International Journal on Emerging Technologies **1**(2): 48-56(2010) **ISSN : 0975-8364**

Dynamic load balancing in distributed and high performance parallel enterprise computing by embedding MPI and open MP

Sandip S. Chauhan, Sandip B. Shah and H.M. Rai

Deptt. of Computer Science, Gujrat Inst. of Technical Studies, Himat Nagar (GJ)
**Deptt. of Electrical Engg. NCC Israna (HR)*

ABSTRACT : Load balancing involves assigning task to each processor to achieve higher performance and, minimizing the execution time of the application. Although static load balancing can solve many problems for most regular applications but, the transient external load due to multiple-users on a network of workstations necessitates a dynamic approach to load balancing. Experiments shows that different schemes are best for different applications under varying program and system parameters. Therefore, dynamic load balancing schemes become essential for achieving higher performance. In this study, a cluster-computing environment is employed as a computational platform. Clusters of SMP (Symmetric Multi-Processors) nodes provide support for a wide range of parallel programming paradigms. In order to increase the efficiency of the system, a dynamic task scheduling algorithm is proposed. The technique is dynamic, adaptive and, it uses divide and conquer approach. The algorithm models the cluster as hyper-grids and then balances the load among them. Recursively, the hyper-grids of k-dimensions are divided into grids of dimensions k ? 1, until the dimension becomes 1. Then, all the nodes of the cluster are almost equally loaded. The optimum dimension of the hyper-grid is chosen in order to achieve the best performance. The simulation result shows the effectiveness of the algorithm.

Keywords : CC - Computational Cluster, MPI -Message Passing Interface, Open Mp - Open Message Passing, PSLB - Positional Scale Load Balancing, NOW - Network of Workstations. IPC - Inter Process Communication

I. INTRODUCTION

Modern Scientific computing problems in many domains, e.g., Computational Fluid Dynamics *(CFD)*, *VLSI* simulations, Ocean Modelling and, many high performance encryption Algorithms involve large amount of data and are based on the concept of a large-scale spatial grid. A natural and efficient way to execute these types of applications is to follow the *Single Program Multiple Data (SPMD)* approach, i.e., to distribute the data on the spatial grid into multiple processors, each of which is assigned with a partition of the N-Dimensional grid onto multiple processors, each of which is assigned with partition of grid. Load balancing is a vital factor in achieving high parallel efficiency especially on platforms with a large number of processors. The amount of work assigned to each processor has to be determined such that turnaround time is minimized. For a parallel application running on a large number of processors, the turnaround time is defined as the maximum of all the times taken by the individual processors to complete the task. Static load balancing techniques are frequently employed to distribute the task on the set of processors to obtain the minimum turnaround time for the application. Total efficiency of resource utilization (constantly using 100% of all the processors) is rarely achieved, however, due to dependencies among the work units on the various processing nodes. Exogenous factors could also affect the computation. For instance, there may be other computations running on the same system, or there may be non-negligible communication time between the processing nodes. Even if load-balancing is used, exogenous factors could render static load-balancing ineffective. In order to achieve effective adaptability, the computation ought to employ dynamic load-balancing. That is, the computation must be able to reconfigure its processing nodes while it runs.

A. Computational cluster

A computational cluster *(CC)* can be defined as a set of independent nodes or computers interconnected by a high-speed communication network such as Fast or Gigabit Ethernet [1]. The number of the participating processing elements or nodes can range from tens to hundreds. However, to fully and effectively exploit any *CC* platform, resource management software must be provided to manage the complexity of different physical architectures for the user. This complexity arises in managing communication, synchronization and scheduling a large number of tasks, in dealing with portability of libraries facilities used to parallelize/distribute user applications, processor speed, available memory, length of the current run queue, percentage of idle time in the recent past, number of recent network interrupts, etc,. The scheduling of the submitted tasks to processing-nodes is a major concern with regard to performance and effective use of any *CC*. Although Graph Partitioning and Heuristic methods provide fast but often sub-optimal solutions within an acceptable time, where an optimal solution cannot be obtained within reasonable time. *Shen and Tsai[4]* proposed a method where problem of load balancing by optimal task assignment is viewed as a graph-matching problem. The task is represented as a

vertex and the communication between these modules represented by edges. The weight associated with the vertices represents the communication cost between two adjacent vertices of the task graph. The work done by *Shen* and *Tsai* uses a *heuristic* approach based in *A* Algorithm [4]*. The problem of finding an optimal solution to the scheduling problem is *NP-complete [1, 6, 8]* where *heuristic* methods appear to be a suitable approach to solve this class of problems.

B. Contribution and organization

This paper presents a method of dynamic load balancing with *MPI, OPEN MP [7, 9]* which combines methodologies, graph-partitioning and graph-matching, to achieve maximum parallel efficiency on for computing clusters. A location based scheme is proposed and evaluated which is based on previous PSLB algorithm, a pure dynamic load balancing technique. The experimental result of proposed technique shows that it is highly parallel and efficient.

