A Distributed Shared Memory Cluster Architecture With Dynamic Load Balancing

MINAKSHI TRIPATHY AND C.R. TRIPATHY
Department of Computer Science and Engineering,
V.S.S. University of Technology,
Burla, Sambalpur, Odisha,
INDIA.
minakshiom@gmail.com

Abstract: - This paper proposes a distributed shared memory cluster architecture with load balancing. The architecture is based on dynamic task scheduling approach for distribution and assignment. It enhances the performance of communication across clusters for data access. The proposed dynamic load balancing model uses the concept of work stealing, which intelligently balances the load among different nodes. The work stealing consistently provides higher system utilization when many jobs are running with varying characteristics. This results in efficient use of the system. The performance analysis shows the proposed architecture to outperform the previously proposed distributed shared memory clusters in terms of scalability and efficiency.

Key-Words: - Block Data Layout, Data Locality, Task Distribution, Master-Slave Paradigm, Work Stealing.

1 Introduction

As The cluster computing can be described as a fusion of high performance microprocessors, high-speed networks and standard tools. A shared memory system, called a tightly coupled multiprocessor enables simple data sharing [1-2]. The shared memory system is portable and relatively easy to program since all processors share a single view of data with common memory. The communication between processors to a global physical memory can be as fast as the memory access. However, it suffers from lower peak performance, limited scalability and longer latencies in accessing the shared memory. A distributed memory system, called a multicomputer consists of multiple independent processing nodes with local modules connected via a general interconnection network [3-4]. These systems are scalable and the communication between processor or nodes requires explicit use of send/receive primitives. But, it becomes difficult to manage communication to achieve data distribution across the system. The distributed shared memory systems also known as distributed global address space (DGAS) combines the advantages of both the above said approaches [5-7]. It logically implements the shared memory model in a physically distributed memory system. The ease of programming, portability and abstraction of shared memory systems are preserved with the cost effectiveness of the distributed memory system. In literature, theoretical analysis has been made in the design of architecture to reduce data movement across the network and to reduce the execution time of the system [8-10].

In distributed system, the load balancing is done to distribute and schedule tasks between computers, processes, disk memories or other resources in order to get optimal resource utilization and to decrease the computing time [11-12]. If the workload is not properly balanced, a heavily loaded processor may be busy executing tasks while other processors sit idle, which degrades the system speedup. The dynamic balancing is based on redistribution of processes among the processors during execution time. Whenever load imbalance exists, the redistribution is performed by transferring tasks from heavily loaded processors to lightly loaded processors [13-16]. Process migration imposes a lot of processing efforts and therefore these systems do not support work stealing. The "work stealing" is an efficient approach to the distributed dynamic load balancing task as it is initiated by the idle processors. Here the idle processors select victim processors at random and attempt to steal work from them [17-20]. A node can be visualized as a queue and every arriving task is to be queued waiting for execution if the job arrival rate is more than the job's served rate [21]. The load balancing with work stealing has been studied with predictable neighborhood data references in [22-23]. In [5] analysis of parallel file system for distributed shared memory cluster system has been done. However the said work does not takes into consideration the intra and inter cluster communication. We extend the work stealing concept reported in [15] and [19] for dynamic load balancing to the proposed distributed shared memory cluster architecture.

The rest of the paper is organized as follows. In the Section2, notation and assumptions used in the paper are presented. The Section3 presents the proposed distributed shared memory cluster architecture with task assignment and distribution. A dynamic load balancing model with work stealing for the proposed distributed shared memory cluster system is proposed in Section 4. In the Section 5, the performance analysis with the proposed cluster system has been carried out and compared with previous systems. Finally, concluding remarks are provided in Section 6.

2 Notation & Assumptions

The following notation and assumptions are used throughout this paper.

Notation

A_{ccr}	:	Cost of reading a block from disk
A_{ccw}	:	Cost of writing a block to disk
TC	:	Total number of clusters
N	:	Number of nodes present in a cluster
P	:	Number of processes in a processor
NB	:	Number of disk blocks required for current task
NB_{WC}	:	Number of blocks within a cluster
NB_{BC}	:	Number of blocks to be transferred between clusters.
T_{WC}	:	Total time to transfer blocks within a cluster.
T_{BC}	:	Total time to transfer blocks between clusters
T_{accr}	:	Total time to read block data
T_{accw}	:	Total time to write block data
T_{access}	:	Total disk block access time
Texe	:	Total execution time
tmin	:	Minimum time to transfer a block
L	:	Current workload of a node
P_i	:	ith processor
L	:	Load of ith processor
N_i	:	Number of tasks assigned to Pi
W_i	:	Execution time of Pi to finish a task
C_i	:	Cost of stealing
R_i	:	Remaining tasks of Pi
E_i	:	Execution time of Pi to finish all tasks
T_i	:	Task transfer time of Pi
E	:	Total execution time of the system
P	:	Total number of processors present in the system
N_{all}	:	Total number of tasks
L_{sum}	:	Total workload of all processors
L_{avg}	:	Average workload of all processors
Eff	:	Efficiency of the system

