Memory Management Technique for Paging on Distributed Shared Memory Framework

Prof. Bal Gopal[1], Rizwan Beg[2] , Pankaj Kumar[3]

[1,2]Integral University, Lucknow, India
bal.gopal@rediffmail.com, rizwanbeg@gamail.com
[3]Sahara Arts & Management Academy, Lucknow, India
pk79jan@rediffmail.com

ABSTRACT

Distributed Shared Memory (DSM) System has become popular paradigm in distributed system. As DSM system involves moving of data from on node to another node which is in typical network, so performance is the important criteria of designing DSM system. A DSM system can be designed as paged-based, shared variable based and object based. In paged-based DSM system the unit of data sharing is the memory page.
In this paper we describe the memory management technique for paging of DSM framework. Implementing DSM framework with paging scheme leads to false sharing and high cost associated with virtual memory operation. The paper discusses the effect of granularity and finds the solution of false sharing. The paper also analysis the different overheads of DSM framework with respect to the page-size and virtual memory operation.

KEYWORDS

Distributed Shared Memory, page- base DSM.

1. INTRODUCTION

DSM system represents a successful hybrid of two parallel computer classes i.e. shared memory and distributed memory. It provides the shared memory abstraction in system with physically distributed memories and consequently combine the advantages of both approaches. DSM system is the memory system that is physically distributed but logically implements a single shared address space [5, 8, 11].

A software DSM implementation emerged in the mid 1980s and is to provide the shared memory paradigm to the programmer on top of message passing. Generally, this can be achieved in user-level, run-time library routines, the operating system, or a programming language. Larger grain sizes on the order of a kilobyte are typical for the software DSM implementations because their DSM management is usually supported through virtual memory [2, 3]. Thus, if the requested data is absent in local memory, a page-fault handler will retrieve the page from either the local memory of another node, or disk of requesting node or another node. Software support for DSM is generally more flexible than hardware support and enables better tailoring of the consistency mechanisms to

the application behavior but usually cannot compete with hardware implementations in performance. IVY [1] was one of the first proposed software DSM solutions. Software DSM systems can be split into three classes: page-based; variable-based; object-based. In each of these approaches our concern is where, and how, transparency of remote access is introduced [10, 11]:

Page-based - uses the memory management unit (MMU) to trap remote access attempts.

Variable-based - requires custom compilers to add special instructions to program code in order to detect remote access

Object-based - special programming language features are required to determine when a remote machine's memory is to be accessed.

Page-based DSM closely follows the shared memory in multiprocessor sphere. The entire address space (spread over all the nodes) is divided into pages. Whenever the virtual memory manager (VMM) finds a request to an address space which is not local, it asks the DSM manager to fetch that page from the remote machine. Such kind of page fault handling is simple and similar to what is done for local page faults. To improve performance, replication of the pages is done so that same page does not have to be transferred again and again. This especially improves performance for read only pages. Keeping multiple copies of same page leads the issues of consistency between these copies. If pages are not replicated, achieving sequential consistency is not difficult. But for replicated copies, page-based DSM generally follows the same scheme as coherency schemes for distributed shared memory. Any DSM system has to rely on the message passing technique across the network for data exchange between two computers. The DSM has to present a uniform global view of the entire address space (consisting of physical memories of all the machines in the network) to a program executing on any machine. A DSM manager on a particular machine would capture all the remote data accesses made by any process running on that machine. A design of a DSM would involve making following choices:

- What kind of consistency model should the system provide?
- What kind of coherency protocol should be used?
- Where, with respect to the Virtual Memory Manager does the DSM operate?
- What should be the granularity of the shared data?
- What should be the size of page?

2. PREVIOUS WORK AND PROBLEM SPECIFICATION

In DSM studies, some researches of Distributed system have discussed the problem of workload distribution. Liang[17] describe the job of work distribution in two phase: Migration phase and Exchange phase. Peris [18] analyzed the influence of physical memory size on system performance

and shows that wrong workload distribution will increase of memory access and degrade system performance. Zhang[19] proposed making use of CPU load index and memory load index for workload distribution.