II. BASIC CONCEPTS OF LOAD BALANCING

The load balancing strategies are classified on three parameters addressing initiation (sender or receiver), load balancer location (centralized or distributed) and decision–making (local or global). The speed at which a NOW-based parallel application can be completed depends on the computation time of the slowest workstation; efficient load balancing can clearly provide major performance benefits [8, 10].

A. The major categories for load-balancing algorithms are:

(I) Static load balancing: Static(Compile-time) load balancing algorithms allocate the tasks of a parallel program to grid based on either the load at the time nodes are allocated to some task, or based on an average load of cluster grid. In static scheduling, information regarding tasks' execution times and nodes' resources is assumed to be known beforehand.

(II) Dynamic load balancing algorithms: Dynamic load balancing algorithms makes changes to the work distribution at run-time among cluster. This technique takes into account over-loaded and under loaded nodes, with the assumption that if the load among all nodes is balanced, then the overall execution time of the application is minimized. It uses current load information when making load balancing distribution decisions.

B. Issues to be considered for dynamic load balancing

(I) Load estimation policy: Determines how to estimate the workload of a particular node of the system.

(II) Process transfer policy: Determines whether to execute a process locally or remotely.

(III) State information exchange policy: Determines how to exchange the system load information among the nodes.

(IV) Priority assignment policy: Determines the priority of execution of local and remote processes at a particular node.

(V) Migration limiting policy: Determines the total number of times a process, can migrate from one node to another.

C. Load balancing strategies: There are three major load balancing strategies:

(I) Sender-Initiated vs. Receiver-Initiated Strategies means-Who makes the load balancing decision.

(II) Global vs. Local Strategies means-What information is used to make the load balancing decision.

(III) Centralized vs. Distributed Strategies means- Where the load balancing decision is made.

III. PROBLEM STATEMENT

A. Problem intuition

Developing a solution for Load balancing into a cluster computing environment which incurred a less overhead compare to the prior techniques that have been proposed. The algorithm not only provides a perfect load balanced system at very reasonable time and, reduces significant overhead, but also should minimize the cost of Inter-process communication *(IPC) [1, 3, 6, and 15].* The technique should also be an adaptive based on the current changes in topology and, offer high degree of parallelism and, efficient utilization of the system resources in general.

B. Related work

In this section we will look at some of the load balancing schemes which have been proposed in the literature. A large number of load balancing and task assignment techniques have been proposed the classical example of this is the work done by Shen and Tsai which uses the well-known *A** algorithm to find optimal task assignment, a Positional Scan Load Balancing algorithm (PSLB) [11,17] which is originally the parallel version of an actual *A** algorithm, Optimal Load Balancing by Issac [14], Customized Dynamic Load Balancing for a Network of Workstations by Mohammed Wei Li, and many more techniques have been proposed in the literature which motivates us to work further in this area of dynamic load balancing. Moreover, different schemes are best for different applications under varying program and system parameters

such as the number of processors, data size, iteration cost, communication cost, etc., therefore, dynamic load balancing becomes essential for good performance.

C. Dynamic scheduling

Predicting the Future: A common approach taken for load balancing on a network of workstation is to predict future performance based on past information. The main contribution of this paper is the methodology for automatic generation of network topology with dynamic load balancing. CHARM implements a local distributed receiver-initiated scheme. If the work-load falls below a *Threshold [1, 18]*, the node requests a neighbour with higher work-load for more work. Much past work has explored the broad problem of dynamic load balancing. Various studies have examined synchronization, task migration, and cost for *IPC [19]*, Processing Power, number of CPU cycles, available memory and, many other issues. However, the consensus is that there is no one silver bullet and various techniques are best implemented on an application-specific basis. To consider some state-of-the-art techniques, one could apply optimistic technique to increase parallel performance.

IV. SYSTEM MODEL & HYPER-GRIDS

- A Computational Cluster (CC) is a collection of independent processing-nodes interconnected by a network.
- Each node ***vi*** is autonomous, has full information on its own resources, and manages its work load.
- Each node ***vi*** has a processing power ***ti*** which represents the number of work units that can be executed per unit of time.
- The network uses a packet switched protocol and let ***w*** be the size in bits of a packet, which is constant.
- The network's flow ***bij***, which is the effective data rate in bits per second on the link that connect the nodes ***vi*** to ***vj***.
- The tasks are independent and can be executed on any node regardless its initial placement.
- There are two parameters associated with each task (***ti***:1) the number of work units (in terms of computations) within the task **(ßi)**, and 2) the number of packets required to transfer the task **(µi)**.

