $\log N$ dimensions. All processor pairs in the first dimension, those processors whose addresses differ in only the least significant bit, balance the load between themselves. Next, all processor pairs in the second dimension balance the load between themselves, and so forth, until each processor has balanced its load with each of its neighbors.

V. COMPARISON ANALYSIS OF ALGORITHMS

We observed from [6] that the Rendez-vous algorithm is convergent. On a „n“ processor system, at most $\log_2(n)$ are needed to reach a steady state where each processor has got the average load of the system as shown in table 1. Communication cost is taken into consideration for accurately evaluating the behavior of redistribution.

Table I Algorithm Convergence & Communication Cost

Algorithm	Speed of Convergence	Communication Cost	Communication Policy
Rendez-vous	$\sim \log_2(n)$	$O(n \log n)cg$	Global
Tiling	$> n/2$	$O(n)cn$	Local
X-Tiling	$\sim \log_2(n)$	$O(\log n)cu$	Uniform
Sliding	$< n$	$O(n)cn$	Local

A. Implementation Results

All five schemes were implemented on a hypercube multiprocessor architecture. The strategies were first tested and analyzed using artificially generated data consisting of a set of tasks with random computational requirements executed as busy loop.

Table II- Artificially Generated Task (task vs time)

No of Task	RID	SID	GM	DEM	HBM
16	4	0	4	4	4
32	4	2	4	4	4
64	4	5	5	5	5
128	5	5	5	5	10
256	5	6	6	10	11
512	6	6	6	11	11
1024	6	7	6	15	15
2048	10	10	10	16	16
4096	15	15	21	21	21
8192	20	28	37	43	43
16384	37	54	60	78	70
32768	78	94	114	147	146
65536	170	190	220	270	270
131072	433	341	411	451	448
262144	702	702	800	921	921
524288	1351	1471	1644	2021	2007
1048576	2612	7282	3385	3620	3434

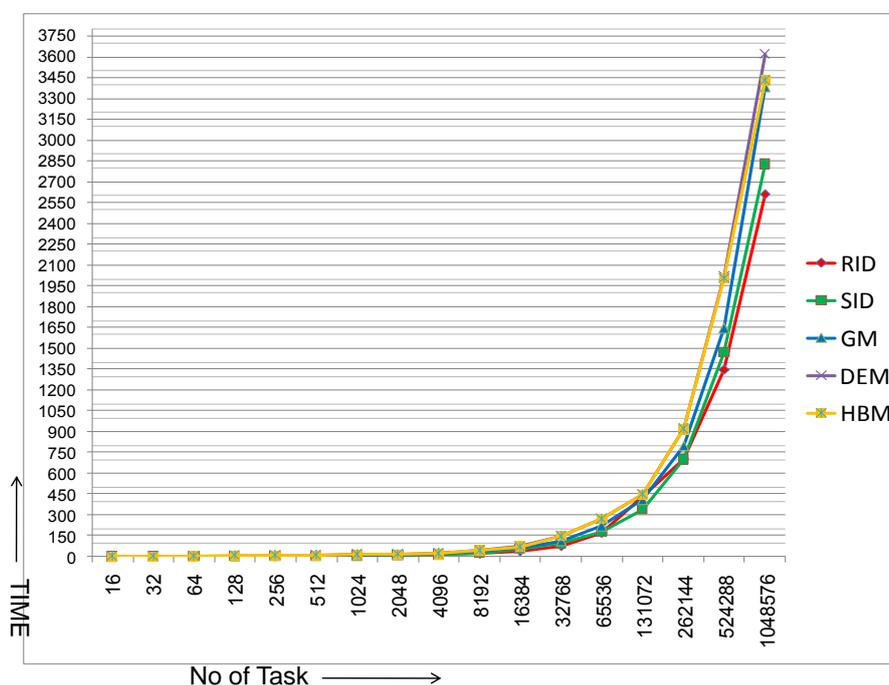


Fig. 4 Task Vs Time (Comparative Study)