A DSM framework [12] is designed by Bal Gopal and Pankaj Kumar in which they use the concept of data replication. The DSM framework is designed with help of processor consistency model and write-update coherency protocol. Multiple Reader and Multiple Writer (MRMW) algorithm is used for the proper implementation of replication [4, 6, 7]. The said framework solves the problem of first two choices of DSM system i.e the kind of consistency model and coherency protocol as discussed in section 1. The attractive feature of the framework is that it allows the hardware to be used to help in the maintenance of consistency and provide easier and simpler synchronization. It focuses on improvisation of the parallelism using processor consistency model and write-update coherence protocol. It concentrates on the broadcasting of multiple writes to different nodes of DSM system and performance cost.

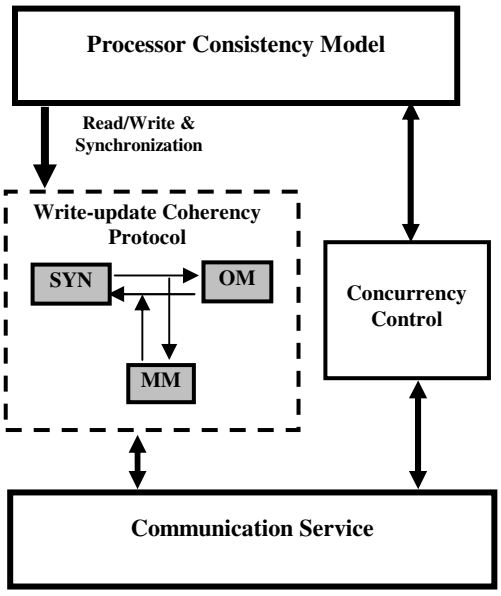

Figure 1: The framework overview (**SYN**: Synchronization, **OM**: Ownership Management, **MM**: Memory Management)

The main disadvantage of the framework is that it does not state the effects of size of page to the system and the problem of false sharing. So in this paper we will discuss on granularity, size of page, memory overhead and operation of DSM framework with respect virtual memory. All these will be discussed with respect to the DSM framework designed with processor consistency model and write-update coherence protocol.

3. GRANULARITY

Granularity refers to the size of a data unit in which it exists in the distributed shared memory. This is an important decision which essentially governs the design of a DSM. The immediate successor of shared memory from multiprocessor world would have a page as the unit for data transfer. Granularity describes the size of the minimum unit of shared memory [13, 14, 15]. In the said DSM framework it is the page-size. The protocols and hardware that are used in framework to propagate updates will have an influence on the choice of granularity. In the framework the efficiency is maximized by making the granularity into a multiple of the size used by these. The framework is designed to optimize the passing of pages around the network so that the page-size will be determined.

3.1 Execution Granularity

Execution granularity is the amount of execution a process has to do between synchronization and communication point in a multiprocess computation. Execution Granularity Ratio (EGRatio) can be calculated by time spent in the execution (T_E) and the spent in the requesting data for the execution (T_R) i.e.

$$EGRatio = \frac{T_E}{T_R}$$

The speedup of any implementation is good if it spends more time on execution rather than requesting for data for requesting. So for good speed T_E should be greater than T_R. or we can say that for good speed the EGRatio should be greater than 1 i.e.

$$EGRatio > 1$$

If EGRatio is less than 1 it means that system spent more time on requesting the data. Hence it will have slow speed.

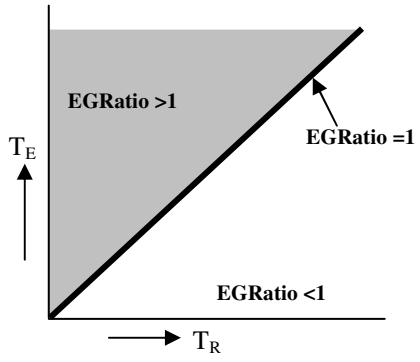

Figure 2. EGRatio requirement.

Fig. 2 shows the plot for EGRatio. The shaded region shows the region where the EGRatio is greater than 1. The systems differ with each other in size of data to be transferred, speed of communication medium and other overhead of transfer. So for a particular implementation the EGRatio of the systems should fall in the shaded regionSo for perfect EGRatio it is good to calculate the execution requirement for the framework by matching the minimum communication time for requiring data.