Assumptions

1. All processors are heterogeneous in nature.
2. The interconnection network is message passing based.
3. Task queues are globally distributed.
4. Dequeues of tasks is maintained popping tasks from head in LIFO order.

3 Proposed System: Distributed Shared Memory Cluster Architecture

This section proposes a distributed shared memory cluster architecture based on dynamic data structure task scheduling. The principle of task assignment, block data layout and task distribution followed by an algorithm are presented in the subsequent sections. A distributed shared memory cluster system can be generally viewed as a set of nodes or clusters connected by an interconnection network. The proposed system architecture is shown in Figure1.

In the proposed system, each cluster node consists of a small-scale shared memory multiprocessor system and multiple clusters form a large-scale system. The proposed clustering architecture is beneficial for both small and large cluster systems. In the proposed clustering architecture, each cluster contains a local distributed shared memory (LDSM), an intercluster controller (ICCL), an intercluster cache (ICC), processors with private caches and a shared local bus. The private caches attached to the processors are inevitable for reducing the memory latency. The LDSM of each cluster is partially or entirely mapped to the global distributed shared memory (GDSM). Regardless of the network topology, a specific ICCL is required to connect a cluster into the system. The LDSM reduces memory contention and improves data locality. The ICC facilitates data sharing among the clusters utilizing data locality. It contains data that are usually referenced by the intra cluster processors. The local bus acts as an intra connection network among intra cluster processors, ICC and LDSM, while the global bus acts as an interconnection network among inter cluster nodes, inter cluster interconnection network and GDSM. Information about states or current locations of particular data blocks and the task scheduling queues are kept in the data structure (DS).

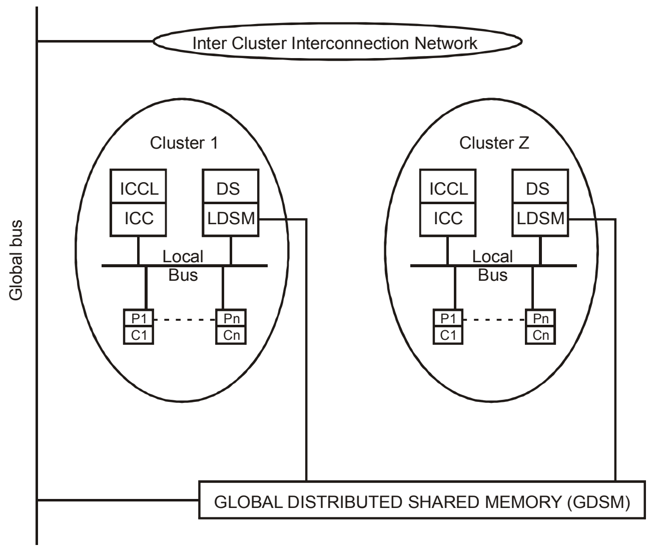

Fig.1: Distributed Shared memory Cluster Architecture

3.1 Task Assignment

In this subsection, the task assignment principles in the proposed architecture are described. While offering good scalability, a dynamic task scheduling approach using data structure creates as many concurrent tasks as possible to prevent processes from becoming idle. A task corresponds to a number of task instances since each task is created and inspected by all the processes on distributed shared memory systems. A task takes a number of inputs and writes to one or more outputs. Thus, the tasks stored in the task list keep information such as the input and output memory location. Whenever, two tasks access the same memory location and one of them is write, the system detects data dependence, and it stalls the successor till the previous task is finished. We consider the true dependence of RAW (Read After Write). If a task has t1 inputs and t2 outputs, then a number of t1+t2 task instances are created and distributed to different processes. Each task instance plays a role of "representatives" for the task's corresponding input or output. Task assignment is performed in two stages: block data layout and task distribution.