A. Hyper-Grid

Usually, a computational cluster has an irregular topology. This topology can be described by non-oriented graph *G (V, E) [1, 7]*, where *V* represents the cluster nodes and *E* the set of links between nodes. The first phase of this technique is to map the graph *G (V, E)* into a multi-dimensional grid, called hyper-grid. The resulting grid is usually incomplete, in the sense that some of the links between neighbours and/or nodes are missing. The missing links and nodes are called *virtual links* and *virtual nodes* respectively. The second phase is to recursively dividing the original hyper-grid into hyper-grids of smaller dimensions. The idea is to balance the load among the hyper-grids dimensions starting from 1-dimension. An **n**-dimensional grid (***Gn***) can be defined as a set of ***n" 1***-dimensional parallel hyper-grids as follows:

$$Gi = (Gi - 1,\ Gi - 2,\ Gi - 3..., Gi - pi)\ i = n \quad ...(1)$$

The hyper-grids of dimension 1 represent the nodes along one dimension (e.g. nodes connected by bus) and from the equation 1 one can deduce that the number of nodes of Gn is $N = i - 1\ \Pi n - 1pi$ therefore, we can define a hyper-grid recursively as follows :

Definition : An n-dimensional hyper-grid is a set of parallel $(n - 1)$-dimensional hyper-grids. Zero-dimensional hyper-grids are the nodes of the system which are connected by links with the following properties:

- Links are either pair-wise vertical or parallel (links which lie on the same line or are parallel lines), and
- The length of links that connect direct neighbour nodes is the unity and it is constant.

Any node of the system can be represented as Vi1, i2, ..., *in*, for clarity reason, let denote by ***I*** the vector [*I*1, *I*2, ..., *In*]. The dynamic task scheduling technique introduced in this paper has a phase that computes a one dimensional vector of loads of task hyper-grids. These loads are then balanced across the linear array of processor hyper-grids. Definition 4.2 A hyper-grid load ***Wx*** is the number of active tasks stored in nodes that are within hyper-grid of dimension ***x,*** called **x-hyper grid**. This value is calculated by each processing-node for each hyper-grid that intersects it.

B. Task Scheduling and allocation

Positional Scan Load Balancing algorithm (PSLB) [11], leads to a perfect load balanced system at a very reasonable time. Algorithm preserves the locality decomposition and it is based on the parallel prefix operation, or scan [4], which can be defined as follows:

Definition : The prefix sum operation **(+, A)** takes the binary associative operator **+**, and an ordered set of n elements ***A = {a0, a1, . . . , an–1}***, returns the ordered set

$$\{0,\ a0,\ (a0 + a1),\ (a0 + a1 + a2),\ ...,\ (a0 + a1 ... + an - 2)\}$$

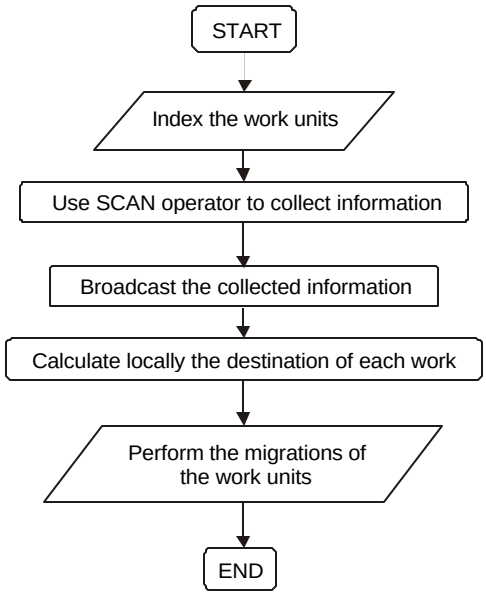
C. PSLB algorithm

The basic PSLB algorithm applies to 1-dimensional data grids. PSLB algorithm is a very powerful dynamic load

balancing algorithm, operating at the fine grain level. The generalization of the algorithm to n-dimensional data-grids is also introduced in [11]. A brief description of the PSLB algorithm for grids of one dimension (line or bus) of system networks is given in algorithm 1.

(*a*) *PSTS Algorithm*

In order to schedule more general applications (tasks) executing on irregular network topologies, we propose a technique based on PSLB, called the Positional Scan Task Scheduling (PSTS). PSTS [1] approaches the task scheduling by applying the same technique as PSLB. PSTS uses the additive scan operation in order to find out the destination node for each work unit within each task. Firstly the algorithm indexes the work units (not the tasks), then uses the scan operator to collect information about the load in the system and processing powers, and finally for each node calculates locally the destination of each work unit. The key issue here is that, instead of considering a work unit as a basic processing unit, a task, consisting of many work units, is considered as a basic element. In other words, a task is a non-divisible load however, the algorithm uses the work units to decide whether a task has to be migrated or not.