3.2 Data Granularity

Data granularity deals with amount of data process during the execution phase. It can be related to the amount of data exchanged between nodes at the end of execution phase as it is data that will be processed in the execution phase. In the framework the amount of data exchanged between nodes is usually a multiple of the physical page-size. The system using paging, regardless of the amount of sharing, the amount of data exchanged between nodes is usually a multiple of the physical page-size of the basic architecture. Problem arises when system that consists very small data granularity are running on system that support very large physical pages. If the shared data is stored in contiguous memory location then most data can be stored in few physical pages. As a result the system performance degrades as the common physical page thrashes between different processors. To overcome with this difficulty the framework partitioned the shared data structure on to disjoint physical pages.

4. FALSE SHARING

A process that accesses one variable is very likely to access other variables located near to it in memory. However this consideration is often overridden by the fact that a large page-size increases the possibility of false sharing. This occurs when a number of processes seek to access unrelated data items that happen to be located on the same page.

False sharing particularly occurs if a page contains a key variable that is accessed very often, this will mean that processes will experience considerable delays in accessing any other variables located on the same page. This problem has been moderated somewhat for page-based systems by adopting a mode of operation where key variables are stored alone on one page, so that accessing them will not cause any difficulties in accessing other unrelated data items. For instance, two user-defined objects are allocated on the same page and two processors P1 and P2 are accessing the page concurrently. As a result, the page has to travel across the network each time P1 wants to write in it when it is being held by P2 or vice versa. This problem occurs mainly because of the difference between the programmer's view of sharing and sharing at the DSM level and can lead to the thrashing problem. *Thrashing* occurs when there are several processes that have frequent needs for access to the same page, this will mean that ownership will be difficult to determine and the page will be transferred back and forth across the network at a very high frequency. These processes will spend much of their time waiting for the page to be returned to them and so their performance will suffer.

The solution to thrashing it is to allow several copies of the page to exist simultaneously on a number of machines and to institute some means of ensuring that they all display a consistent view of the data that they contain.

False sharing is a particularly serious problem for DSM systems for two reasons:

1. The consistency units are large, so false sharing is very common and
2. The latencies associated with detecting modifications and communicating are large, so unnecessary faults and messages are particularly expensive.

To overcome with the problem of false sharing the framework allows multiple processes to write concurrently into a shared page with the updates being merged at the appropriate synchronization point in accordance with the definition of processor consistency and write-update coherence protocol.

5. OVERHEAD

The overhead in DSM framework is the large amount of communication that is required to maintain consistency. The network hardware for the DSM framework is optimized for passing pages across the network. It divided the address space into pages, each page being located in the local memory of one machine in the system. When a processor tries to access an address in a page that is not local the DSM software identifies the page, locates it and fetches it from the machine it is currently stored on. Thus the virtual memory consists of pages, stored in local memory of a machine on the network, that migrate across the network as different processors need to access their contents. This migration is required for read accesses as well as writes accesses, because the only way data can migrate across the network is if the page it is located on is transferred to another machine [10, 14, 16].

We have categorized the times of DSM framework into three cases for the calculation of overhead.

1. Time taken for the servicing of page fault.
2. Includes updating state information for shared segment and coherence maintenance.
3. Network communication between nodes and a page transfer.

Case 1: As the framework is using Multiple Read Multiple Write (MRMW) algorithm for the replication. It improves system throughput by having all processing nodes synchronously. The throughput of the framework depends on the following two factors:

a. Average overhead of parallel context switches between the processes (T_{PCS}).

b. Average effective overhead of page fault (T_{EPF}).

The context switch occurs when the execution switches from one process to other process. If T_{PF} is the time to service the local page fault and the N_{PF} is the number of page fault then the total time overhead for servicing of page fault will be

$$T_{TPF} = N_{PF} \times T_{PF}$$

So average effective overhead of page fault will be

$$T_{EPF} = \frac{T_{TPF}}{P}$$

Where 'P' is the number of processor. The average fraction of page fault service time overhead will be given by

$$F_{PF} = \frac{T_{EPF}}{T_{PF}}$$

Case 2: The framework uses write-update coherence protocol for the consistency maintenance. The protocol supports multiple copies of data to be transferred to different nodes. So the total number of bits to be transferred will depend upon the number of processor, number of bit in page, and the number of cache in the framework. Let

P = number of processor

B = number of words per page

W = number of bits per word

M = number of blocks in each cache of the B memory module

C= number of cache in each of the P caches

So the total number of bits in the system to transfer

$$D = PBW\ (M + C)$$

If N_X is the number of bits dedicated to coherency function for maintaining the consistency and coherence protocol then the corresponding overhead will be

$$O_X = \frac{N_X}{D}$$

Case 3: The network communication overhead can be expressed as a sum of two types of cost. *Fixed cost* consists of the overhead associated with the virtual memory subsystem and the cost of

sending a data request to the data server. *Variable cost* consists of cost of sending the data to requester.