3.1.1 Block data layout

This subsection describes about the block data layout techniques. The system has a queue called 'task queue' in data structure. The task queue stores a pointer pointing to the corresponding ready task. The implementation of the task queue uses the block access indexed by block location [m, n]. The maximum number of tasks to be generated is constrained by the task queue size. The Block data layout is a technique used to improve memory hierarchy performance. In the block data layout, a matrix is divided into submatrices (or blocks) of size NB x NB. The proposed system uses 2D cyclic distribution method to map matrix blocks to different processes. The process block is used to map a 1D array of P processes to a 2D matrix block in a cluster. We assume that a process block has Pr rows and Pc columns where Pr x Pc=P. Let A [m, n] be a matrix block located at mth row and nth column of matrix A. Then A[m,n] will be mapped to process [m mod Pr, n mod Pc] through local bus. If the output of a task is A[m, n], then the task is assigned to process[m mod Pr, n mod pc].

3.1.2 Task distribution

The task distribution through its ICCL decides the data dependence for the blocks. In order to illustrate the task distribution let us consider an example. There are three operations to access the task queue: START, READ and WRITE. When the system finds a new task ti, it generates WRITE operation to put task ti at the end of the task queue. Before writing, it first scans the task queue to check if there exists a task tj to write into data x. Then the START operation searches for the task ti to READ data x in case of data dependence. If no tasks are present to write into ti's input, task ti becomes a ready task.

As an example, suppose a matrix of size 3x3 blocks is distributed to a 2x2 process block by 2D cyclic distribution where the processes P1, P2 executes a sequential program and generates a set of tasks t1, t2 and t3. Let the tasks read and write a block as below:

i) Task t1 reads and writes block1
ii) Task t2 reads block1 and writes block4
iii) Task t3 reads block1 and writes block7

Based on the status of task queues on P1 and P2, it is easy to find that t2 and t3 can be started simultaneously when task t1 is finished. Hence, a task ti is unable to execute for one of its parent task tj to finish. In this case task ti must be either tj itself or behind tj in task queue. Accordingly, the tasks t1,t2,…….,ti,tj,………tn,tn-1,…..t1 are assigned to the processes P1, P2,………..Pi,Pj,…….., Pn, Pn-1,…..P1.

3.2 Theoretical Analysis

In this subsection, we make an analysis for data communication and data accessing. The task queue in data structure makes an analysis after reception of a new task. It determines the particular processor and the corresponding cluster to which the task can be forwarded for task assignment and execution. It considers the cost of accessing data from the clusters through LDSM and GDSM.

The total time required to transfer the blocks between the nodes within a cluster through LDSM is expressed as follows.

$$T_{WC} = \sum_{i=0}^{sizeof(NB_{WC})} t_{\min} \quad (1)$$

The total time required to transfer the blocks between the clusters through GDSM is expressed as follows.

$$T_{BC} = \sum_{i=0}^{sizeof(NB_{BC})} t_{\min} \quad (2)$$

Now the total communication time to transfer the required blocks for the task is calculated as follows:

$$T_{comm} = T_{WC} + T_{BC} \quad (3)$$

The total execution time includes the time to read or write the blocks and to satisfy the task distribution. The time taken to READ data from the particular block is

$$T_{accr} = accr*NB \quad (4)$$

The time taken to WRITE the result from the particular block is

$$T_{accw} = accw*NB \quad (5)$$

Hence the total disk block access time is

$$Taccess = NB*(accr+accw) \quad (6)$$

Theorem 1: $T_{exe}=NB*(accr+accw)+2*N*T_{comm}$ (7)

Proof: After read and write operations, the whole disk block is stored in LDSM of a cluster. During execution, every node has to fetch the required data blocks. The total communication time spent on the node is $2*N*T_{comm}$ for both of the read and write operations. Hence the result for overall execution time can be expressed as $T_{exe}=NB*(accr+accw)+2*N*T_{comm}$.

3.3 Proposed Algorithm: DDST

This section proposes an algorithm for distributed shared memory cluster architecture with dynamic data structure task scheduling.

```
 For Each Node in N
 For Each Process in P
 Accept a new ready task from the task queue
 Assign task to process by 2D cyclic distribution method
For Each Node in $NB_{BC}$
 For Each Cluster in TC
   t=Calculate time to transfer blocks between
        clusters through GDSM
 End
$t_{min}$=min(t)
Update TBC
End
For Each Block in NBWC
For Each Node in N
 t=Calculate time to transfer blocks between nodes within clusters through LDSM
End
$t_{min}$=min(t)
Update TWC
End
Calculate $T_{comm}$, $T_{exe}$
Update private cache of nodes
End
Update ICCL and DS of the system
End
```

Theorem 2:Time Complexity of the algorithm is $O(NB)^3$

Proof: The time complexity of the proposed algorithm (DDST) is $O(NB)^3$ due to the three times of NB iterations for task assignment, data block communication and task execution.