Algorithm 1 PSLB Algorithm -

Let $T = \{t1, t2, ..., tm\}$, represents the set of active tasks in the system, which consists of nodes $\{v1, v2, ..., vn\}$. The total load in the system is

$$W = \sum_{i=0}^{m} \beta i \qquad ...(2)$$

As the system is heterogeneous, each node vi has different processing power, $f\tilde{N}i$, and is given as the number of work units can be executed per unit of time. So, the total processing power of the system is

$$\pi = \sum_{i=0}^{m} \tau i \qquad ...(3)$$

In the algorithm, we utilize the normalized quantities. Let $\pi = \Sigma_{i=0}^{m} \tau i$ the total processing power of the system, the normalized processing of a node vi, $?_i = t i/\Pi$. Therefore in a perfect load balanced system, the load of each node, according to the equations 2, and 3 is given by

$$Wi = W * \frac{\tau i}{Vi} \Pi = W\gamma i \qquad ...(4)$$

The goal of the PSTS algorithm is to move each task t_i, from its current location v_i, to a node v_j, $v_j = F(t_i)$, so that the whole system is well balanced, and therefore the response time of different active tasks in the system is minimized $R(t_i, v_i \rightarrow v_j)$.

PSTS algorithm, firstly calculates the exclusive additive scan of the work units of all tasks on the hyper-grids G^1 of dimension equal to 1.

$$S1r = (+ L1r), r\ 1, 2,... ... ..., p1 \qquad ...(5)$$

Where $L1r$, is a vector of elements representing the number of work units of a node belonging to the hyper-grid $G1\ x$. Thus, $(+, L1r)$ is performed concurrently for all the hypergrids of a dimension equal to one. In the same way, this operation is performed concurrently on hyper-grids of the same dimension,

$$Spr = (+, Lpr), \quad r = 1, 2, ..., pp \qquad ...(6)$$

The resulting vector S_{p*}. Represent the exclusive additive scans for all hyper-grids of the same dimension p. The total work load W in the system is calculated by the operation $(+, S_{nr})$ on the n-dimensional hyper-grid. The scan operation is also used for determining the relative processing powers of the hyper-grids of different dimensions:

$$\lambda pr = (+, \lambda pr), r = 1, 2, ..., pp \qquad ...(7)$$

This implies that each hyper-grid knows the normalized processing power of its hyper-nodes. The next step is to calculate the destination of each task within the hyper-grids of the same dimension. Each task t_i, according to its initial placement, is assigned to a node v_i, $v_i = {}_{\text{Finit}}(t_i)$, and let $F(t_i) = v_J$. Then, the problem consists of calculating the index of the destination node vJ, $J = [j1, j2, ..., jn]$.

The calculations start from the highest dimension n and continues until the dimension is 1. In order to balance the load in the system, algorithm calculates the least index $\lambda n <= i/W$. This means that the algorithm works according

to the relative power of each hyper-grid and the sum of the work load of the entire system. After these scans each 1- dimensional hyper-grid knows whether is a "**receiver**" or a "**sender**". A receiver (resp. sender) means that a hyper-grid is under-loaded (resp. over-loaded). If, for instance, a 1-dimensional hyper-grid has to receive additional tasks then its own work consists of balancing its own workload according to the PSLB algorithm and just wait to receive more tasks which will go to the appropriate nodes. On the other hand, if it is over-loaded, then it uses the PSLB algorithm to balance its own workload (only for the tasks that have to remain in the hyper-grid), knowing that the hyper-grids of higher dimensions will balance the tasks among their elements, and therefore will migrate the extra tasks to their appropriate hyper-grids of lower dimension. This procedure guarantees that after its completion the entire system will be as close as possible to the perfect load balanced state. The description of the PSTS algorithm is given in algorithm 2.

Algorithm 2 Positional Scan Task Scheduling Algorithm

1. repeat
2. $r = 1$
3. for all r-dimensional hyper-grids in parallel do
4. $Srq \leftarrow (+, L_{rq})$, $q = 1, 2, ..., pr$
5. $?rq \leftarrow (+, t_{pr})$, $q = 1, 2, ..., pr$
6. end for
7. $W_r \leftarrow S_r + n_{i1, p1-1, i3 ......, ir.}$ ($n-1$ times)
8. $? r \leftarrow ?r + t_{i1, p1 "1, i3, ..., ir.}$ (n–1 times)
9. $r = r + 1$
10. *until* $(r = n - 1)$
11. $W \leftarrow \sum_{i=1}^{n-1} Wi + n_{p2, p1 - 1, p3, ..., pn(n-1\ times)}$
12. $?r \leftarrow \sum_{i=1}^{n"1} ? i + t_{p2, p1 - 1, p3, ..., pn}$ (n–1times)
13. for all 1-dimensional grids in parallel do
14. Calculate, using the PSLB algorithm, if the 1-dimensional grid is a sender or a receiver
15. if any 1-dimensional grid has to migrate tasks then
16. Apply the PSLB algorithm for the destination 1-dimensional grid, and
17. Migrate the tasks to the appropriate nodes.
18. else
19. Apply the PSLB algorithm for its own workload
20. end if
21. end for
22. End.