Fixed cost depends upon the virtual memory overhead (VMO), data request cost (DRC) and the page-size (PS). The virtual memory cost and the data request depends upon the size of data to be transfer to other nodes. So the fixed cost can be calculated by

$$FixedCost = \frac{VMO + DRC}{PS} \quad (1)$$

It implies that if the framework acquires high virtual memory overhead and high data request cost then the fixed cost can be minimized by increasing page-size.

The variable cost controls the latency of data and depends upon server processing cost (SPC), page-size (PS) and media bandwidth (MB). It can be calculated by

$$VariableCost = (SPC) \times PS + \frac{PS}{MB} \quad (2)$$

It implies that for getting low latency the page-size should be kept small. The overhead for the network communication can be calculated by

$$NCO_{verhead} = FixedCost + VariableCost \quad (3)$$

For the proper implementation of distributed shared memory the data transfer cost should be minimum. If there are more data have to be transfer then the page-size should be kept small as the variable cost increases rapidly otherwise the page-size should be kept large. So for minimizing the overhead it is suggested to take variable page-size.

6. PAGE-SIZE

The unit of data managed and transferred by DSM is a data block which is call DSM page. In this DSM framework we have taken heterogeneous page system which supports different virtual memory page-size, so choosing a size for DSM page become an important issue. If we choose small virtual memory page-size for DSM page then multiple small virtual memory pages fit exactly in one DSM page, hence they can be treated as group of page-fault. If we choose large virtual memory page-size for DSM page then more data than necessary data may be moved between hosts. The one thing which also be considered that when data items in the same DSM page are being updated by multiple hosts at the same time, causing large number of page transfer among the host without much progress in the execution of the application. This causes false sharing, where non overlapped regions in the same page are shared and updated by different host, causing repeated page transfer.

So for proper calculation of page-size we have concentrated on the network communication overhead. Therefore by the equation 3 we get network communication overhead (NCO)

$$NCO = \frac{VMO + DRC}{PS} + (SPC) \times PS + \frac{PS}{MB}$$

Let VMO+DRC='A' and SPC+1/MB='B' the total overhead will be

$$NCO = \frac{A}{PS} + B.PS$$

As the page-size is not taken fixed and is dependent on the overhead of the network communication therefore differentiating NCO with respect to PS we will get

$$\frac{dNCO}{dPS} = -\frac{A}{PS^2} + B \qquad (4)$$

$\frac{dNCO}{dPS}$ will give the network communication overhead with respect to the page-size. The network communication overhead with respect to the page-size is constant so let it is '1' then equation 4 will be

$$1 = -\frac{A}{PS^2} + B$$

So the page size will be

$$PS = \sqrt{\frac{A}{B-1}}$$

Putting the value of 'A' and B we will get

$$PS = \sqrt{\frac{VMO + DRC}{(SPC + \frac{1}{MB}) - 1}}$$

And finally

$$PS = \sqrt{\frac{(VMO + DRC) \times MB}{(SPC - 1) \times MB + 1}} \qquad (5)$$

By eq. 5 we can say that the page-size of the framework depends upon virtual memory overhead (VMO), data request cost (DRC), server processing cost (SPC) and media bandwidth (MB). As the virtual memory overhead and the data request cost depends upon the size of data to be transfer and the media bandwidth is constant for any data size, the page-size will be

$$PS = \sqrt{\frac{VMO + DRC}{SPC}}$$

If the server processing cost is small then the page-size will also be small, but the page size will be large if virtual memory overhead or data request cost is high. As the data