4 The Proposed Dynamic Load Balancing With Work Stealing Model

This section proposes a dynamic load balancing model with work stealing for the proposed distributed shared memory cluster (Figure 2). The proposed model consists of three phases. Those are:

i) To get information about the slave node's status and send that information to the master node. The master estimates the performance of the slaves in terms of their computational latency and then makes an intelligent decision for the task assignment.
ii) After the estimation, the master distributes tasks on the basis of the performance in this distribution phase. The master collects information from the slave nodes to spawn new tasks based on decision made by work stealing.
iii) In the final phase, the master monitors the workload of the slaves and redistributes task whenever load imbalance is detected.

As the master is responsible for both the scheduling and distribution of task, the model allows slaves to compute data redundantly. This mechanism also makes the model tolerable to the failure of slaves.

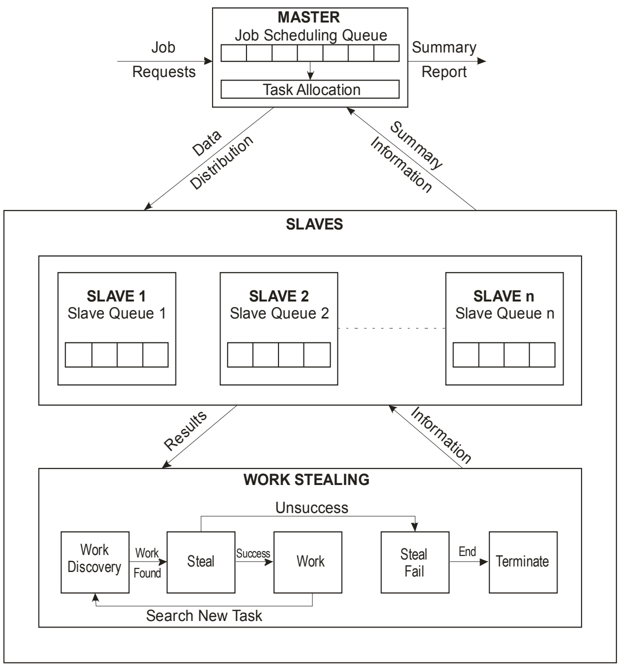

Fig.2: Dynamic Load Balancing Model for Distributed Shared Memory Clusters

4.1 Distributed Task Queues

This subsection describes the concept of distributed task queues. The proposed dynamic load balancing scheme can be expressed and understood through the use of task queues. A task queue provides a convenient parallel computation as a set of dynamic tasks. In the proposed model, the task queue first contains an initial set of tasks. In distributed systems, the distributed task queues store a set of task queues that are distributed across the process during the computation. In this work, focus is given on a 1:1 scheme where each process maintains its own task queue that allows for efficient local access. In a distributed shared memory clusters environment, the tasks execute with respect to the data stored in Global Distributed Shared Memory (GDSM). The GDSM enables the tasks to be executed on any process in any processor during the computation. The proposed model provides a global view of the physically distributed data. By storing distributed task queues in the GDSM, the ability to perform work stealing is gained.

4.1.1 Work Stealing

This subsection discusses the concept of work stealing. As already mentioned, the work stealing is a distributed dynamic load balancing scheme. Under the work stealing, each process maintains a double-ended queue or dequeue of tasks. The processes execute tasks from the head of their dequeue. When no work is available they steal tasks from the tail of another process's dequeue. The process that initiates the steal operation is called as *thief*. The process targeted by the steal is called as *victim*. The *thief* is responsible for initiating load balancing requests and the work stealing is a receiver initiated load balancing process. In the distributed shared memory clusters system, while performing a steal operation, the thief must first select its *victim*. Once a *victim* has been selected, the *thief* must then fetch data from *victim's* task queue to determine if work is available. If so, it transfers tasks from the tail of *victim's* queue to its own task queue. If the *victim* has no work available with it, then the *thief* selects a new *victim* at random and repeats this process until either work is found or global termination is detected. In order to determine the time of completion of the computation, the processes must actively detect that all the processes are idle and no more work is available. This is referred to as termination detection.