D. PSTS Performance model

Let d denote the dimensionality of the grid. If $d = 1$ (bus topology) the number of communication steps needed is $_{comm} S^1 = 2(n - 1)$, where n is the number of the participating nodes, (see Figure 1). The number of computation steps is $_{comp} S^1 = 2(n - 1)$.

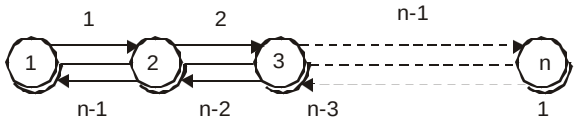


Fig. 1. d = 1 Comm. And Comp. Steps.

Let p and q be the costs in time units of a communication and a computation step respectively, then the total cost of the algorithm can be expressed as follows :

$$S1n = S1comm + S1comp$$
$$= 2(n - 1)p + 2(n - 1)q$$
$$= 2(n - 1)\ (p + q) \quad ...(8)$$

For $d = 2$, the network topology is a grid consisting of n1lines and n2 columns where $n = n1 \ . \ n2$ (see Figure 2).

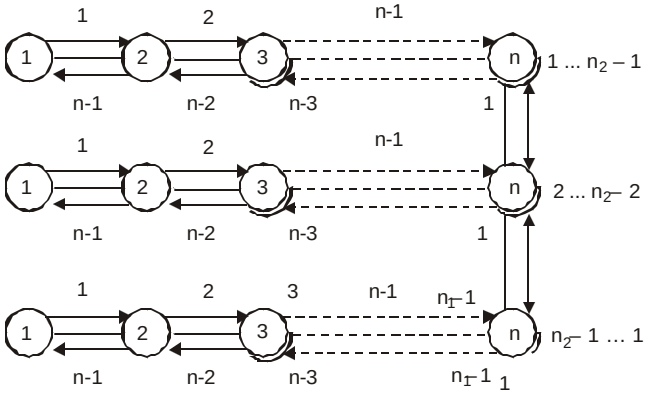


Fig. 2. d = 2 Comm. and Comp. Steps

The number of communication and computation steps needed for each line (of $n1$ nodes) is given by the equation 8. $2(n1 - 1)p + 2(n1 - 1)q$. This corresponds to balancing the load along each line of the grid (or hyper-grid of one dimension). Balancing the load along the columns can be done by performing the algorithm on hyper-grid of dimension 2 (by considering each line as a hyper-node of that hyper-grid). Therefore, the total cost for a 2-D grid is:

$$S2 = S2 \text{ comm} + S2 \text{ comp}$$
$$= 2(n1 - 1)p + 2(n1 - 1)q + 2(n2 - 1)p + 2(n2 - 1)q \quad ...(9)$$

Finally, for $d = 3$ (figure 4.3), the total cost of computation and communication steps needed is:

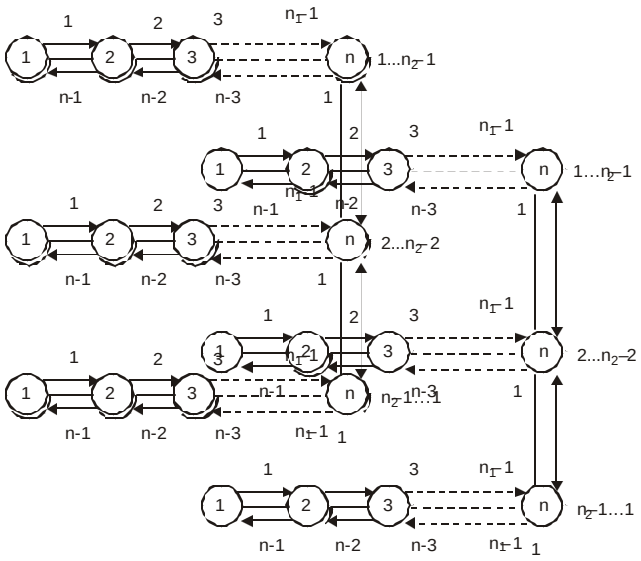


Fig. 3. d = 3, comm.. and comp. steps

$$S3 = S3\ \text{comm} + S3\ \text{comp}$$
$$= 2(n1 - 1)p + 2(n1 - 1)q + 2(n2 - 1) + 2(n2 - 1)q + 2(n3 - 1)p + 2(-1)q \quad ...(10)$$

or

$$S3 = S3\ \text{comm} + S3\ \text{comp} = 2\ (n1 + n2 + n3 - 3)\ (p + q)$$

and consequently for $d = k$

$$Sk = Sk\ \text{comm} + Sk\ \text{comp} = 2(n1 + n2 + ... + nk - k)\ (p + q)$$

E. Embedding Irregular Network into N-Dimensional Grid

There are many ways of embedding an irregular network topology $G(V, E)$ into an n-D grid. The resulting grid is called incomplete and contains two types of nodes and links. Nodes (resp. links) which are mapped onto V(*resp. E*) elements are called active nodes (resp. links). The nodes (resp. links) which are not assigned to any element of V (resp. E) are called virtual nodes (resp. virtual links). In order to ensure that the algorithm described above works on an incomplete grid, the virtual nodes are considered as active node with zero processing power. In the same way we can consider the virtual links as active links with zero bandwidth. In order to minimize the cost of the PSTS algorithm on an incomplete grid, one need to minimize the number of virtual nodes or the dimension of the corresponding gird. Proposition 4.1. Consider a network $G(V, E)$ consisting of n nodes. The best performance of PSTS is achieved when $G(V, E)$ is mapped onto a [log2(n)]-D grid [proof [3]].