7. DISCUSSION

As the framework uses memory page as the unit of data sharing, the entire address space (spread over all to the processor) is divided into pages. Whenever the virtual memory manager (VMM) finds a request to an address space which is not local, it asks the DSM manager to fetch that page from the remote machine. Such kind of page fault handling is simple and similar to what is done for local page faults. So for improving performance, the framework suggests for doing *replication* of the pages so that same page does not have to be transferred again and again. Keeping multiple copies of same page leads to the issue of consistency between these copies. This consistency issue is maintained by write-update coherence protocol. This is done by updating all processor about the write operation. The page manager maintains a list or copy set telling which processors hold which pages. When a page must be invalidated, the old owner, new owner, or page manager sends a message to each processor holding the page and waits for an acknowledgment. When each message has been acknowledged, the invalidation is complete. In this DSM framework, it may happen that a page is needed but that there is no free page frame in memory to hold it. When this situation occurs, a page must be ejected from memory to make room for the needed page. Two sub problems immediately arise: which page to eject and where to put it. In the context of framework, a replicated page that another process owns is always a prime candidate to eject because it is known that another copy exists.

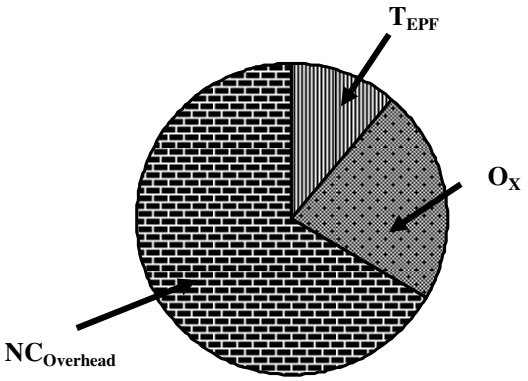

Figure 3: Different timing overheads

In section 5 we have described the timing overheads of distributed shared memory with respect to page fault, coherency scheme and network communication overhead. When data is transferring in the system it has been observed that the timing overhead in network communication is very much high with respect to the page fault servicing time as well as coherency maintenance time. In general

it is found that the network communication overhead is more than triple to the sum of page fault servicing time and the coherency maintenance time. A graphical representation of all these timing overheads is shown in fig. 3. The bricked region of the circle is showing the network communication overhead. The dotted region is showing the coherency maintenance. The reason behind the selection of $NC_{Overhead}$ for page-size is that it affects a lot on virtual memory operation.

8. CONCLUSION

In this paper we have described the different overheads of distributed shared memory framework with respect to paging. For this we have taken DSM framework which is implemented with processor consistency and write-update coherence protocol. In section 3 & 4 various issues related to granularity and false sharing is discussed. The paper describes the effect of granularity and find out the solution false sharing. But the paper mainly concentrated on the paging concept. In this regard the paper study the page fault overhead, coherency overhead and network communication overhead at the time of paging. The result of study points out that the page size of Distribute Shared Memory framework may vary according to the total overheard of the system. So the paper proposed the size of page of DSM system based on the total overhead.

ACKNOWLEDGEMENT

I would express my gratitude to **Shri Sudhir Kumar**, Head, Sahara Arts & Management Academy, Lucknow for his sincere support and motivation. I can not forget the cooperation and guidance of Dr. Brijesh Khandelwal, Dean Academics, SAMA, Lucknow. I would also thank to my entire colleague who gave me a lot of encouragement for writing this paper.

REFERENCES

[1] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. ACM Transactions on Computer Systems, 7(4):321–359, November 1989.

[2] Changhun Lee "Distributed Shared Memory" Proceedings on the 15th CISL Winter Workshop Kushu, Japan, February 2002.

[3] Yvon Jégou "Implementation of Page Management in Mome, a User-Level DSM" INRIA November 2003.

[4] Bal Gopal, Pankaj Kumar “Perfect Memory Consistency Model for Improving Parallelism in DSM System” Proceedings of Third international conference on information processing (ICIP-09), 7-9 Aug 2009.

[5] Sarita V. Adve, Vijay S. Paiand Parthasarathy Ranganathan “Recent Advances in Memory Consistency Models for Hardware Shared Memory Systems” Proceedings of the IEEE, VOL. 87, NO. 3, MARCH 1999, pp. 445-455.

[6] Bal Gopal and Pankaj Kumar “Framework for Improving Parallelism by Write-Update Coherence Protocol in Distributed Shared Memory System” IJRTE, Issue. 1, Vol. 2, June 2009 pp. 201-204

[7] Bal Gopal, Pankaj Kumar "Maintaining Memory Consistency with Coherence Protocol in DSM System" IEEE International Advance Computing Conference (IACC March 2009).

[8] Jelica Protic, Velijko Milutinovic "Distributed Shared Memory: Concepts and System" Summer 1996 IEEE.