4.2 Theoretical Analysis

In this subsection, we provide a mathematical analysis for dynamic load balancing. In distributed shared memory clusters environment, each processor maintains a task queue with tasks that are ready to be executed. Whenever a node runs out of work, it becomes a *thief* and attempts to steal a task from another processor. If this *victim* processor has no available work in its task queue, the steal is unsuccessful and the thief processor makes new attempts to steal elsewhere until it is successful. In the proposed centralized dynamic load balancing for shared memory clusters, the current workload is calculated from the CPU, memory and network load status of nodes [2]. It is defined as the sum of CPU usage of task (Wcpu), memory occupied by tasks (Wmem) and amount of data transferred through network (Wnet).

$$L = Wcpu + Wmem + Wnet \quad (8)$$

After each node determines its current load, the master process distributes all the tasks among themselves. If a processor is overloaded, it is given fewer tasks so that the actual workloads with its task are evenly distributed. From the load on ith processor (L_i), The number of tasks assigned to the ith processor (P_i) is given by:

$$N_i = N_{all} * \left(\frac{1/L_i}{L_{sum}} \right) \quad (9)$$

where $L_{sum} = \Sigma\, Li$

Theorem 3: $E_i = R_i * (Wi + C_i)$ (10)

Proof: A processor must determine if it is under loaded before work stealing. So the execution time of a processor (P_i) to finish all its tasks is calculated from the execution time to finish a single task (W_i) with the cost of stealing (C_i) based on remaining tasks to execute (R_i). Here the cost of stealing tasks from the processor P_i is given by the ratio of task transfer time to that of execution time and is given by:

$$C_i = T_i / E_i \quad (11)$$

Hence the result for execution time to finish all the tasks of a processor is $E_i = R_i * (Wi + C_i)$.

A processor with a small C_i ratio is either over loaded or it can send tasks to others very quickly. Both indicate that P_i is the most suitable victim processor for work stealing.

We define that a processor P_i to be under loaded if

$$Ei < k * L_{avg} \quad (12)$$

Where, k is a constant for idleness of processors. The average of all workloads from the total number of processor present in the system is

$$L_{avg} = L_{sum} / P \quad (13)$$

Now, the efficiency of the DLBWS model is defined as

$$Eff = E_i / (E * P) \quad (14)$$

Where the total number of processors P, means the total number of processors allocated including the master, and E is the total execution time to finish their corresponding tasks present in all the processors of the distributed shared memory cluster system.

4.3 Proposed Work Stealing Algorithm: DLBWS

This subsection proposes an algorithm for the work stealing operation of dynamic load balancing in distributed shared memory clusters.

```
If Mq be the master processor task queue
 Sq_i be the slave processor task queue
 Vq be the victim slave node task queue and
 Tq be the thief slave node task queue
Initialize all the tasks to Mq
    For slaves from i=1 to n
    Collect load status of slaves Sq_i
    Distribute tasks from Mq to Sq_i
    End for
    While tasks are available in Sq_i{i=1..n}
    Select thief Ti and victim Vi
     Fetch work from victim's queue Vq.
    If  work Wi found
     Transfer tasks from Vq to Tq
     Steal and execute work Wi.
     Search for new task
    Else
     Steal Failed
    Terminate steal operation
    End If
End While
```

Theorem 4: Time Complexity of the algorithm is O (T1/P+Tp).

Proof: On a fixed number of processors P, the proposed work stealing scheduling algorithm completes job in O (T1/P+Tp) expected time, where T1 is the task execution time on one processor and Tp is the job execution time on P number of processors.

5 Performance Analysis

This section is devoted towards the performance analysis of the distributed shared memory clusters. The various performance measures of the proposed distributed shared memory cluster architecture with dynamic data structure task scheduling are analyzed and the results are compared with the previous works in [5]. The DDST algorithm is implemented in core java. The data structure programs are written in core java with subroutine to perform cache memory allocation and deallocation. It can be executed by the runtime system automatically. Similar to C++ programs, we use new and delete operators through user defined functions alloc_cache() and free_cache() to allocate and free cache memory. The special subroutines alloc_block() and free_block() to allocate and free the block are provided. The proposed DLBWS algorithm is implemented under matlab test bed. A program describing work stealing algorithm is run on the manager. It is responsible for running the proposed algorithm and gathering results from computing tasks. The manager assigns tasks to each worker by allotting data. The programs developed using the equations (8-16) are run on the slave nodes. It estimates the status or the load performance of each slave node. We take the mean value of execution times after ten executions of the program for final results and comparison. To validate the effectiveness of the proposed DLBWS model, we have compared the experimental results obtained with two previous works in [15] and [19]. The results of comparision of the execution time and efficiency are shown in the Figure 5-6.