V. EXPERIMENTAL RESULTS AND DESCUSSION

The algorithm that's proposed and discussed in previous chapter is implemented on cluster environment which is actually a modification of original *Positional Scan Task Scheduling (PSTS) Algorithm* which uses *Divide and Conquer* approach for load balancing. Generally the problem of load balancing is *NP-Complete* problem. This technique first recursively partitions and *N - Dimensional* grid into a *1-Dimentional* grid then applies *SCAN - Operator* [10,20] to which keeps tracks of current load information of grid and transfers this information to the broadcaster which is responsible for the load balancing in a grid of cluster. The broadcaster also computes locally the transfer of the load of grid to another grid based on the *Threshold* values if load exceeds the *Threshold* [1] then that node becomes heavily loaded and requires to be off-load and if the node has a work load below threshold value then that node becomes lightly loaded node and this information is noted by the broadcaster using the SCAN operator and the broadcaster will transfer the load to this lightly loaded nodes. The technique is *Adaptive, fully dynamic* and *non-per-emptive* which incurred a small amount of overhead to achieve fully dynamic load balanced cluster.

A. Results

To test the methodology, experiments are implemented in *Sequential* and parallel with the use of *P-THREAD, MPI* and *Open MP* on cluster of grid. In the following text, description of test cases is given, and then based on data from the given test case follows the result test, which include comparison between sequential and parallel with the use of *Positional Scan Load Balancing (PSLB)* algorithm in cluster of two-way grids and also the experiments analyzed with *V-Tune* Performance Analyzer by *Intel*$^{(R)}$.

(*a*) Test case

A 2 node SMP cluster is used for testing. Every node is equipped with Intel$^{(R)}$ Core$^{(TM)}$ 2 Duo Processor clocked at 1.83 GHz. Each node has 1 GB memory resulting in a total of 2 GB RAM for whole cluster. Both nodes are connected using shared memory and a high speed switch in (LAN).

(*b*) Tests with sequential technique

The algorithm which is proposed is implemented on a single machine with the Dual Core machine with the number of work unit as shown in Table 1 and, the estimated program execution time is also listed out for both Sequential and parallel execution using P-Threads.

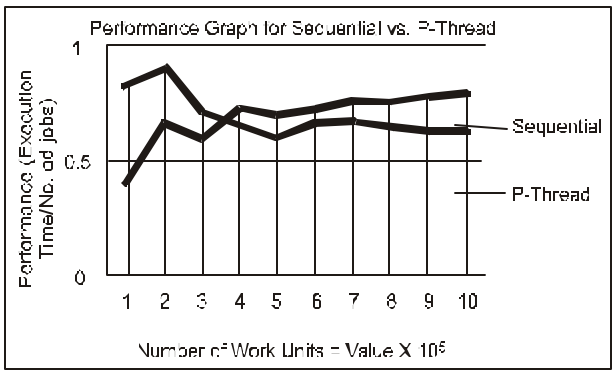


Fig. 4. Performance Graph for Sequential and P-Thread implementation

Table 1. Execution time in number of seconds for both sequential and parallel using (P-Thread) implementation.

Sr. No.	Number of work Units	Execution Time (seconds)	
		Sequential	P-Thread
1	100,000	1.202	2.407
2	200,000	2.405	2.945
3	300,000	4.228	4.982
4	400,000	6.230	5.550
5	500,000	8.235	7.125
6	600,000	8.923	8.146
7	700,000	10.265	19.988
8	800,000	12.298	23.571
9	900,000	14.263	24.615
10	10,00,000	15.938	25.710

C. Tests with proposed algorithm and MPI and Open MP

Now, it is attempted to use of graph matching to obtain load balancing across a cluster comprising of a two-way SMPs. This amounts to re-assigning of tasks to each node in such a manner that the sum of the computation and communication time for all processor in the node remains same. First, proposed algorithm is applied with *P-THREAD*. Two threads are created and task modules are assigned to each thread, than both threads are executed parallel to get optimal execution time, which are presented in Table – 2. Finally, the same the technique is used with *MPI* and *Open MP* to parallelize dynamically with 2 processors for achieving a load balancing and higher performance [9, 28].