[9] Yang-Suk Kee, Jin-Soo Kim, Soonhoi Ha "Memory Management for Multi-Threaded Software DSM Systems" Parallel Computing 2003.

[10] B. Buck and P. Keleher. Locality and performance of page- and object-based DSMs. In Proc. of the First Merged Symp. IPPS/SPDP 1998), pages 687– 693, 1998.

[11] Michael Kistler, And Lorenzo Alvisi "Improving The Performance of Software Distributed Shared Memory With Speculation" IEEE Transactions on Parallel And Distributed Systems, Vol. 16, No. 9, September 2005.

[12] Bal Gopal and Pankaj Kumar, " DSM Framework with Processor Consistency and Write-update Coherence Protocol: Implementation" International Journal of Computational Intelligence Research ISSN 0973-1873 Volume 5, Number 4 (2009), pp. 469–484.

[13] Mainak Chaudhuri, Mark Heinrich, Chris Holt, Jaswinder Pal Singh, Edward Rothberg and John Hennessy, "Latency, Occupancy, and Bandwidth in DSM Multiprocessors: A Performance Evaluation" IEEE Transactions on Computers, Vol. 52, No. 7, July 2003.

[14] Y. Zhou et al. "Relaxed Consistency and Coherence Granularity in DSM Systems: A Performance Evaluation," Proc. Sixth Symp.Principles and Practice of Parallel Programming, pp. 193-205, June1997.

[15] Bernd Mohr, "Introduction to Parallel Computing", NIC Series, Vol. **31**, ISBN 3-00-017350-1, pp. 491-505, 2006.

[16] Brian T. Gold, Babak Falsafi, and James C. Hoe "Tolerating Processor Failures in a Distributed Shared-Memory Multiprocessor" Computer Architecture Lab at Carnegie Mellon (CALCM) Technical Report 2006.

[17] Tyng-Yen Liang, Ce-Kuen Shieh, Deh-Cheng Lin, "Scheduling Loop Application in Software Distributed Shared Memory System" , IEICE Transaction on Information and System, Vol. E83-D no.9 PP. 1721-1730 sep 2000.

[18] Vinod G.J. Peris, Mark S. Squillaute and Vijay K. Naik, "Analysis of the Impact of Memory in Distributed Parallel Processing System", Proceeding of the 1994 ACM SIGMETRRICS conference pp 5-18 feb 1994.

[19] Lio Xiao, Songqing chen and Xiaodong Zhang, "Dyanimcs Cluster Resource Allocation for Jobs with Known and Unknown Memory Demand, IEEE Transactions on Parallel and Distributed System, Vol.13 No.3 pp. 223-240, march 2002.

Author

1. Bal Gopal is with integral university Lucknow, India. He is currently Dean, Computer Application. He received B.Tech and M. Tech Degree from IIT, Kanpur, India. He is life time member of FIETE and also Life member of Institution of Engineers (India). He guided more than 34 of M.Tech thesis.

2. Dr. Rizwan Beg is currently working as Working as Director in Dr.Z H Institute of Technology & Management Agra. He received his Doctor of Philosophy (PhD) in computer Science & Engineering from Integral University, Lucknow. He has done M.Tech. in Computer Science & Engineering from U.P. Technical University, Lucknow and B.Tech in Computer Sc. & Engineering from Gulbarga University, Gulbarga. His Area of Expertise is Software Engineering and Software Project Management. Dr. Rizwan Beg has published papers in various National/International Journals and IEEE proceeding publication in the area of "Software Engineering". Currently 10 students have been enrolled as research scholar under his supervision. Dr. Beg has Guided more than 70 projects of Under Graduate and Post Graduate. He is member of Computer Society of India (CSI), International Association of Engineers (IAENG).

3. Pankaj Kumar is currently working as Lecturer (Sr. Scale) in Sahara Arts & Management Academy, Lucknow. He received **MCA** degree in 2001 from Rohilkahnd University, Bareilly, India. His research interests are Parallel Computing, Memory Architecture of Parallel Computer and Distributed Computing. He is pursuing Ph.D in Computer Application. Many of the valuable research papers of Mr. Pankaj Kumar have been published in various national/international journals and IEEE proceeding publication in the area of "Parallel Computing". He is member of Computer Society of India (CSI).