5.1 Results & Discussions

This subsection provides the results and discusses on them. The cluster and block information in the form of tables are stored inside the data structure of distributed shared memory cluster architecture. Here, the Table 1 stores the cluster information. The information about transferring blocks from one cluster to another and the transferring blocks from one node to another node within a cluster are respectively stored in Table 2 and 3.

Disk access in our testing environment is fast enough, taking only 2ms to read or write a single KB data block. The program is assumed to be solved on 2, 4, 8 and 16 computer node clusters with distributed shared memory. To evaluate the effectiveness of the proposed approach, the performance of our proposed distributed shared memory cluster architecture for dynamic data structure task scheduling (DDST) is compared against two existing analytical methods NPIO and NIO taken from Successive Over Relaxation (SOR) [5].

The Table 4 shows the performance of DDST in terms of the execution time and the Table 5 gives scalability comparison of DDST with those of NPIO and NIO [5]. The Figure 4 shows how the execution time of DDST affects the overall system performance reducing execution time as compared to NPIO and NIO [5]. This establishes the efficiency of proposed (DDST) method over NPIO and NIO of SOR [5] in terms of scalability.

The Figure 5 compares the total execution time for SAMR [15], RAS [19] and the proposed DLBWS model. The execution time of DLBWS is reduced greatly. As it is clear from the Figure 2, the execution time decreases with increase of number of processors in the distributed systems. Again it can be noticed that the techniques using work stealing approach gives faster execution time as compared to previous methods [15][19]. It can be observed in the Figure6, that the efficiency of DLBWS is better as compared to that of SAMR [15] without work stealing and RAS [19] with work stealing. This establishes the superiority of the proposed DLBWS model over SAMR and RAS models in terms of speedup and efficiency.

TABLE 1: Cluster Information

Cluster Id	Id of Clusters
Node Id	Id of Nodes for the particular clusters
Block Id	List of all blocks with their Id in the cluster

TABLE 2: Inter Cluster Information

From	Cluster Id from which blocks are transferred
To	Cluster Id to which blocks are Block transferred
Id	List of all block Ids stored in the cluster
Time	Time taken to transfer blocks from one cluster to another

TABLE 3: Inter Cluster Information

From	Node Id from which blocks are transferred
To	Node Id to which blocks are transferred
Block Id	List of all blocks Ids stored in the cluster
Time	Time taken to transfer blocks from one node to another
Cluster Id	Cluster Id of the particular node

TABLE 4: Performance of DDST Method

N	NB(in kb)	T_{comm}(in ms)	T_{exe}
16	1000	500	20000
8	2000	1300	28000
4	4000	4500	52000
2	8000	12000	80000
1	10000	30000	100000

TABLE 5: Scalability Evaluation

Node	Execution Time (in ms)		
N	NPIO	NIO	DDST
16	40000	25000	20000
8	45000	30000	28000
4	75000	55000	52000
2	120000	110000	80000
1	200000	200000	100000