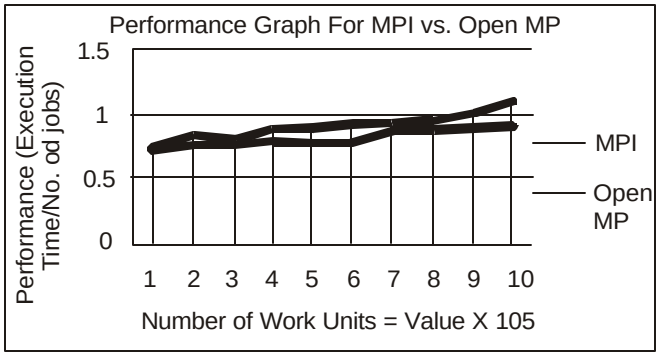


Fig. 5. Performance Graph For MPI and OpenMP implementation

Table 2. Execution time in MPI and Open MP

Sr. No.	Number of work Units	Execution Time (seconds)	
		MPI	Open MP
1	100,000	1.331	1.338
2	200,000	2.564	2.432
3	300,000	3.816	3.003
4	400,000	5.0426	4.925
5	500,000	6.342	5.553
6	600,000	7.616	6.451
7	700,000	8.943	6.495
8	800,000	10.383	7.541
9	900,000	11.619	8.596
10	10,00,000	13.057	10.121

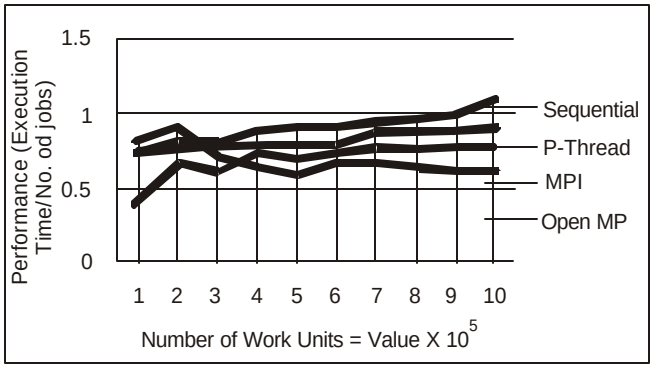


Fig. 6. Combined Performance Graph For Sequential and P-Thread MPI AND OpenMP implementation

Table 3. Execution time in Sequential, P-Therad, MPI and Open MP

Sr. No.	Number of work Units	Execution Time (seconds)			
		Sequ.	P-Thread	MPI	OpenMP
1	100,000	1.202	2.407	1.331	1.338
2	200,000	2.405	2.945	2.564	2.432
3	300,000	4.228	4.982	3.816	3.003
4	400,000	6.230	5.550	5.0426	4.925
5	500,000	8.235	7.125	6.342	5.553
6	600,000	8.923	8.146	7.616	6.451
7	700,000	10.265	19.988	8.943	6.495
8	800,000	12.298	23.571	10.383	7.541
9	900,000	14.263	24.615	11.619	8.596
10	10,00,000	15.938	25.710	13.057	10.121

A Speed – up factor denoted by μ is introduced to quantify the result. The optimality index is defined as :

$$\text{Speed} - \text{up} = \frac{\text{ExecutionTime(Sequential)}}{\text{ExecutionTime(Parallel)}} \quad ...(5.1)$$

B. Analysis using Intel(R) V-Tune Performance Analyzer

Finally, the parallel implementation is tested with Intel(R) V-Tune performance Analyzer which provides the information about behavior of different function used. This information is used to find the maximum Speed-Up that can be achieved using proposed algorithm and using parallel implementation.

Performance gain is: As the Fig 4 shows that the only 23.88% of process running sequentially so Scalar fraction a can be calculated as follows:

$$\alpha = 23.88/100 = 0.2388$$

$$\alpha = 0.2388$$

Now, According to Amdahl's Law, Speed-up can be calculated using following equation:

$$\text{Speed} - \text{up} = \frac{1}{\alpha \frac{1-\alpha}{p}} \quad \text{...(5.2)}$$

Where a = 0.2388 and *p* = No. of Nodes in SMP = 2, so Speed-up

$$\text{Speed} - \text{up} = \frac{1}{0.2388 \frac{1-0.2388}{2}} \quad \text{...(5.3)}$$

$$\text{Speed} - \text{up} = 1.6144.$$

C. Discussion

The result presented in Table-1 is based on single processor and in Table-2 is based on a two-way SMP node. In the result, the advantage of the POSITIONAL SCAN TASK SCHEDULING (PSTS) Algorithm methodology is clearly evident. The program calculates the actual time using timeval, timezone structure and using gettimeofday() function built in UNIX <time.h> library header file, this represents the actual time taken from launch of the parallel job to its completion. The parallel tasks are programmed using P-thread, MPI and, Open MPI (Open Message Passing Interface) and the utility mpirun –np # Output_ filename [10] was used to launch the jobs. np parameter indicates the number of processes. # sign indicates the number of nodes on which the job will executes. Here we can also provide a file name named as .Profile which contains all the information about nodes and there port addresses if more number of nodes are participating in task execution. Finally the file name to be executed on different node. Though the size of the state-space reduces drastically because of parallel implementations, the overheads incurred in communication over the network and mpirun to launch and terminate processes actually increase the turnaround time. This implies that the methodology shall be effective for cases with large number of task modules where the speed-up due to reduction is state-space size makes up for these overheads.