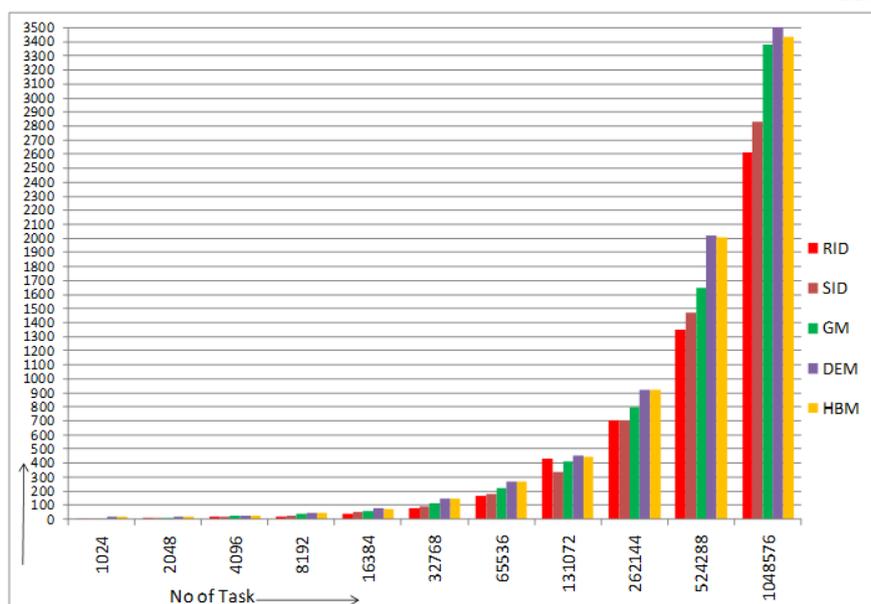


Fig.5. Task vs Time (Comparative Study)

VI. CONCLUSIONS

Five 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. Further- more, the system sizes tested were very small in the context of highly parallel systems. The overhead of synchronization costs [scale as $O(N\log N)$] 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] 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.
- [2] 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.
- [3] Bin Lu, Hongbin Zhang, "Grid Load Balancing Scheduling Algorithm Based on Statistics Thinking", 9th International Conference for Young Computer Scientists, IEEE, 2008.
- [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 Using Intelligent 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] Robson Eduardo De Grande, Azzedine Boukerche, and Hussam Mohamed Soliman Ramadan, "Measuring Communication Delay for Dynamic Balancing Strategies of Distributed Virtual Simulations", *IEEE Transactions on Instrumentation and Measurement*, Vol.60, No.11, November 2011.

- [10] Dennis Roubos and Sandjai Bhulai, "Session-Level Load Balancing for High-Dimensional Systems", IEEE Transactions on Automatic Control, Vol.54, No.8 August 2009.
- [11] Xiao Qin, Hong Jiang, Adam Manzanares, Xiaojun Ruan, Shu Yin, "Communication-Aware Load Balancing for Parallel Applications on Clusters", IEEE Transaction on Computers, Vol.59, No.1, January 2010.
- [12] Sun Nian, Liang Guangmin, "Dynamic Load Balancing Algorithm for MPI Parallel Computing", International Conference on New Trends in Information and Service Science (NISS), IEEE, Gyeongju, Korea, pp.95–99, June 30 - July 2, 2009.
- [13] Yongzhi Zhu, Jing Guo, Yanling Wang, "Study on Dynamic Load Balancing Algorithm Based on MPICH", World Congress on Software Engineering, IEEE, Xiamen, China, May 19-21, 2009.
- [14] Tarek Helmy, Fahd S. Al-Otaibi, "Dynamic Load-Balancing Based on a Coordinator and Backup Automatic Election in Distributed Systems", International Journal of Computing & Information Sciences, Vol. 9, No. 1, pp.19 27, ISSN: 1708-0460 (print) - 1708-0479 (online), April 2011.
- [15] Jie Chang, Wen'an Zhou, Junde Song, Zhiqi Lin, "Scheduling Algorithm of Load Balancing Based on Dynamic Policies", Sixth International Conference on Networking and Services (ICNS), IEEE Xplore Digital Library, Cancun, ISBN: 978-1-4244-5927-8, May 6, 2010.