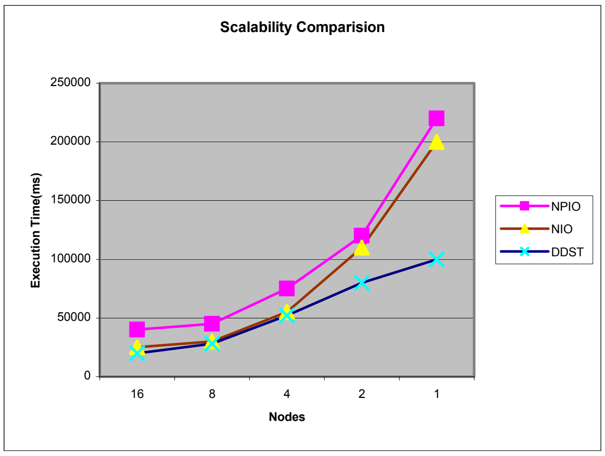

Fig. 4: Total Execution Time Vs Number of Nodes

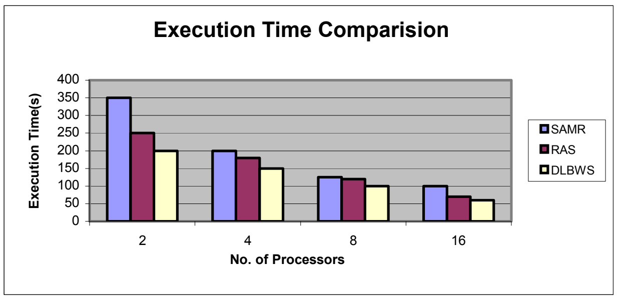

Fig. 5: Comparison of Execution Time Vs Number of Processors

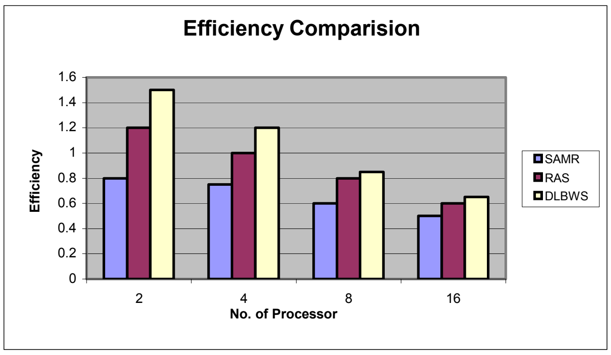

Fig. 6: Comparison of Efficiency Vs Number of Processors

6 Conclusion

In this paper, a distributed shared memory cluster architecture is proposed based on dynamic data structure task scheduling. The inter cluster caches, private processor caches and data structure in the linked- base type of cluster- based distributed shared memory architecture has an advantage of sharing data in a more effective manner. The work also proposes and illustrates the simple technique of work stealing that improves the execution time and the efficiency. When the machine has a large number of processors and has many jobs running on it, the idle processors steal tasks from the busy processors so that every processor can be busy all the time. When the master schedules task inappropriately, it tries to balance the loads with additional stealing. As expected, the number of stealing increases as the number of processors grows and the environment becomes more dynamic. Based on the results of comparision with the existing methods, we conclude the proposed architecture to have a better performance that reduces the communication and idle time. It also requires less space in stable storage and obtains faster execution time.

References:

[1] Minakshi Tripathy and C.R.Tripathy, Design and analysis of a Dynamically Reconfigurable Shared Memory Cluster*, International Journal of Computer Science and Network Security,* Vol.10, No.9, Sept 2010, pp 145-158.

[2] Minakshi Tripathy and C.R. Tripathy, Centralized Dynamic Load Balancing Model for Shared Memory Clusters, *Proceedings of the International Conference on Control, Communication and Computing*, Feb 18-20, 2010, pp 173-176.

[3] Bill N. and Virginia L., Distributed Shared Memory: A Survey of Issues and Algorithms, *Journal Computer - Distributed computing systems,* Vol. 24, No. 8, August 1991, IEEE Computer Society Press.

[4] S. Zhou, M. Stumm, D. Wortman and K. Li, Heterogeneous Distributed Shared Memory, *IEEE Transactions on Parallel and Distributed Systems,* Vol.3, No. 5, Sept 1992, pp 540-554.

[5] Su-Cheong Mac, Ce-Kuen Shieh and Jyh-Biau Chang: Design and analysis of a parallel File system for distributed shared memory systems, *Journal Of System Architecture*, Vol. 45, No. 8, 1999, pp 603-617.

[6] Der-Lin Pean, Chao-Chin Wu, Huey-Ting Chua and Cheng Chen, Design of a scalable multiprocessor architecture and its simulation, *The Journal of Systems and Software*, Vol.58, No. 2, 2001, pp 135-152.

[7] Jyh-Chang Ueng, Ce-Kuen Shieh, Tyng-Yue Liang and Jyh-Biau Chang, Proteus: an efficient runtime reconfigurable distributed shared memory system, *The Journal of Systems and Software*, Vol. 56, No. 1, 2001, pp 247-260.

[8] Fengguang Song, Asim YarKhan and Jack Dongarra, A look at scalable dense linear algebra libraries, Proceedings of the *International conference on Scalable High Performance computing,* Apr 1992, pp 372-379.

[9] P. Sammulal and A. Vinaya Babu, Enhanced Communal Global, Local Memory Management for Effective Performance of Cluster Computing, *IJCSNS International Journal of Computer Science and Network Security*, Vol.8, No.6, June 2008, pp 209-215.