VI. CONCLUSION

In this project, the load balancing and, parallel task assignment policies for cluster computing environment and distributed network are discussed. Also, some existing load balancing techniques and their drawbacks and possible kinds of solutions are discussed. A perfect and low-Inter Process Communication(IPC) cost based load balancing technique which is based on Positional Scan Load Balancing (PSLB) and its modification Positional Scan Task Scheduling (PSTS) is implemented, which makes use Divide and conquer approach to partition the N-Dimensional grid recursively in 1-Dimentional grid and then applies SCAN operator to collect recent network load information. The performance evaluation shows that proposed scheme offers high performance and low communication overhead which increase throughput and significantly improves system response time. Till now many techniques and solutions have been proposed and they works correctly with the selection of suitable parameters on specific application. To enhance the performance of the parallel and distributed system load balancing is vital parameter. To ensure the reliability and fault-tolerance to further improvement of system performance proposed technique can be implemented by introducing some fault tolerant mechanism with that.

The technique which is proposed is fully dynamic and adaptive according to the changes into the topology of network. This solution achieves high performance by providing perfect load balancing. As the fault-tolerance and reliability is also an important factor in order to improve the overall system performance. This can be enhanced by introducing a feature of fault tolerance in the technique which we shown above.

REFERENCES

[1] Ilias K. Savvas and M-Tahar Kechadi, "Dynamic Task Scheduling in Computing Cluster Environments", *IEEE Proceedings of the ISPDC/HeteroPar*' 04.

[2] Virginia Mary Lo. "Heuristic Algorithms for Task Assignment in Distributed Systems" IEEE Transaction on Advanced in Parallel and Distributed computing Proceeding, **37**(11)**:** November, (1988).

[3] Shen and Tsai. "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion", *IEEE Transaction on Advanced in Parallel and Distributed computing Proceeding*, **C-34**(3)**:** March, (1985).

[4] Isfaq Ahmad and Muhammad Kafil, "A Parallel Algorithm for Optimal Task Asignment in Distributed System" *IEEE Transaction on Advanced in Parallel and Distributed computing Proceeding*, March, (1997).

[5] Babak Taati & Michael Greenspan ."A Dynamic Load-Balancing Parallel Search for Enumerative Robot Path Planning", *Springer journal for Distributed and Parallel computing September*, (2006).

[6] F.M. Lopes. "Improving Load Balancing in a Parallel Cluster Environment Using Mobile Agents" *Springer HPNC,* (2001).

[7] Gabriele and Jin. "Comparing the OpenMP, MPI, and Hybrid Programming Paradigms on an SMP Cluster" *NAS Technical Report NAS*-03-019, November, (2003).

[8] Sandeep Singh. "Classification of dynamic load balancing strategies in a Network of Workstations", Department of computer science, *Khalsa College, Amritsar* (INDIA).

[9] Chao Huang. "Adaptive MPI", *Springer-Verlag Berlin Heidelberg,* (2004).

[10] Isaac Keslassy, Cheng-Shang Chang, Nick McKeown3, Duan-Shin Lee2, "Optimal Load Balancing" *Ieee transaction fo parallel computers*, (2005).

[11] Mark Baker. Cluster computing white . Technical report, University of Portsmouth, UK, December (2000).

[12] J. Basney et al. Utilizing widely distributed computational resources efficiently with execution domains. *J. of Computer Physics Comm.*, **140:** 246–252, (2001).

[13] Beowulf. http://www.beowulf.org.

[14] Guy E. Blelloch. Prefix sums and their applications. Technical report, School of Computer Science, Carnegie Mellon University, USA, (1990).

[15] T.L. Casavant and J.G. Kuhl. A taxonomy os scheduling in general-purpose distributed computing systems. *IEEE Trans. Soft. Eng.,* **14**(2)**:** 141–154, (1988).

[16] S. K. Das, D. J. Harvey, and R. Biswas. Parallel processing of adaptive meshes with load balancing. *IEEE Trans. On Parallel And Distributed Systems*, **12**(12)**:** 1269–1280, December (2001).

[17] M.K. Dhodhi et al. An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. *J. of Parallel and Distributed Computing*, **62:** 1338–1361, (2002).

[18] M. Maheswaran et al. Dynamic map of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, **59:** 107–131, (1999).

[19] http://www.intel.com/design/nettwork/products/ernet/.

[20] Michael J. Fischer and Michael Merritt. Appraising two decades of distributed computing theory research. *Journal of Distributed Computing*, **16:** 239–247, (2003).