[10] S. Dimitrios and Nikolopoulos, Quantifying contention and balancing memory load on hardware DSM multiprocessors, *Journal of Parallel and Distributed Computing,* Vol. 63, No. 9, 2003, pp 866–886.

[11] Kashif Bilal, Tassawar Iqbal, Asad Ali Safi and Nadeem Daudpota, Dynamic Load Balancing in PVM Using Intelligent Application, World academy of Science, *Engineering and Technology*, Vol. 5, May 2005, pp 132-135.

[12] James Dinan, Stephen Olivier, Gerald Sabin, Jan Prins, P. Sadayappan, and Chau-Wen Tseng, Dynamic Load Balancing of Unbalanced Computations Using Message Passing, *Proceedings of 21st International Parallel and Distributed Processing Symposium* , March 26-30, 2007, pp 1-8.

[13] Ahmad Dalal'ah, A dynamic Sliding Load Balancing Strategy in Distributed systems, *The International Arab Journal of Information Technology*, Vol. 3, No. 2, 2006,.pp: 178-182.

[14] Z. Khan, R. Singh, J. Alam and R. Kumar, Performance Analysis of Dynamic Load Balancing Techniques for Parallel and Distributed Systems, *International Journal of Computer and Network Security*, Vol 2, No. 2, 2010, pp: 123-127.

[15] Zhiling Lan, Valerie E. Taylor and Greg Bryan, Dynamic Load Balancing of SAMR Applications on Distributed Systems, *Proc. Conference on High Performance Networking and Computing of the 2001 ACM / IEEE conference on Supercomputing*, Nov 10-16, 2001, pp:979-993.

[16] Kalim Qureshi and Masahiko Hatanaka, An introduction to load balancing for parallel raytracing on HDC systems, *Current Science*, Vol. 78, No. 10, 2000, pp 818-820.

[17] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha, Scalable Work Stealing, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Nov 14-20, 2009, pp-45-54.

[18] Stephen Olivier and Jan Prins, Scalable Dynamic Load Balancing Using UPC, *Proceedings of the 2008 37th International Conference on Parallel Processing,* Sept 8-12, 2008, pp: 123-131.

[19] Yangsuk Kee and Soonhoi Ha, A Robust Dynamic Load Balancing Scheme for Data Parallel Application on Message Passing Architecture, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Jul 1998, pp: 974-980.

[20] Minakshi Tripathy and C.R.Tripathy, Dynamic load balancing with work stealing for distributed Shared memory Clusters, *International Conference on Industrial Electronics, Control & Robotics*, 27- 30 Dec 2010, pp 1-5.

[21] Minakshi Tripathy and C.R. Tripathy, Design and Analysis of a Distributed Shared Memory Cluster Architecture based on "Dynamic Data Structure Task Scheduling", *Proceedings of the International Conference on Electronic Systems,* 07-09 Jan 2011, pp 395-398.

[22] Pangfeng Liu and Chih Hsuae Yang. Locality-Preserving dynamic Load Balancing for Data Parallel Applications on Distributed Memory multiprocessors, *Journal of Information*

Science and Engineering, Vol.18, No. 6, 2002, pp: 1037-1048.

[23] Kunal Agarwal, Yuxiong He and Charles E. Leiserson, An Empirical Evaluation of Work Stealing with Parallelism Feedback, *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems,* Jul 2006, pp: 19-27.

Ms. Minakshi Tripathy received the degree of B.Sc. (PCM), M.Sc. (Statistics) and MCA from Sambalpur University. She has done 'A' level course from DOEACC, New Delhi. She is currently a Ph.D. (Computer Science) student at Sambalpur University, Burla, Orissa. She has publications in five different international conferences and four different international journals. Her research interest includes shared memory, cluster computing, load balancing and fault tolerance.

Prof. (Dr.) C.R. Tripathy received the B.Sc. (Engg.) in Electrical Engineering from Sambalpur University and M. Tech. degree in Instrumentation Engineering from I.I.T., Kharagpur respectively. He got his Ph.D. in the field of Computer Science and Engineering from I.I.T., Kharagpur. He has more than 50 publications in different national and international Journals and Conferences. His research interest includes Dependability, Reliability and Fault–tolerance of Parallel and Distributed system. He was recipient of “Sir Thomas Ward Gold Medal” for research in Parallel Processing. He is a fellow of Institution of Engineers, India. He has been listed as leading scientist of World 2010 by International Biographical Centre, Cambridge, England, Great Britain.